# Midterm Study Case

## Advance in Data Sciences and Architecture

TEAM MEMBERS (Team 1):

Byron Kiriibwa

Wenqi Cui

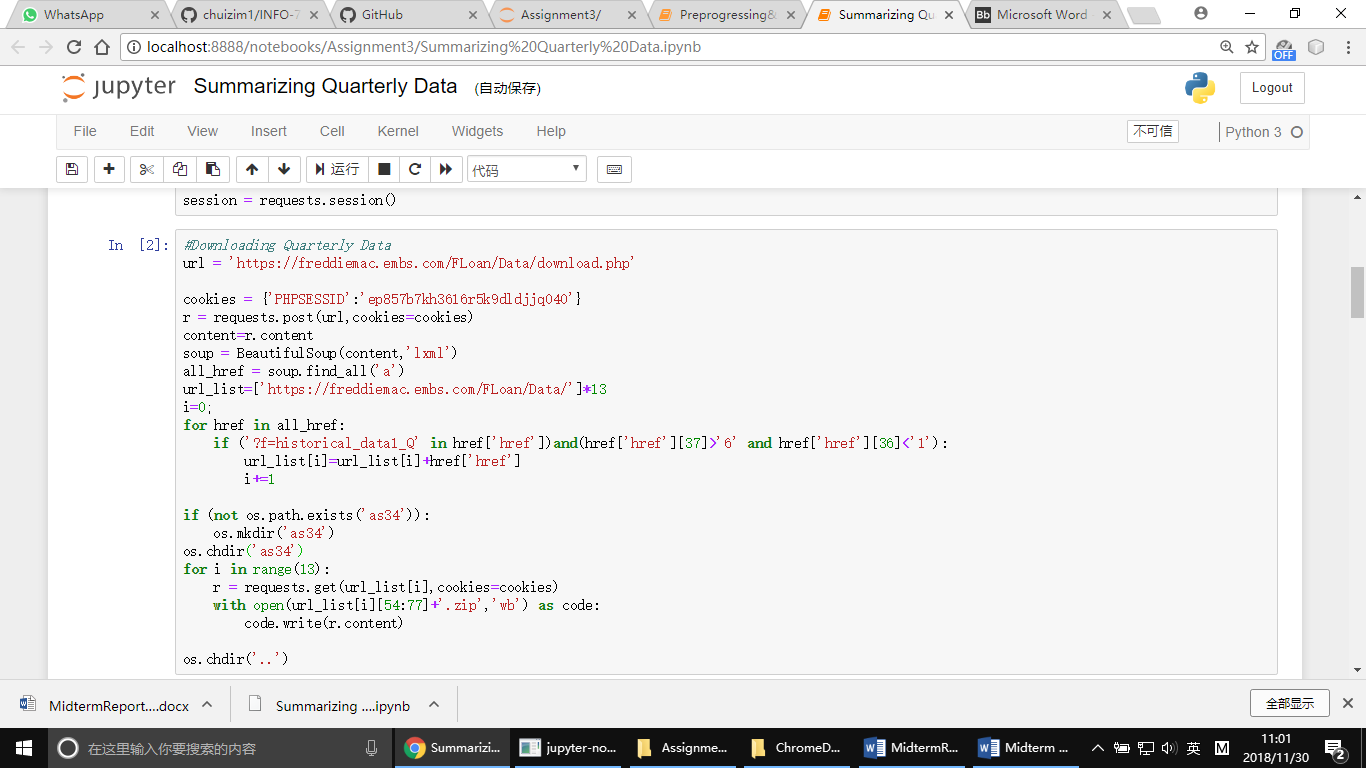
Yuchen Qiao

# Part I: Data wrangling

## (1)Data Download and pre-processing

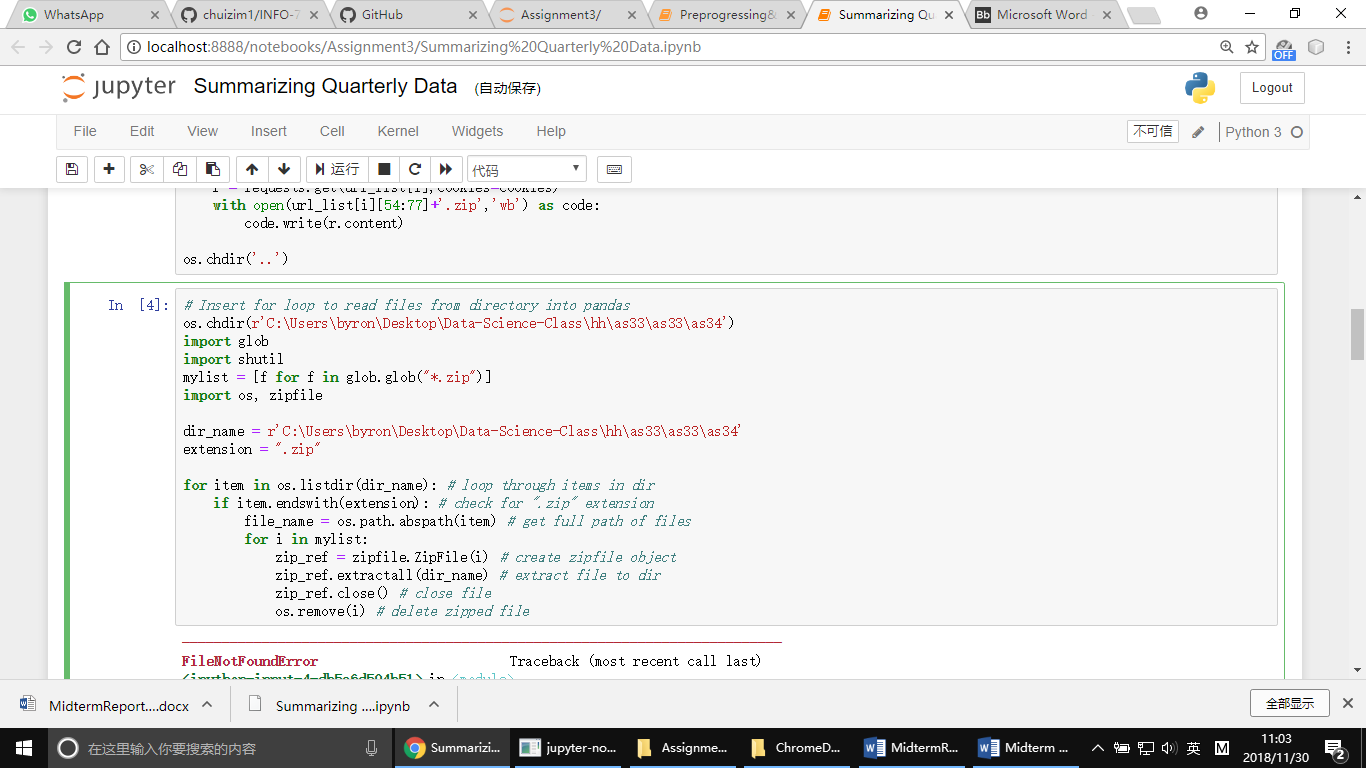
### 1. Login into website using cookies and downloading data.

We used a combination of BeautifulSoup, requests and cookies to access the Freddie Mac website and to download data. Before downloading the data, we created a new directory called ‘as34’ in the current working directory, where all the downloaded files were saved.



### 2.Insert for loop to read files from directory into pandas

After downloading the files, we used the glob library to loop through the folder and used the zipfile library to unzip all the zipped files in the folder, and used the os.remove() to delete the files from which text files had already been extracted. The code section below shows how the files were extracted.



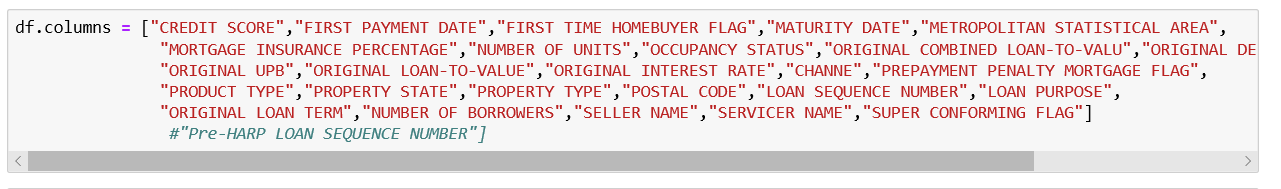
### Merging the files

We merged the files using the code block below to create one file



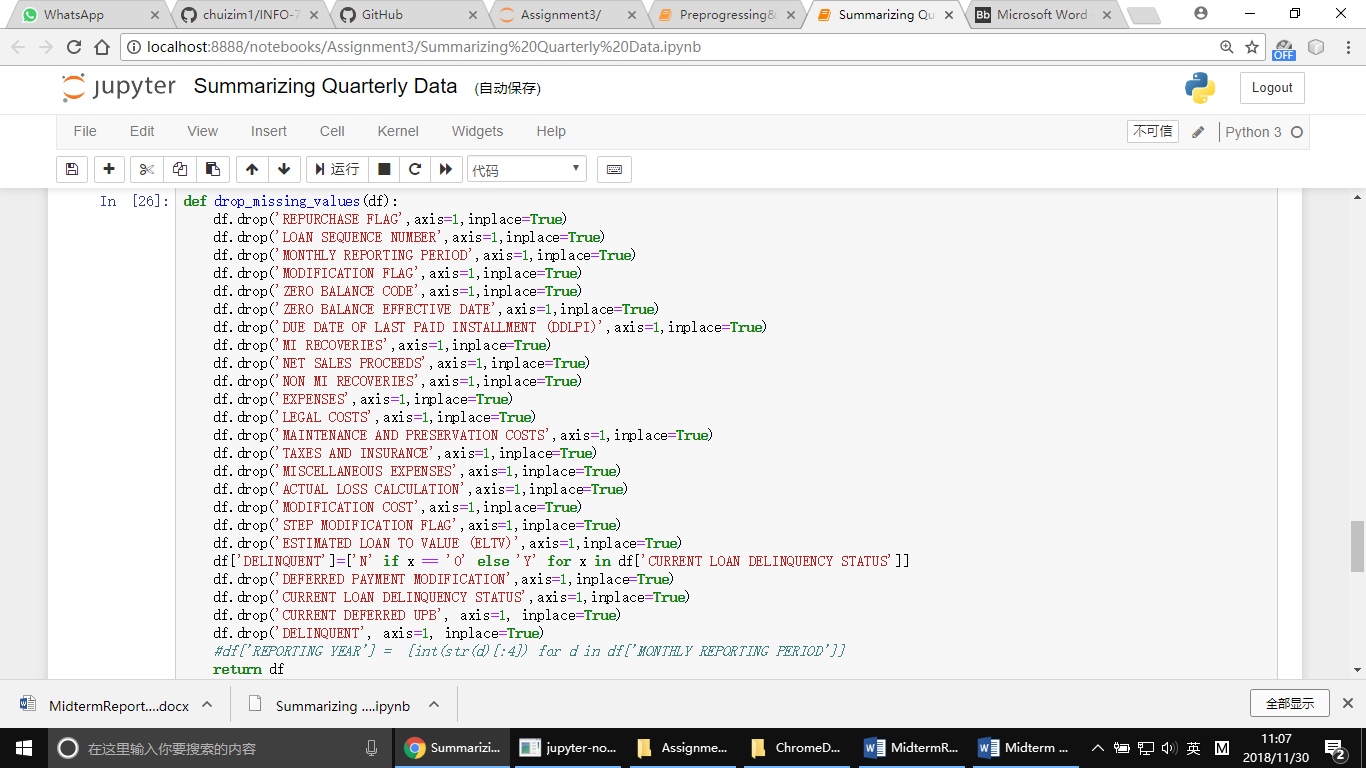
After extracting and combining the files into a single file, we use the function below to preprocess the data.

Adding column names to the data

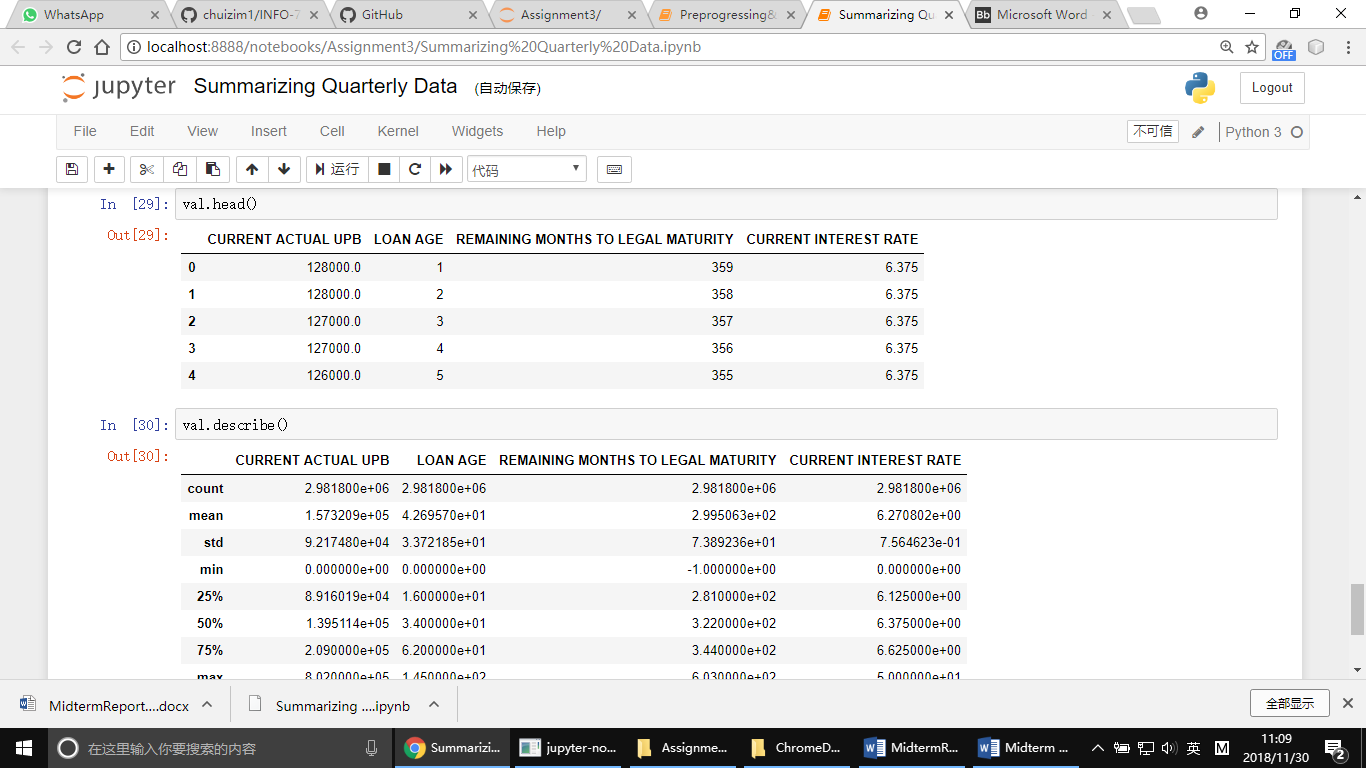


We used the function below to preprocess the data, delete columns that were not significant to determining the interest rate of a loan and extracts a data frame

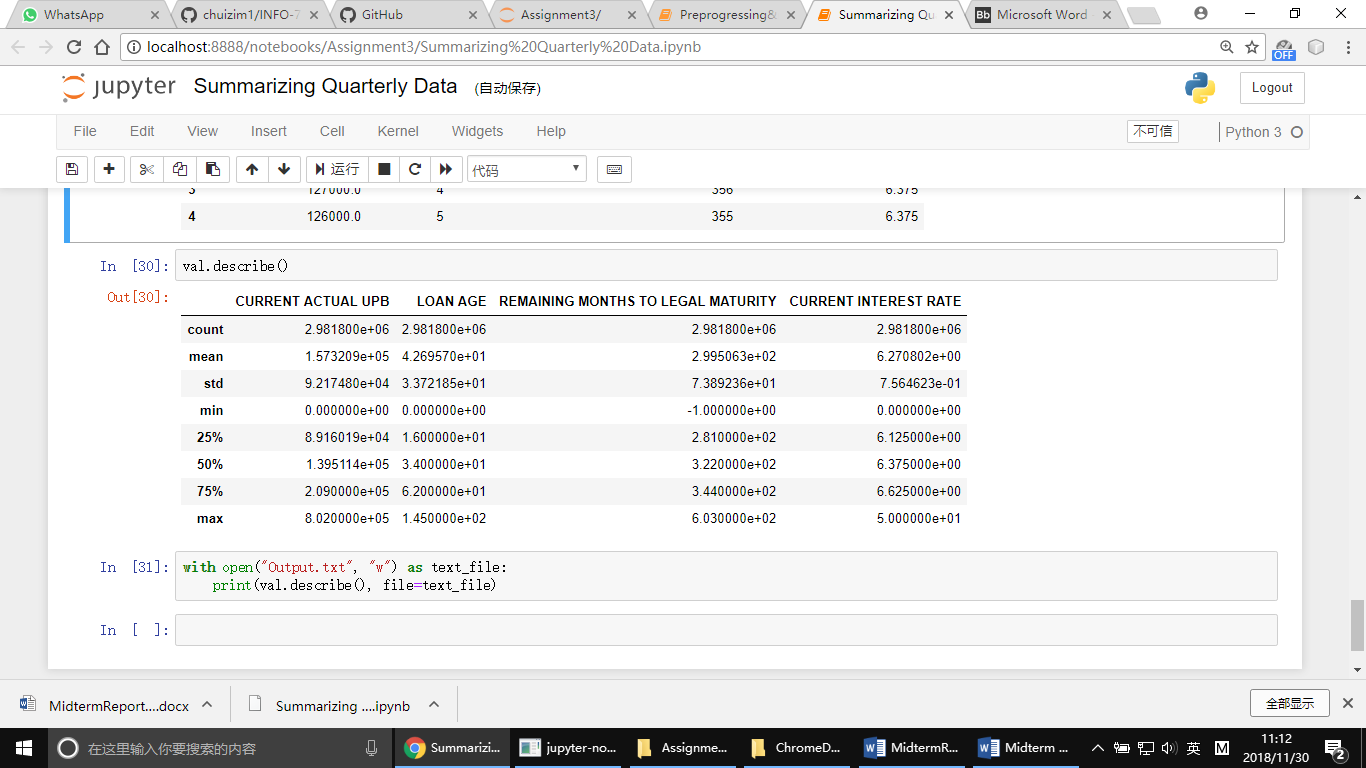
### Processing missing value



Data after cleaning:

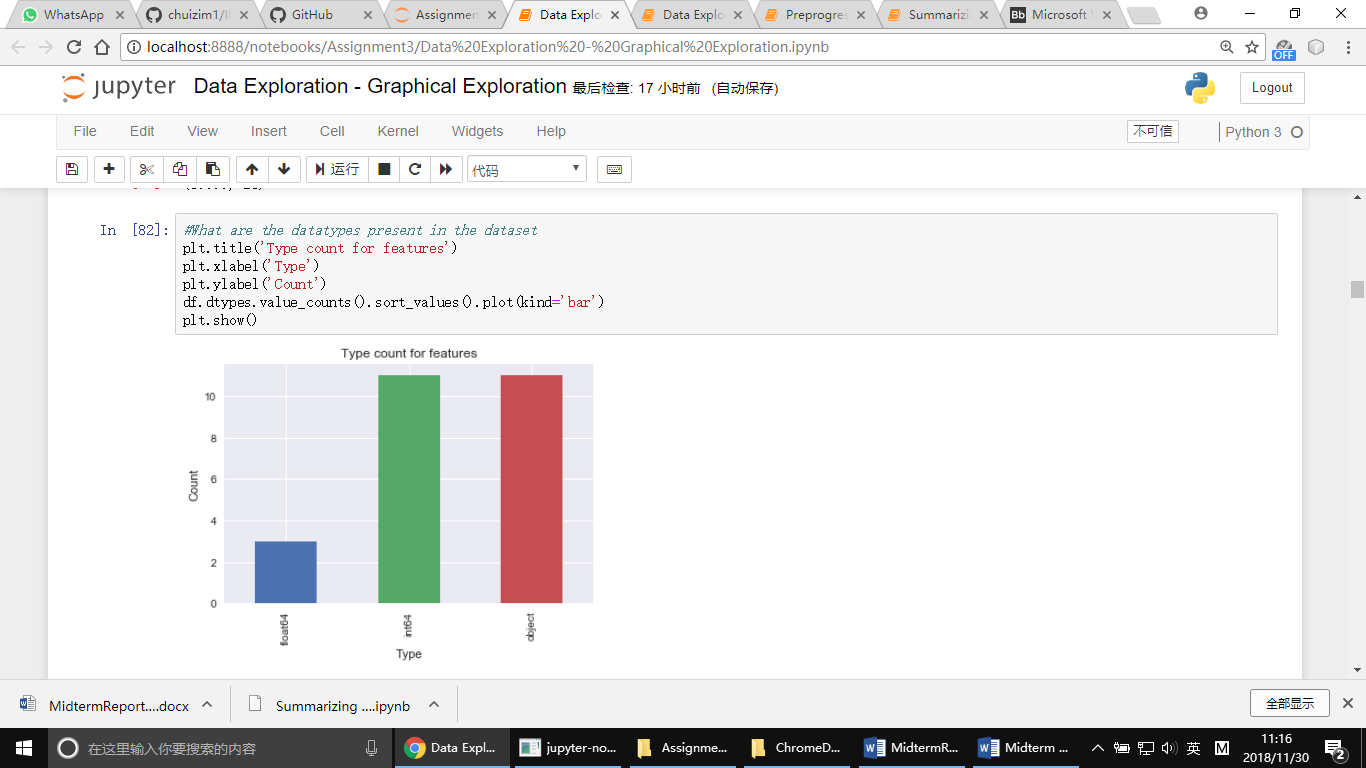


### 4.Generating Summaries. We choose the following summaries: CURRENT ACTUAL UPB, LOAN AGE, REMAINING MONTHS TO LEGAL MATURITY, CURRENT INTEREST RATE.

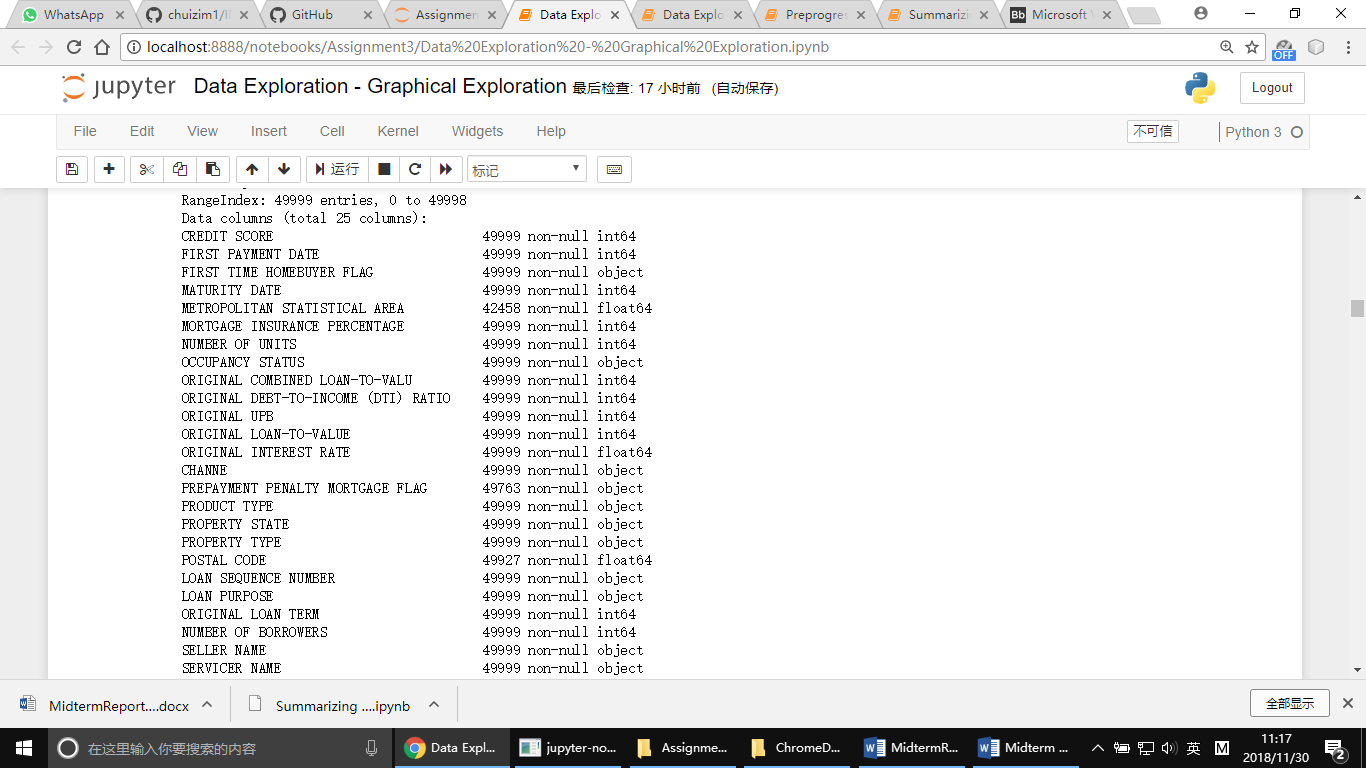


## Data Exploration: Sample Data

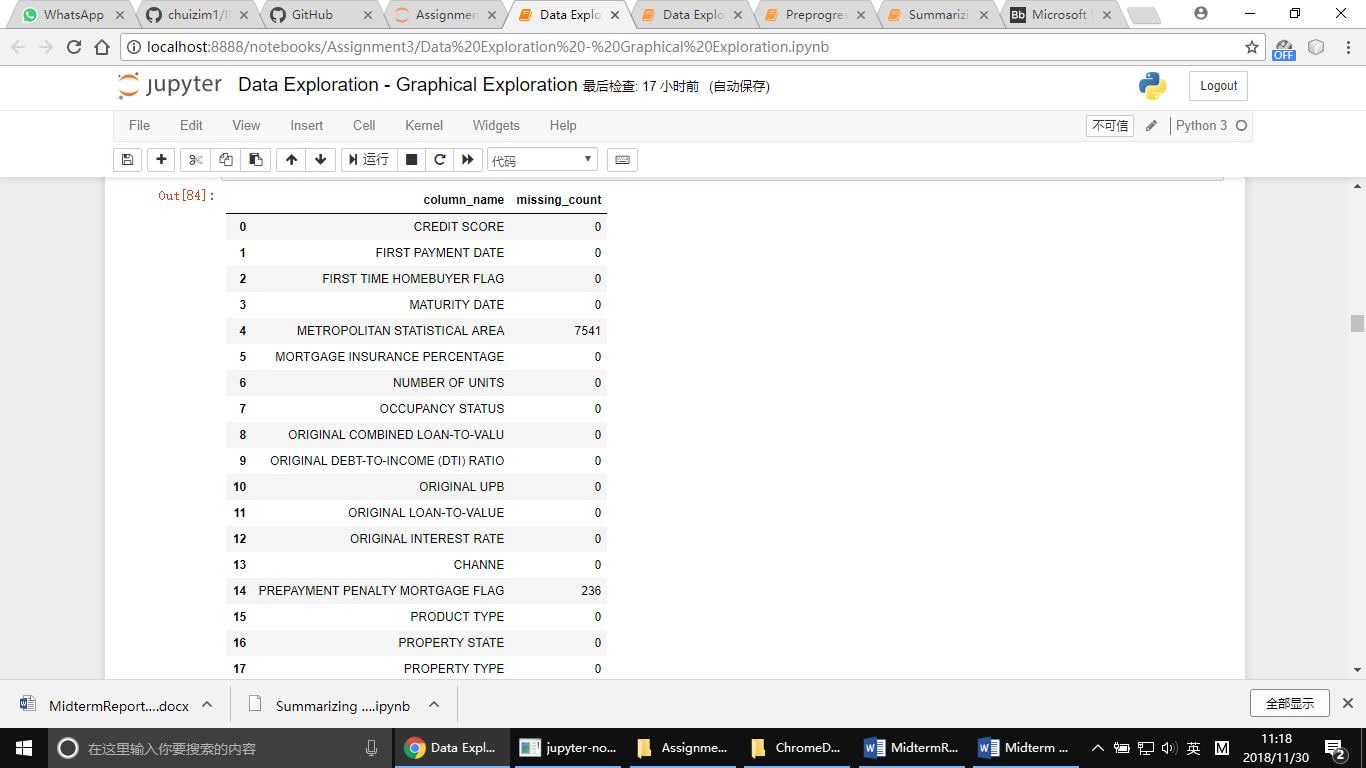
The bar graph below shows the counts of datatypes of the variables in the dataset. From the graph, we can infer that we have floats, integers and objects types in the dataset.



Most of the features in the dataset are either objects or integers with a few float values. The models we used can handle data type like float64 and int64 so the next step we need to do is to convert those object type into int or float.



Based on the dataset info (Structure of the dataset) above, most of the columns have their data, as such we won’t be dealing with a lot of missing values.

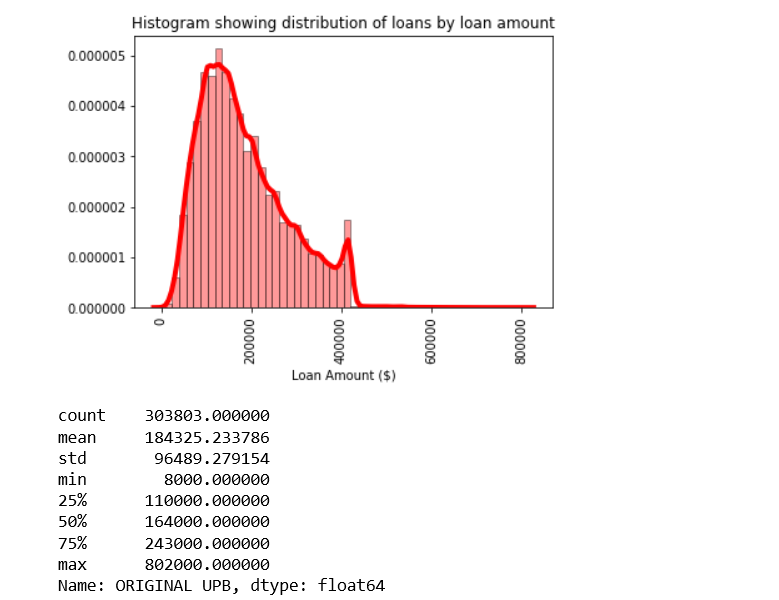


Only the METROPOLITAN STATISTICAL AREA, PREPAYMENT PENALTY MORTGAGE FLAG, POSTAL CODE are the only columns in the dataset that have missing values. We shall investigate and fill these columns. In POSTAL CODE and METROPOLITAN STATISTICAL AREA column, if we replace missing value with mean, we actually replace some places with center of the whole area. For column PREPAYMENT PENALTY MORTGAGE FLAG, it only have boolean values. So we decide to replace missing data in those columns with mode value.

The graph below shows that 20% of all loans in the sample data were issued in California, Florida and Texas.

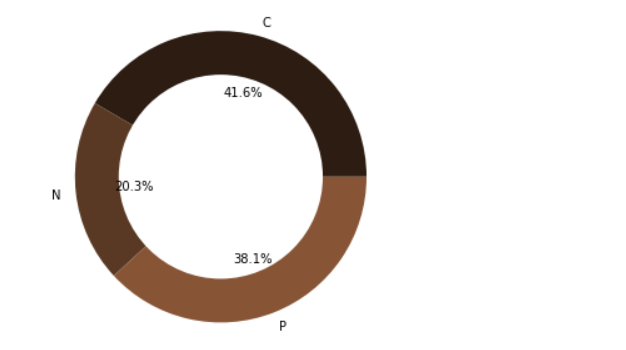
**ORIGINAL UPB**

Most of the loans are between 8k & about8k & about802k with the average loan amount equaling to $184k



**LOAN PURPOSE**

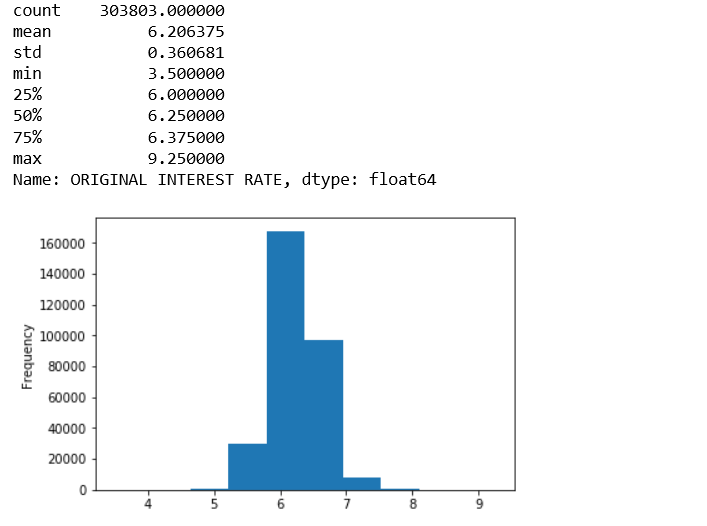
The graph below shows the distribution of loan purposes



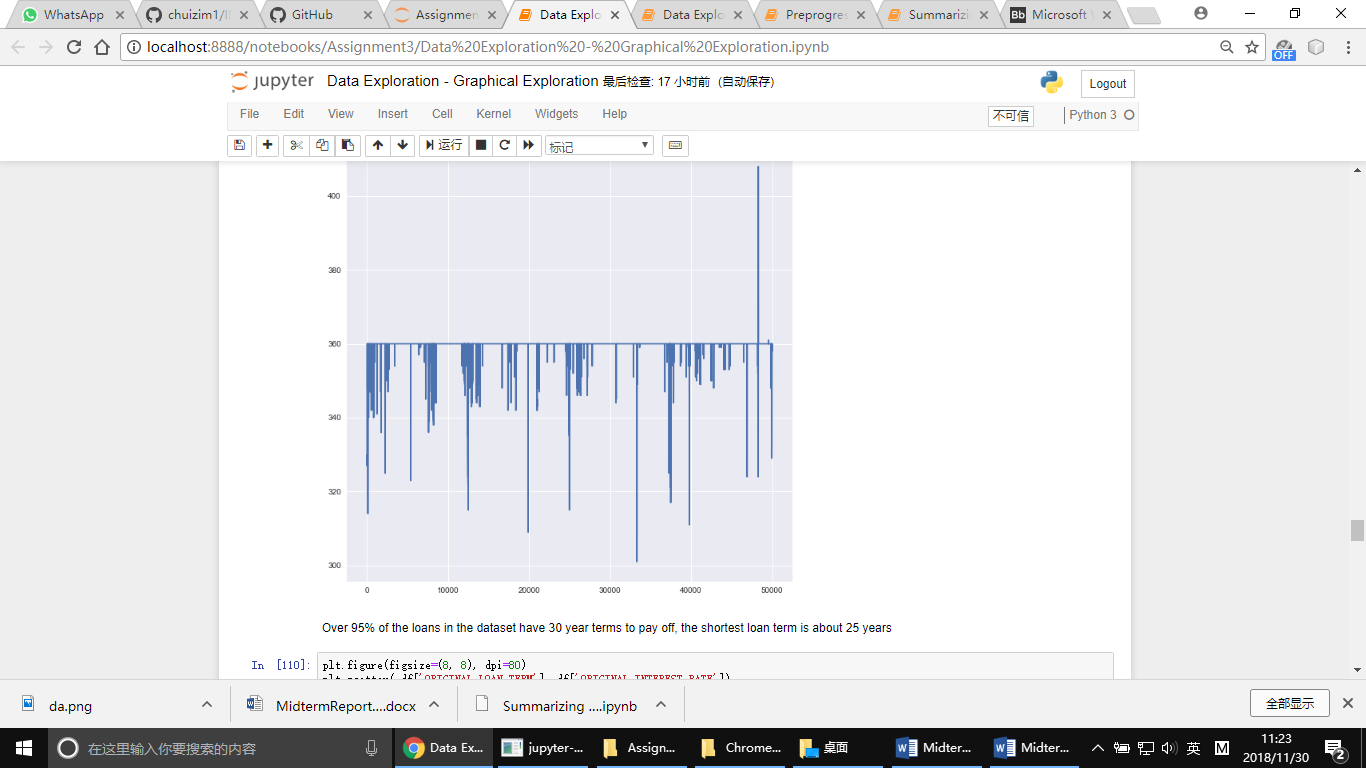
All the loans in the dataset were either for P = Purchase, C = Cash-out Refinance or N = No Cash-out Refinance

**ORIGINAL INTEREST RATE**

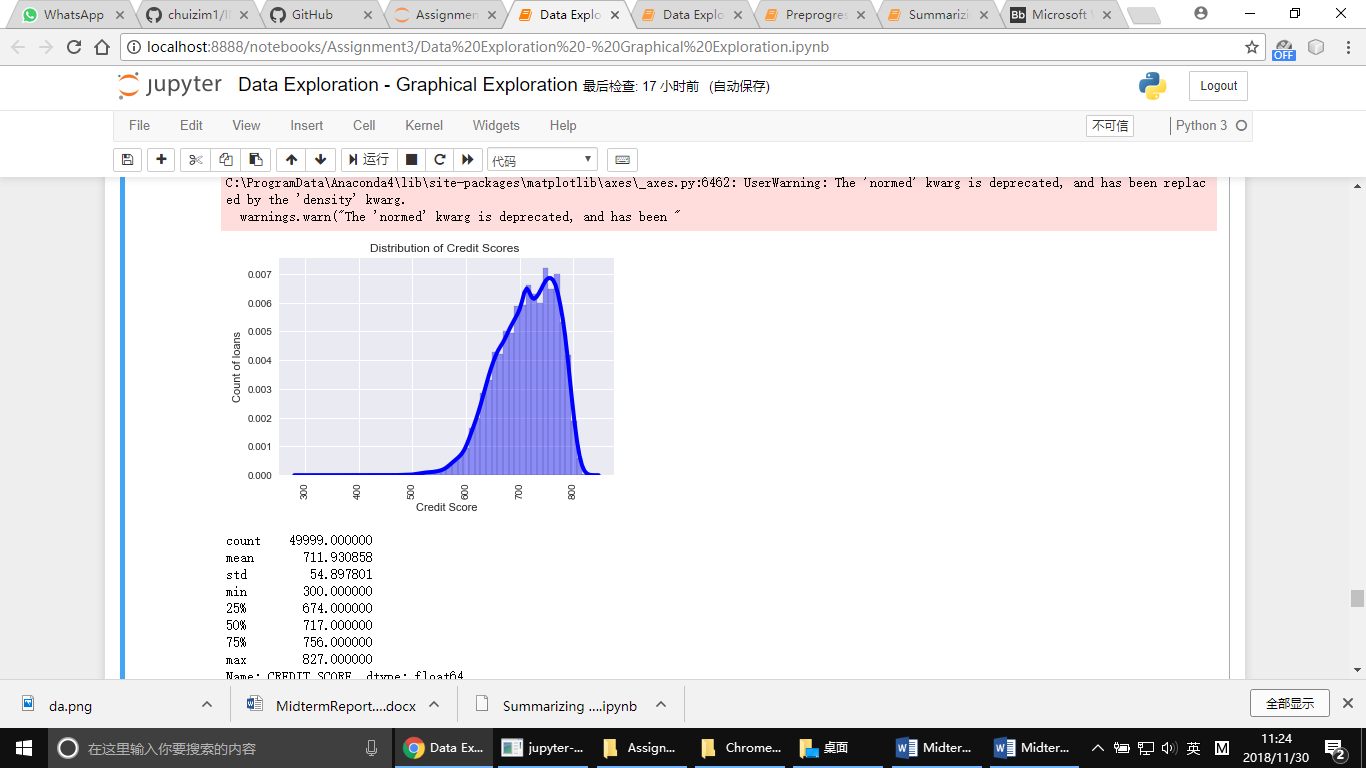
The interest rates are between 4.5% and 8%



The minimum interest rate was 5.125% while the max interest rate was 10.875%. The median interest rate was 8.18%.



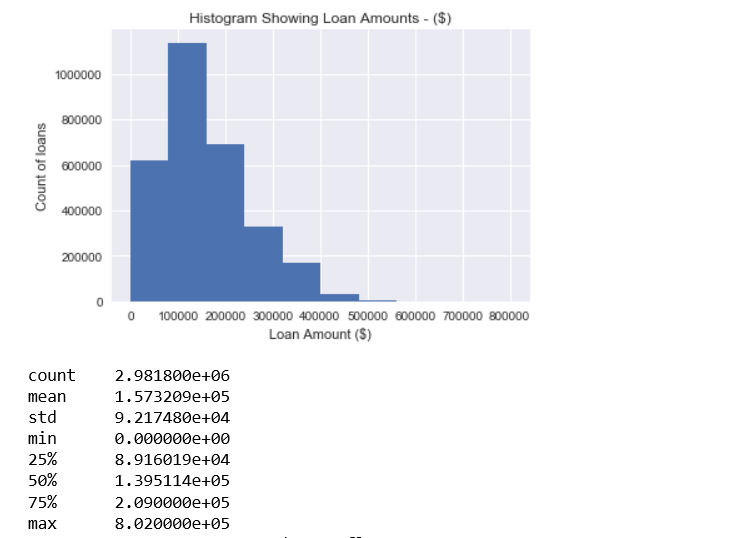
Over 95% of the loans in the dataset have 30-year terms to pay off, the shortest loan term is about 25 years.



People with a credit score of at least 500 had the highest chances of getting approved to buy homes.

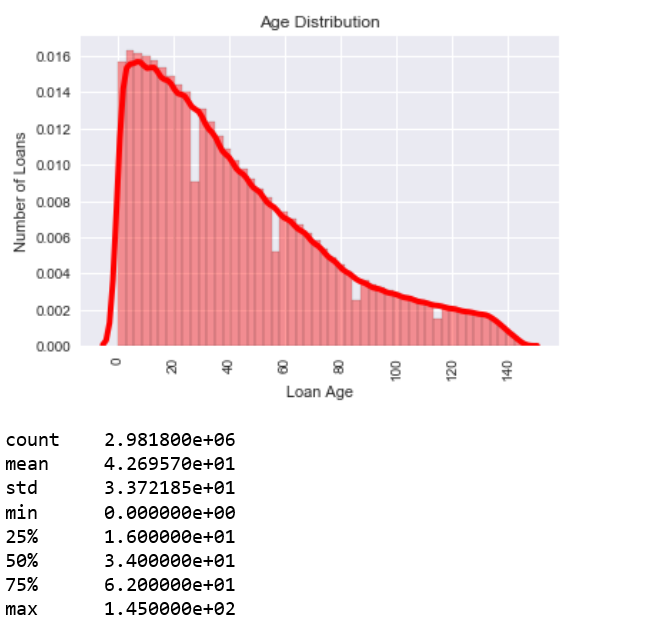
# Summarizing Quarterly Data

The maximum loan amount is about 800k while the average loan amount in the quarter was about 157k



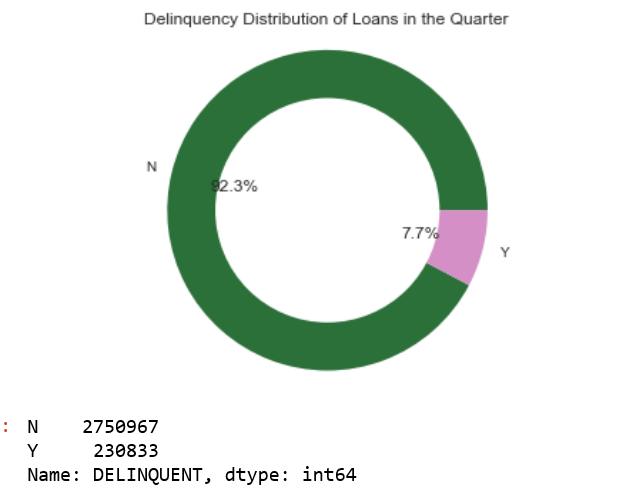
**Loan Age**

The oldest loan in the dataset is about 140 months old while over 60% of the loans were between 0 and 60 months old



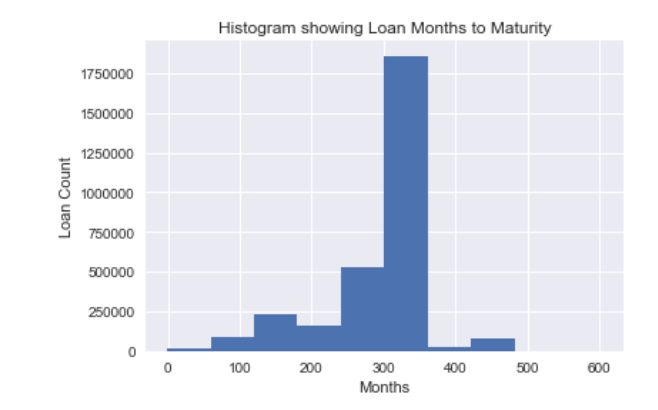
Over 80% of the loans are less than 5 years old

**Delinquency**



Only 7.7% of the loans in the quarter are non-performing (delinquent)

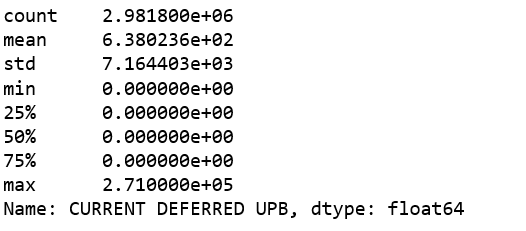
**Remaining Months to Legal Maturity**



One loan had been completely paid off before time, while the average months to legal maturity was about 300 months

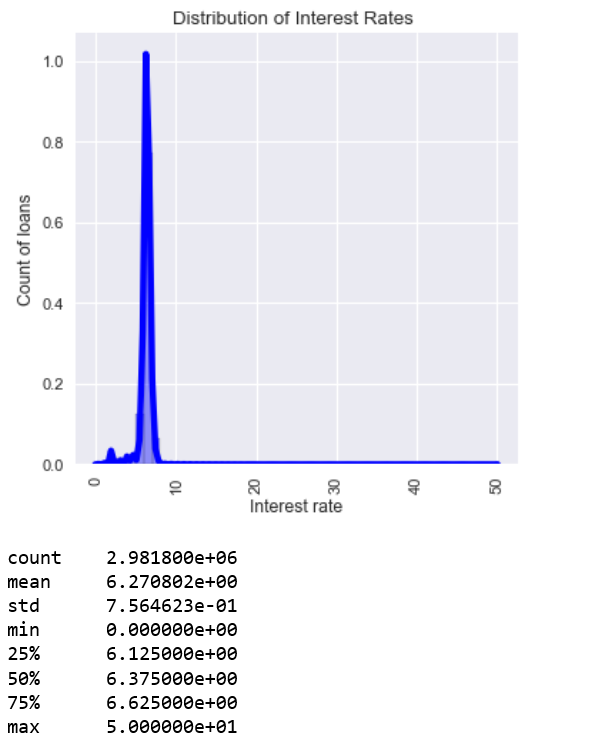
**CURRENT DEFERRED UPB**

The mean non-interest bearing loan amount during the quarter was 0, this means that almost all the mortgage loans in the dataset had no non-interest bearing percentages

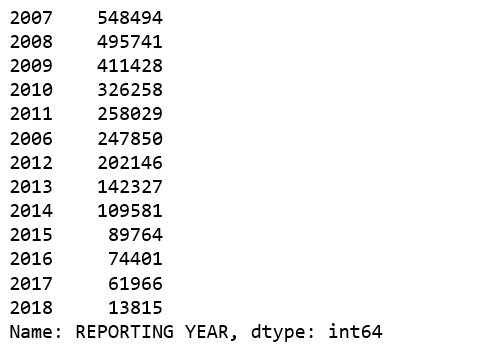


**Interest Rates**

The highest interest rate was about 50%. Overall the interest rate for the loans in the dataset was between 0 and 10%



**MONTHLY REPORTING PERIOD**



# Part 2 Building and evaluating models

## Prediction

* Programmatically downloads Q12005 and Q22005 origination data and pre-processes it.

import requests

import urllib

import os

from bs4 import BeautifulSoup

import http.cookiejar

import shutil

import pandas as pd

url = 'https://freddiemac.embs.com/FLoan/Data/download.php'

cookie={'PHPSESSID':'11vr6kgl2a9lt7b8if0mv08vq0'}

r = requests.post(url,cookies=cookie)

content=r.content

soup = BeautifulSoup(content,'lxml')

all\_href = soup.find\_all('a')

i=0;

dww=['Q12005','Q22005']

lod=len(dww)

url\_list=['https://freddiemac.embs.com/FLoan/Data/']\*lod

for href in all\_href:

for s in dww:

if s in href['href']:

url\_list[i]=url\_list[i]+href['href']

i+=1

if (not os.path.exists('datapart2')):

os.mkdir('datapart2')

os.chdir('datapart2')

for i in range(len(url\_list)):

r = requests.get(url\_list[i],cookies=cookie)

with open(url\_list[i][71:77]+'.zip','wb') as code:

code.write(r.content)

files= os.listdir()

for file in files:

for docu in dww:

if ('.zip' in file) and (docu in file):

shutil.unpack\_archive(file)

os.chdir('..')

Use cookies to get access to the download page, use BeautifulSoup to get content of the webpage, find the link we need, download the zip file, then unzip it by shutil package.

def preprocessing(data):

data['POSTAL CODE'].fillna(85200.0,inplace=True)

data['PREPAYMENT PENALTY MORTGAGE FLAG'].fillna('N',inplace=True)

data['METROPOLITAN STATISTICAL AREA'].fillna(16974.0,inplace=True)

data.drop(['SUPER CONFORMING FLAG'],axis=1,inplace=True)

data['CREDIT SCORE'].replace(to\_replace=9999, value=np.nan, inplace=True)

data['CREDIT SCORE'].fillna((data['CREDIT SCORE'].mean()), inplace=True)

data['FIRST TIME HOMEBUYER FLAG'].replace(to\_replace='9', value=np.nan, inplace=True)

data['FIRST TIME HOMEBUYER FLAG'].fillna('N', inplace=True)

data['Maturity\_year'] = [int(str(d)[:4]) for d in data['MATURITY DATE']]

del data['MATURITY DATE']

data['MORTGAGE INSURANCE PERCENTAGE'].replace(to\_replace=999,value=0,inplace=True)

data['NUMBER OF UNITS'].replace(to\_replace=99,value=1,inplace=True)

data['ORIGINAL COMBINED LOAN-TO-VALU'].replace(to\_replace=999,value=80,inplace=True)

data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].replace(to\_replace=999,value=float('nan'),inplace=True)

data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].fillna(data['ORIGINAL DEBT-TO-INCOME (DTI) RATIO'].mean(),inplace=True)

data['ORIGINAL LOAN-TO-VALUE'].replace(to\_replace=999,value=float('nan'),inplace=True)

data['ORIGINAL LOAN-TO-VALUE'].fillna(data['ORIGINAL LOAN-TO-VALUE'].mean(),inplace=True)

data.drop('PRODUCT TYPE',axis=1,inplace=True)

data.drop('LOAN SEQUENCE NUMBER',axis=1,inplace=True)

data['PROPERTY TYPE'].replace(to\_replace=99,value='SF',inplace=True)

data['LOAN PURPOSE'].replace(to\_replace=9,value='C',inplace=True)

data['NUMBER OF BORROWERS'].replace(to\_replace=99,value=2,inplace=True)

del data['FIRST PAYMENT DATE']

data.drop('ORIGINAL LOAN-TO-VALUE',axis=1,inplace=True)

data.drop('Maturity\_year',axis=1,inplace=True)

#dictionary for each column

cleanup\_nums = {"FIRST TIME HOMEBUYER FLAG":{"Y": 1, "N": 0},

"OCCUPANCY STATUS": {"P": 1, "S": 2, "I": 3 },

"CHANNE":{"T": 1, "R":2,"C":3,"B":4},

"PREPAYMENT PENALTY MORTGAGE FLAG":{"Y":1,"N":0},

"PROPERTY TYPE":{"SF":1,"PU":2,"CO":3,"MH":4,"CP":5,"99":99},

"LOAN PURPOSE":{"C":1,"P":2,"N":3},

"SELLER NAME":{"Other sellers": 0, "COUNTRYWIDE": 1, "TAYLOR,BEAN&WHITAKER": 2, "PROVIDENTFUNDINGASSO": 3, "USBANKNA": 4, "FIFTHTHIRDBANK": 5, "ABNAMROMTGEGROUP,INC": 6, "CHASEHOMEFINANCELLC": 7, "NATLCITYMTGECO": 8, "WELLSFARGOBANK,NA": 9, "GMACMTGECORP": 10, "WASHINGTONMUTUALBANK": 11, "FLAGSTARBANK,FSB": 12, "BANKOFAMERICA,NA": 13},

"PREPAYMENT PENALTY MORTGAGE FLAG":{"Y": 1, "N": 0},

"PROPERTY STATE":{"RI": 0, "OK": 1, "NY": 2, "MO": 3, "MN": 4, "IL": 5, "KY": 6, "WA": 7, "TX": 8, "FL": 9, "CA": 10, "IN": 11, "NJ": 12, "ID": 13, "TN": 14, "KS": 15, "MI": 16, "IA": 17, "MT": 18, "GA": 19, "OH": 20, "OR": 21, "ME": 22, "CT": 23, "WV": 24, "NH": 25, "VA": 26, "NC": 27, "AZ": 28, "NE": 29, "MD": 30, "MA": 31, "UT": 32, "CO": 33, "ND": 34, "PA": 35, "SC": 36, "DE": 37, "SD": 38, "WI": 39, "AL": 40, "AK": 41, "VT": 42, "LA": 43, "AR": 44, "NM": 45, "HI": 46, "DC": 47, "MS": 48, "NV": 49, "GU": 50, "WY": 51, "PR": 52, "VI": 53},

"SERVICER NAME":{"USBANKNA": 0, "Other servicers": 1, "PNCMTGESERVICES,INC": 2, "WELLSFARGOBANK,NA": 3, "WASHINGTONMUTUALBANK": 4, "PNCBANK,NATL": 5, "NATLCITYMTGECO": 6, "JPMORGANCHASEBANK,NA": 7, "NATIONSTARMTGELLCDBA": 8, "COUNTRYWIDE": 9, "BANKOFAMERICA,NA": 10, "BACHOMELOANSERVICING": 11, "CITIMORTGAGE,INC": 12, "PROVIDENTFUNDINGASSO": 13, "ABNAMROMTGEGROUP,INC": 14, "FIFTHTHIRDBANK": 15, "GMACMORTGAGE,LLC": 16},

"OCCUPANCY STATUS":{"P": 1, "S": 2, "I": 3 },

"CHANNE":{"T": 1, "R":2,"C":3,"B":4}

}

data.replace(cleanup\_nums, inplace=True)

#d\_seller = {"Other sellers": 0, "COUNTRYWIDE": 1, "TAYLOR,BEAN&WHITAKER": 2, "PROVIDENTFUNDINGASSO": 3, "USBANKNA": 4, "FIFTHTHIRDBANK": 5, "ABNAMROMTGEGROUP,INC": 6, "CHASEHOMEFINANCELLC": 7, "NATLCITYMTGECO": 8, "WELLSFARGOBANK,NA": 9, "GMACMTGECORP": 10, "WASHINGTONMUTUALBANK": 11, "FLAGSTARBANK,FSB": 12, "BANKOFAMERICA,NA": 13}

#d\_service = {"USBANKNA": 0, "Other servicers": 1, "PNCMTGESERVICES,INC": 2, "WELLSFARGOBANK,NA": 3, "WASHINGTONMUTUALBANK": 4, "PNCBANK,NATL": 5, "NATLCITYMTGECO": 6, "JPMORGANCHASEBANK,NA": 7, "NATIONSTARMTGELLCDBA": 8, "COUNTRYWIDE": 9, "BANKOFAMERICA,NA": 10, "BACHOMELOANSERVICING": 11, "CITIMORTGAGE,INC": 12, "PROVIDENTFUNDINGASSO": 13, "ABNAMROMTGEGROUP,INC": 14, "FIFTHTHIRDBANK": 15, "GMACMORTGAGE,LLC": 16}

#d\_seller.setdefault(data["SELLER NAME"], default=0)

#d\_service.setdefault(data["SERVICER NAME"], default=0)

for i in range(len(data['SELLER NAME'])):

if (not(str(data['SELLER NAME'][i]).isdigit())):

data['SELLER NAME'][i]=0

for i in range(len(data['SERVICER NAME'])):

if (not(str(data['SERVICER NAME'][i]).isdigit())):

data['SERVICER NAME'][i]=1

return data

Preprocessing the data.

* Builds a Regression model for the interest rate using Q12005 data as training data (col 13) Does variable selection to choose the best Regression model using Forward, Backward, Stepwise and Exhaustive search methods. Validates against the Q22005 datasets. Computes MAE, RMS, MAPE for training and testing datasets

def mean\_absolute\_percentage\_error(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

#Linear Regression

lin\_reg = LinearRegression()

lin\_reg.fit(train\_x,train\_y)

pred\_y = lin\_reg.predict(test\_x)

print("Score for train: "+str(lin\_reg.score(train\_x,train\_y)))

print("Score: "+str(lin\_reg.score(test\_x,test\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

Above is an example of linear regression model. We tried several different models include Linear Regression, Lasso, Decision Tree Regression, Ridge Regression, Bayesion Ridge.

* Repeat this using Random Forest & Neural Network algorithms.

Random Forest

rf\_reg = RandomForestRegressor()

rf\_reg.fit(train\_x,train\_y)

pred\_y = rf\_reg.predict(test\_x)

predvali\_y = rf\_reg.predict(vali\_x)

print("Score for train: "+str(rf\_reg.score(train\_x,train\_y)))

print("Score: "+str(rf\_reg.score(test\_x,test\_y)))

print("Score for Validates: "+str(rf\_reg.score(vali\_x,vali\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

Neural Network

transformer = Normalizer().fit(train\_x)

mlp\_train\_x = pd.DataFrame(transformer.transform(train\_x))

mlp\_train\_x.columns = train\_x.columns

mlp\_test\_x = Normalizer().fit(test\_x)

mlp\_test\_x = pd.DataFrame(transformer.transform(test\_x))

mlp\_test\_x.columns = test\_x.columns

mlp\_vali\_x = Normalizer().fit(vali\_x)

mlp\_vali\_x = pd.DataFrame(transformer.transform(vali\_x))

mlp\_vali\_x.columns = vali\_x.columns

#mlp

mlp\_reg2 = MLPRegressor(hidden\_layer\_sizes=(300,20),learning\_rate='adaptive',solver='adam',random\_state=3,learning\_rate\_init=0.001,max\_iter=400)

mlp\_reg.fit(mlp\_train\_x,train\_y)

pred\_y = mlp\_reg.predict(mlp\_test\_x)

predvali\_y = mlp\_reg.predict(mlp\_vali\_x)

print("Score for train: "+str(mlp\_reg.score(mlp\_train\_x,train\_y)))

print("Score: "+str(mlp\_reg.score(mlp\_test\_x,test\_y)))

print("Score for Validates: "+str(mlp\_reg.score(mlp\_vali\_x,vali\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

* Try TPOT, H20.Ai and AutoSKLearn Automl algorithms

tpot = TPOTRegressor(generations=3,scoring='r2',population\_size=10)

tpot.fit(train\_x,train\_y)

tpot.score(test\_x,test\_y)

tpot.export('tpot\_assignment3.py')

h2o.init()

train = train\_x.copy()

train['ORIGINAL INTEREST RATE']=train\_y

train = h2o.H2OFrame(train)

test = test\_x.copy()

test['ORIGINAL INTEREST RATE']=test\_y

test = h2o.H2OFrame(test)

vali = vali\_x.copy()

vali['ORIGINAL INTEREST RATE']=vali\_y

vali = h2o.H2OFrame(vali)

aml = H2OAutoML(max\_runtime\_secs = 600, seed = 1, project\_name = "lending\_club")

aml.train(y = 'ORIGINAL INTEREST RATE', training\_frame = train, leaderboard\_frame = test)

aml.leaderboard.head()

* Choose the best model amongst the different types of algorithms.

The best model we get is Gradient Boosting Regression.

gbm\_reg = GradientBoostingRegressor(alpha=0.85, learning\_rate=0.1, loss="huber", max\_depth=10, max\_features=0.7, min\_samples\_leaf=18, min\_samples\_split=19, n\_estimators=100, subsample=0.4,verbose=2)

gbm\_reg.fit(train\_x,train\_y)

pred\_y = gbm\_reg.predict(test\_x)

print("Score for train: "+str(gbm\_reg.score(train\_x,train\_y)))

print("Score: "+str(gbm\_reg.score(test\_x,test\_y)))

print("RMS: "+str(sqrt(metrics.mean\_squared\_error(test\_y,pred\_y))))

print("MAPE: "+str(mean\_absolute\_percentage\_error(test\_y,pred\_y)))

print("R2: "+str(metrics.r2\_score(test\_y,pred\_y)))

print("MAE: "+str(metrics.mean\_absolute\_error(test\_y,pred\_y)))

* You are asked to do what-if analysis had your algorithm used in various scenarios:
* Financial crisis
* Run your algorithm for 4 rolling quarters and report your findings and discuss it in your report. (i.e Use Q12007, Q22007, Q32007, Q42007 for training and predict for Q22007, Q32007, Q42007, Q12008)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12007 | Q22007 | 0.3352401243090923 | 3.916927487675391 | 0.2684 | 0.2501980157915151 |
| Q22007 | Q32007 | 0.4791132294150997 | 5.923953649333251 | -0.5436 | 0.40251641096175517 |
| Q32007 | Q42007 | 0.46435446061447644 | 6.178674941610631 | -0.1010 | 0.38000884041383104 |
| Q42007 | Q12008 | 0.554393207159252 | 8.351641591136772 | -0.2060 | 0.47045738324609065 |

We found that the accuracy of the model drops rapidly from 2ed quarter, because of the influence of financial crisis

* Run your algorithm 2 years later (i.e, 2009 for all 4 quarters)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12009 | Q22009 | 0.2878 | 4.5366 | 0.3000 | 0.2194 |
| Q22009 | Q32009 | 0.4095 | 6.2581 | 0.0150 | 0.3313 |
| Q32009 | Q42009 | 0.3088 | 5.2760 | 0.2751 | 0.2540 |
| Q42009 | Q12010 | 0.2044 | 3.1715 | 0.6824 | 0.1572 |

Compare to above, the model performs pretty good if there isn’t a financial crisis.

* Economic boom (1999, 2013)
* Discuss your design and results in a report. Would you recommend using this model for the next quarter? Justify

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q11999 | Q21999 | 0.4169 | 4.2585 | -0.2088 | 0.3132 |
| Q21999 | Q31999 | 0.7315 | 8.0781 | -1.7689 | 0.6418 |

The model performs bad during a economic boom. I don’t recommend using this model for the next quarter, because the interest rate changes with the economic conditions. We need to consider the economic conditions when built our model. Only predicting interest rates in the next quarter based on information from the origination data from the prior quarter won’t get a good result in many cases.

* Regime change (2016) from election

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train data | Validation data | RMS | MAPE | R2 | MAE |
| Q12016 | Q22016 | 0.2688 | 5.9976 | 0.6034 | 0.2205 |
| Q22016 | Q32016 | 0.2628 | 5.9924 | 0.6442 | 0.2123 |
| Q32016 | Q42016 | 0.2765 | 5.3891 | 0.6754 | 0.2026 |
| Q42016 | Q12017 | 0.5821 | 12.3629 | -0.5579 | 0.5300 |

The model works well in the first three quarters, but when election happened in 4th quarter, the accuracy drop rapidly.

## Classification

* Programmatically downloads Q12005 and Q22005 origination data and pre-processes it.

import requests

import urllib

import os

from bs4 import BeautifulSoup

import http.cookiejar

import shutil

import pandas as pd

url = 'https://freddiemac.embs.com/FLoan/Data/download.php'

cookie={'PHPSESSID':'11vr6kgl2a9lt7b8if0mv08vq0'}

r = requests.post(url,cookies=cookie)

content=r.content

soup = BeautifulSoup(content,'lxml')

all\_href = soup.find\_all('a')

i=0;

dww=['Q12005','Q22005']

lod=len(dww)

url\_list=['https://freddiemac.embs.com/FLoan/Data/']\*lod

for href in all\_href:

for s in dww:

if s in href['href']:

url\_list[i]=url\_list[i]+href['href']

i+=1

if (not os.path.exists('datapart2')):

os.mkdir('datapart2')

os.chdir('datapart2')

for i in range(len(url\_list)):

r = requests.get(url\_list[i],cookies=cookie)

with open(url\_list[i][71:77]+'.zip','wb') as code:

code.write(r.content)

files= os.listdir()

for file in files:

for docu in dww:

if ('.zip' in file) and (docu in file):

shutil.unpack\_archive(file)

os.chdir('..')

Use cookies to get access to the download page, use BeautifulSoup to get content of the webpage, find the link we need, download the zip file, then unzip it by shutil package.

def preprocessing(data):

data['Delinquent']=[0 if x == '0' else 1 for x in data['CURRENT LOAN DELINQUENCY STATUS']]

data.drop(['LOAN SEQUENCE NUMBER','REPURCHASE FLAG','CURRENT LOAN DELINQUENCY STATUS','MODIFICATION FLAG','ZERO BALANCE CODE'

,'ZERO BALANCE EFFECTIVE DATE','DUE DATE OF LAST PAID INSTALLMENT (DDLPI)','MI RECOVERIES'

,'NET SALES PROCEEDS','NON MI RECOVERIES','EXPENSES','LEGAL COSTS','MAINTENANCE AND PRESERVATION COSTS'

,'TAXES AND INSURANCE','MISCELLANEOUS EXPENSES','ACTUAL LOSS CALCULATION','MODIFICATION COST'

,'STEP MODIFICATION FLAG','ESTIMATED LOAN TO VALUE (ELTV)'],axis=1,inplace=True)

dictionary\_25 = {'Y':1,'N':0,' ':-1}

data['DEFERRED PAYMENT MODIFICATION'] = [dictionary\_25[x] for x in data['DEFERRED PAYMENT MODIFICATION']]

return data

Do necessary preprocessing steps.

* Builds a Logistic regression model for the CURRENT LOAN DELINQUENCY STATUS, using Q12005 data as training data (col 4). Note anytime col 4 is > 0, add a new variable as Delinquent.

data['Delinquent']=[0 if x == '0' else 1 for x in data['CURRENT LOAN DELINQUENCY STATUS']]

Generate ‘Delinquent’ column by the CURRENT LOAN DELINQUENCY STATUS, ‘0’ present not delinquent, ‘1’ present delinquent.

logi = LogisticRegression()

logi.fit(train\_x,train\_y)

pred\_y = logi.predict(test\_x)

cm=confusion\_matrix(test\_y,pred\_y)

Build logistic model and train it with Q12005 data. Validate against Q22005, use the confusion matrix to show the performance.

* Validates against Q22005 data and selects the best Classification model. Computes ROC curve and Confusion matrices for training and testing datasets. Repeat this using Random Forest & Neural Network algorithms.Try TPOT, H20.Ai and AutoSKLearn Automl algorithms. Choose the best model amongst the different types of algorithms.

We tried Logistic Regression, SVM, Decision Tree, Naïve Bayes, KNN, Random Forest, Extra Tree, MLP, H2o.ai, TPOT. The best model we got is Random Forest.

rf = RandomForestClassifier(max\_depth=7,n\_estimators=100,verbose=2)

rf.fit(train\_x,train\_y)

pred\_y = rf.predict(test\_x)

cm=confusion\_matrix(test\_y,pred\_y)

tn, fp, fn, tp = cm.ravel()

print(cm)

print(tn, fp, fn, tp)

[[285663 463]

[ 13326 548]]

285663 463 13326 548

prob\_y=rf.predict\_proba(test\_x)

fpr, tpr, thresholds = metrics.roc\_curve(test\_y, prob\_y[:,1],pos\_label=1)

plt.plot(fpr, tpr, 'b')A screenshot of a cell phone

Description generated with very high confidence

* Parameterize the input (example it should take Q12005) and modify the code so that it outputs the 5 parameters listed in the matrix below.

from configparser import ConfigParser

cfg = ConfigParser()

cfg.read('classification.config')

value = cfg.get('cookie','value')

docu = cfg.get('file','name')

Use configparser to parameterize the input.

print('Number of Actual Delinquents:', tp+fn)

print('Number of Predicted Delinquents:', tp+fp)

print('Number of records in the dataset:', tn+tp+fn+fp)

print('Number of Delinquents properly classified:', tp)

print('Number of nondelinquents improperly classified as delinquents:', fp)

Compute the five columns by confusion matrix.

* Write another script that calls the above classification script from Q11999-Q42016 and computes the following matrix.

# -\*- coding: utf-8 -\*-

"""

Created on Thu Nov 29 01:26:05 2018

@author: wenqi

"""

import pandas as pd

import gc

from cla\_main import main\_function

filelist=[

'Q11999',

'Q21999',

'Q31999',

'Q41999',

'Q12000',

'Q22000',

'Q32000',

'Q42000',

]

value='11vr6kgl2a9lt7b8if0mv08vq0'

df = pd.DataFrame(columns = ["Actual Delinquents", "Predicted Delinquents", "Records in the dataset"

, "Delinquents properly classified","nondelinquents improperly classified as delinquents"])

for file in filelist:

tn, fp, fn, tp=main\_function(file,value)

df.loc[df.shape[0]+1]=[tp+fn,tp+fp,tn+tp+fn+fp,tp,fp]

gc.collect()

filelist2=[

'Q21999',

'Q31999',

'Q41999',

'Q12000',

'Q22000',

'Q32000',

'Q42000',

'Q12001',

]

df["Quarter"]=filelist2

print(df)

df.to\_csv('matrix.csv')

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Actual Delinquents | Predicted Delinquents | Records in the dataset | Delinquents properly classified | nondelinquents improperly classified as delinquents | Validation Quarter |
| 1 | 37190 | 178 | 100000 | 51 | 127 | Q21999 |
| 2 | 4062 | 4861 | 100000 | 122 | 4739 | Q31999 |
| 3 | 4927 | 141 | 100000 | 53 | 88 | Q41999 |
| 4 | 5450 | 55 | 100000 | 55 | 0 | Q12000 |
| 5 | 4655 | 125 | 100000 | 117 | 8 | Q22000 |
| 6 | 6405 | 394 | 100000 | 163 | 231 | Q32000 |
| 7 | 4593 | 106 | 100000 | 62 | 44 | Q42000 |
| 8 | 5469 | 151 | 100000 | 87 | 64 | Q12001 |