# CAPP 30254: Maching Learning for Public Policy

Assignment 1: Machine Learning Pipeline

**Yuliana Zamora**

Collaboration with Jean Salac Due: April 18, 2018

GitHub Page

## Ml Pipeline Description

Here I create a python script, pipe_tools.py, that allows us to build a simple, modular, extensible, machine learning pipeline then use this pipeline to predict who will experience financial distress in the next two years. I use scikit learns logistic regression to evaluate the data.

Below I will explain the current functions in the pipeline.

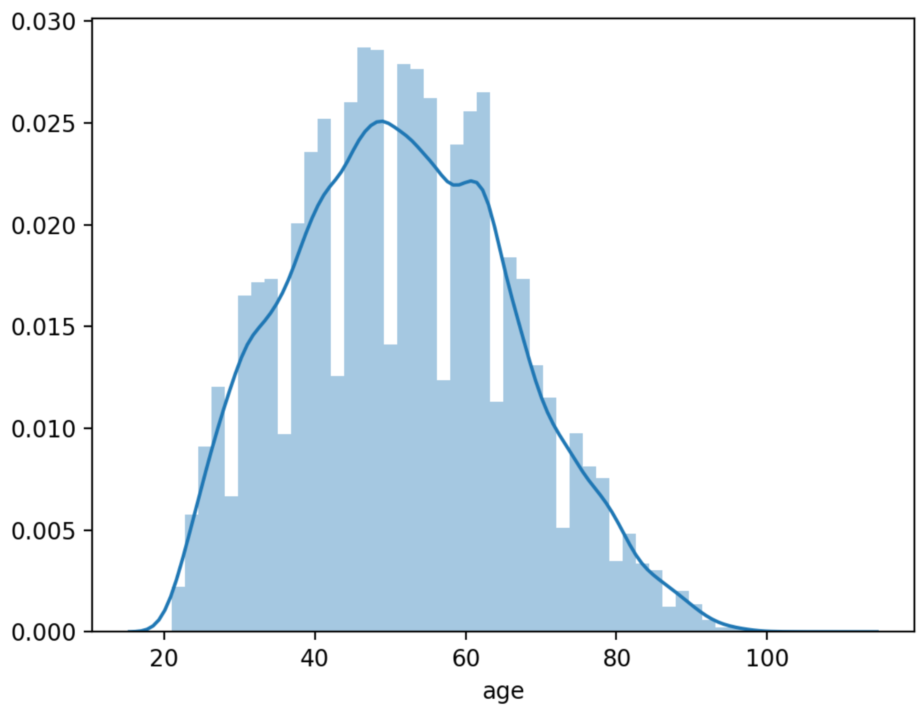
## 1 Components of pipeline

### 1.1 Read Data

**Read in Data** - *load_data(csv_file)* - takes in a csv file and converts to dataframe using pandas pd.read function

### 1.2 Explore Data

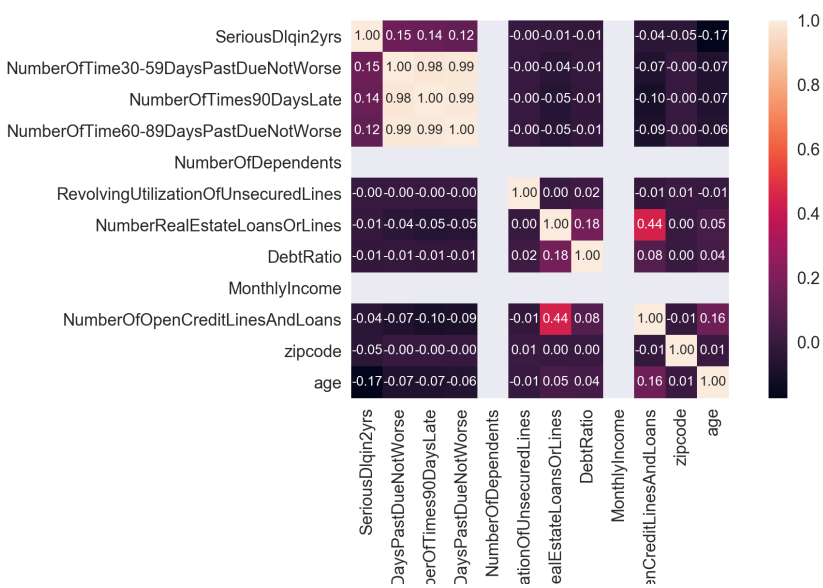
**auto histogram** - *histogram(data_frame)* - function combines the matplotlib hist function (with automatic calculation of a good default bin size). Below, you can find a simple output of this function using the age as the category to plot against. Below is the statistics of the debt ratio and a graph with a particular distribution.



**Summary of data** - *summary(data_frame)* - gives common statistical results of data, such as mean, median, etc, of data in reference to the debt ratio.

```
count     41016.000000
mean        331.458137
std        1296.109695
min           0.000000
25%           0.176375
50%           0.369736
75%           0.866471
max      106885.000000
Name: DebtRatio, dtype: float64
```

**Correlation Heat Map** - *cor_heat(data_frame,var_name* - creates a correlation heat map from the data set where var_name is the variable which has the most correlation. Here, SeriousDlqin2yrs, was chosen to see the correlation it has among all the other factors. There seems to be a lot more negative correlations than I was anticipating.




**Correlation Graphs** - Multiple graphs are created using pairplot function. The graphs below compare "SeriousDlqin2yrs","NumberOfOpenCreditLinesAndLoans","NumberRealEstateLoansOrLines",and "age".


**Missing data** - *miss_data(data_frame)* - Creates a table where the items with the most missing data is at the top. It allows to quickly see which items have the most missing data. From the table below, you can see that MonthlyIncome and NumberofDependents has the largest amount of missing data.

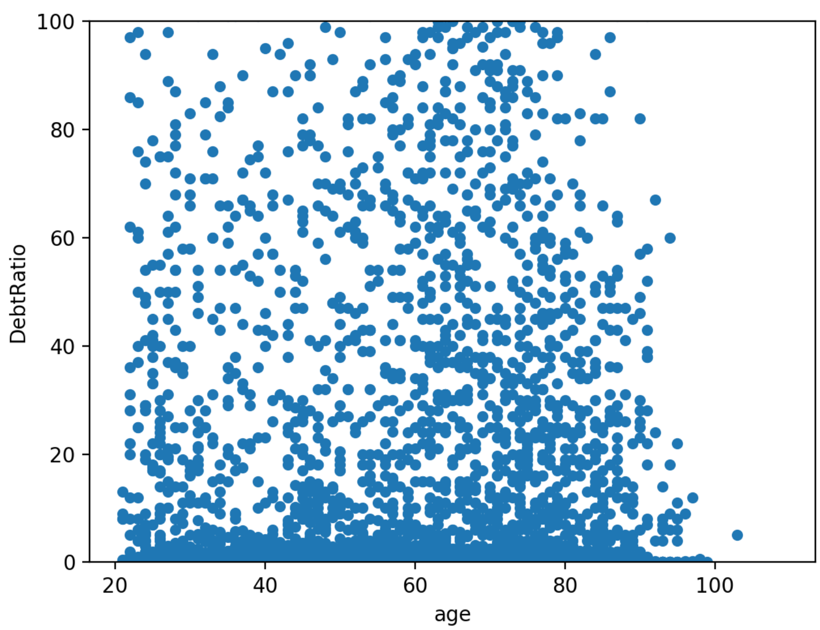
| | Total | Percent |
|---|---|---|
| MonthlyIncome | 7974 | 0.194412 |
| NumberOfDependents | 1037 | 0.025283 |
| NumberOfTime60-89DaysPastDueNotWorse | 0 | 0.000000 |
| NumberRealEstateLoansOrLines | 0 | 0.000000 |
| NumberOfTimes90DaysLate | 0 | 0.000000 |
| NumberOfOpenCreditLinesAndLoans | 0 | 0.000000 |
| DebtRatio | 0 | 0.000000 |
| NumberOfTime30-59DaysPastDueNotWorse | 0 | 0.000000 |
| zipcode | 0 | 0.000000 |
| age | 0 | 0.000000 |
| RevolvingUtilizationOfUnsecuredLines | 0 | 0.000000 |
| SeriousDlqin2yrs | 0 | 0.000000 |
| PersonID | 0 | 0.000000 |

**Scaling - univariate** - *scale(data_frame,var_scale)* - Creates a univariate analysis and scales the data in order to print out low range and high ranges. Below is the output of low and high distribution using data from seriousdllqin2yrs data.

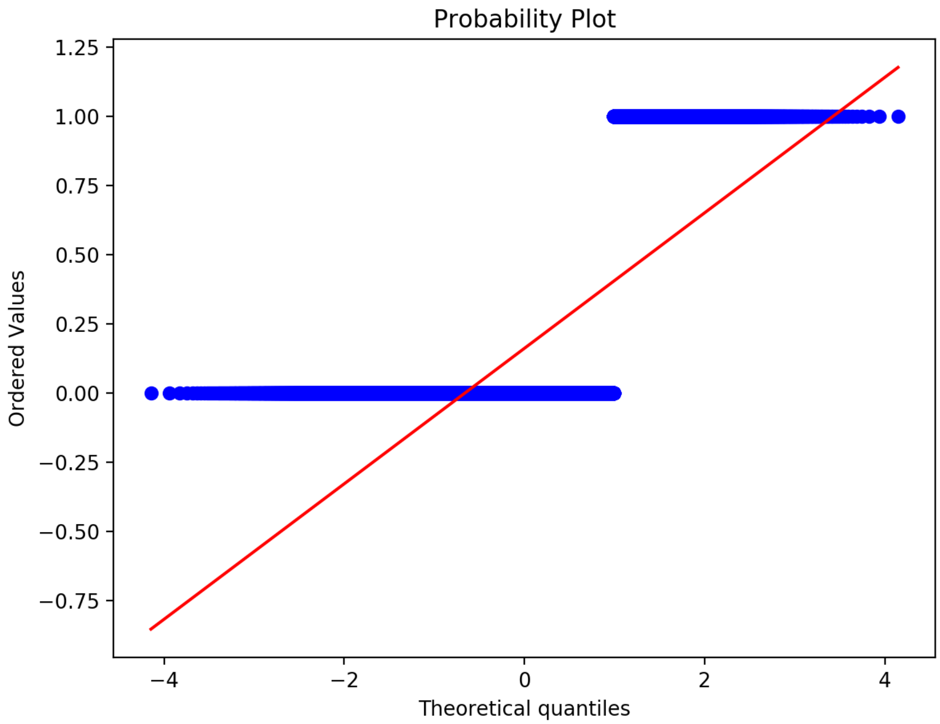
```
outer range (low) of the distribution:
[[-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]
 [-0.43870747]]

outer range (high) of the distribution:
[[2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]
 [2.27942326]]
```

**Bivariate Analysis** - *bivariate(data_frame,var_1,var_2)* - Creating a bivariate analysis to see the association and strength of the two attributes given in the function. Below, you can see the relationship between debt ratio and age.




**Normal Probability Plot** - *norm_plot(data_frame,var_name)* - function creates a plot with the normal probability and a histogram to help with visualization of the relationships




## 1.3 Pre-Process Data

**Pretty font** - *camel_case(column_name)* - changes formatting of words from camel case to snake case. In certain cases, it is easier to read snake case over camel case. It will be a handy option to have.
**Clean data** - *clean_miss(data_frame)* - function drops all values with corresponding nan values. For example, the MonthlyIncome category would be taken out as it has a high percentage of data missing, in addition to NumberofDependents even though only 2.5% is missing. Work needs to be done in order to get rid of data of with a certain threshold of data missing.
**Empty Data** - *fill_empty(data_frame,var,new_var)* - filling empty items of the specified item in the data frame with a specified number (new_var).

| | |
|---|---|
| 0 | 0.0 |
| 1 | 15666.0 |
| 2 | 4200.0 |
| 3 | 9052.0 |
| 4 | 10406.0 |
| 5 | 13500.0 |
| 6 | 3583.0 |
| 7 | 2700.0 |
| 8 | 3400.0 |
| 9 | 5050.0 |
| 10 | 2750.0 |
| 11 | 2901.0 |
| 12 | 2500.0 |
| 13 | NaN |
| 14 | NaN |
| 15 | 3393.0 |
| 16 | 3894.0 |
| 17 | 9000.0 |
| 18 | 10279.0 |
| 19 | 4400.0 |
| 20 | 6700.0 |
| 21 | 4999.0 |

Original Data:

| | |
|---|---|
| 0 | 0.000000 |
| 1 | 15666.000000 |
| 2 | 4200.000000 |
| 3 | 9052.000000 |
| 4 | 10406.000000 |
| 5 | 13500.000000 |
| 6 | 3583.000000 |
| 7 | 2700.000000 |
| 8 | 3400.000000 |
| 9 | 5050.000000 |
| 10 | 2750.000000 |
| 11 | 2901.000000 |
| 12 | 2500.000000 |
| 13 | 6578.995733 |
| 14 | 6578.995733 |
| 15 | 3393.000000 |
| 16 | 3894.000000 |
| 17 | 9000.000000 |
| 18 | 10279.000000 |
| 19 | 4400.000000 |
| 20 | 6700.000000 |
| 21 | 4999.000000 |
| 22 | 9400.000000 |
| 23 | 5250.000000 |
| 24 | 11333.000000 |
| 25 | 7000.000000 |
| 26 | 2405.000000 |

Data with no nans filled with mean value:

## 1.4 Generate Features/Predictors

**Discretize continuous data** - *descretize(data_frame,var,num)* - Using pandas cut function, it allows the data to go from a continuous variable to a categorical variable. The values are split into num categories or bins(in this case 4), where they go in this range.

$$[(20.912, 43.0] < (43.0, 65.0] < (65.0, 87.0] < (87.0, 109.0]]$$

**Dummy variable creation** - *dummy_var(data_frame,var)* - creating dummy variables from categorical variables. Good if placeholder is needed.

## 1.5 Build Classifier

**Logistic Regression** - *logReg(data_frame,IV,var_list)* -function takes in an independent variable and a list of dependent variables which you want to create a logistic regression. The function returns the accuracy with the original data corresponding to the variable, resulting in 83.48% accuracy with original data. This function uses pandas logistic regression function and model.fit in order to acquire the accuracy results. Below is the output of the results.

```
                                               0                           1
0                                      Intercept      [-0.05067901911182967]
1        RevolvingUtilizationOfUnsecuredLines    [-0.0002813058192122971]
2                               SeriousDlqin2yrs       [0.15571588239572673]
3    Q("NumberOfTime30-59DaysPastDueNotWorse")       [0.08739428774735851]
4                                      DebtRatio    [-0.0004078842573678497]
5                                  MonthlyIncome   [-0.00013154733922158266]
6              NumberOfOpenCreditLinesAndLoans      [-0.07948117826898649]
7                     NumberOfTimes90DaysLate       [0.05516669351265237]
8                 NumberRealEstateLoansOrLines      [0.014790228840361956]
9    Q("NumberOfTime60-89DaysPastDueNotWorse")       [0.02577974150417159]
10                            NumberOfDependents      [0.012740166916521465]
0.834876823436838
```

## 1.6 Evaluate Classifier

Though the accuracy may appear high, it is probably due to the fact that it was trained and tested on the same data. Considering this, this accuracy actually doesn't seem to be very good. Additionally, when using the the fill_empty function to replace all the incomes with the mean monthly income, the accuracy increases to 90.8% with the results below. MonthlyIncome was used as it has the most missing data among the other options. This goes to further show how easily the data is influenced, even though we put the mean of the data. Accuracy weights all errors equally. Because this, the importance of outcome of false negatives or false positives will have the same weight regardless if they deserve to or not.

```
Name: MonthlyIncome, Length: 41016, dtype: float64
                                             0                          1
0                                    Intercept        [-0.442258096771196]
1        RevolvingUtilizationOfUnsecuredLines    [-3.482660700586518e-05]
2                             SeriousDlqin2yrs        [1.3633875383536422]
3    Q("NumberOfTime30-59DaysPastDueNotWorse")         [0.375304887224147]
4                                    DebtRatio   [-0.00016431947685750607]
5                                MonthlyIncome    [-7.711078215887043e-05]
6              NumberOfOpenCreditLinesAndLoans      [-0.10040845364733057]
7                     NumberOfTimes90DaysLate       [0.21417061367951398]
8                 NumberRealEstateLoansOrLines        [0.1102974894107628]
9    Q("NumberOfTime60-89DaysPastDueNotWorse")       [0.04328230576710495]
10                          NumberOfDependents      [0.021190183516032315]
0.9080564130500575
```