CAPP 30254: Maching Learning for Public Policy

Assignment 3: Machine Learning Pipeline

Yuliana Zamora

Due: May 20, 2018

GitHub Page

1 ML Pipeline 'Improved'

The biggest addition to the pipeline was the functions that split data into training and test set, and running the this data across several classifiers. The classifiers and their parameters include:

- LogisticRegression: $'C' : [10 * * -1, 10 * * -2, 10 * * -3],' penalty' : ['l1','l2']$
- KNeighborsClassifier:$'n\_neighbors' : [5, 10, 25, 100],' p' : [1, 2, 3],' n\_jobs' : [2]$
- DecisionTreeClassifier:$'max_depth' : [5, 10, 15],' min\_samples\_leaf' : [2, 5, 10]$
- RandomForestClassifier:$'n\_estimators' : [100],' criterion' : ['gini',' entropy']'max\_features' : ['sqrt',' log2'],' max_depth'$
- GradientBoostingClassifier:$'learning_rate' : [.1, .01],' n_estimators' : [100],' max_features' : ['sqrt',' log2'],' max\_depth' : [$
- BaggingClassifier:$'max_samples' : [.1, .25, .65],' n\_jobs' : [4]$
- SVC:$'kernel' : ['linear',' rbf'],' gamma' : [10, 1, .1, .01],' C' : [10, 1, .1, .01],' probability' : [True]$

Specifically, the functions that were created to do all of the analysis, modeling, and data splitting are: temp_val, class_comp, extract_train_test_sets, plot_precision_recall, get_metrics.

All the above models were run to train and test according to the combinations of their parameters. The results were then run through the metrics function to get the precision and recall plots. Because there were so many charts that were generated from the combination of all the parameters, below will be a list of only a few. After much debugging, the plots of the different parameters finally aligned to what made sense. From the lecture notes, normally as recall goes up, the precision goes down. That being said, it is difficult to compare the results across the metrics as they had similar performances. My recommendation to someone that is working on the model is to look at a leave one out strategy to see if that causes any differences in the results. With the current implementations, most of the classifiers seem to do a very similar job in terms of precision and recall with small variances on he it deals with low amounts of given population.

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.(sorry for the order of the graphs). The caption under each classification result graph will have a brief interpretation of the results.

2 Components of pipeline

2.1 Read Data

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

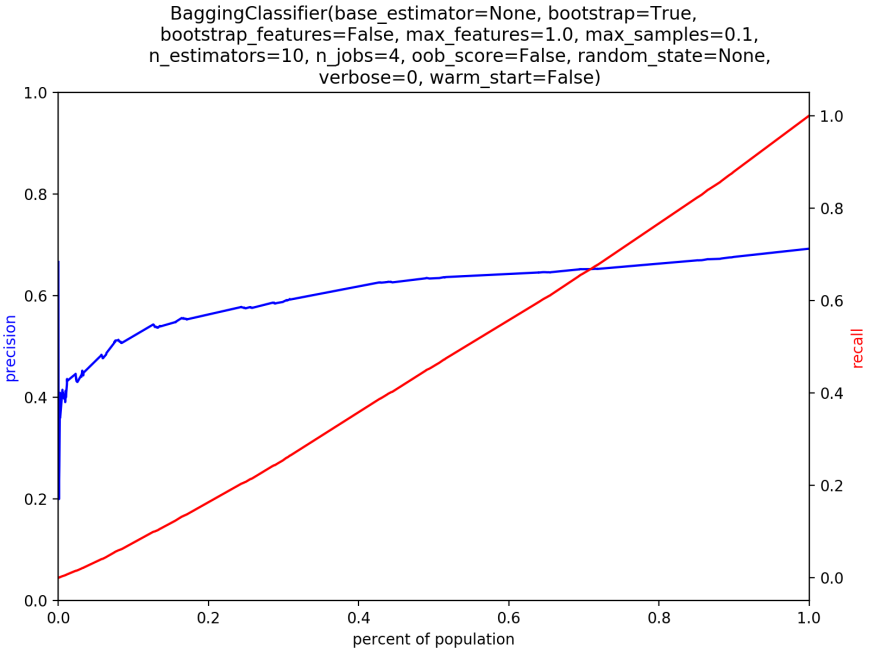

Figure 1: Bagging classifier (an ensemble meta-estimator). The results of using the bagging classifier are most similar to using the knearestneighbor classifier. I'm interested in where the jagged results at the beginning of the graph mean, but like the following graphs, it has similar results after .2 mark.

2.2 Initial Data Exploration

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. Figure 8 shows the statistics of the debt ratio and Figure 9 shows 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.

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 (see Figure 10). 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. I guess it makes sense that the number of real estate loans receives will mean the person will have less likelihood to have days past due.

Correlation Graphs - Multiple graphs are created using pairplot function. The graphs shown in Figure 11 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 in Figure 12.

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. Figure 13 shows the output of low and high distribution using data from seriousdllqin2yrs data.

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. If Figure 14, you can see the

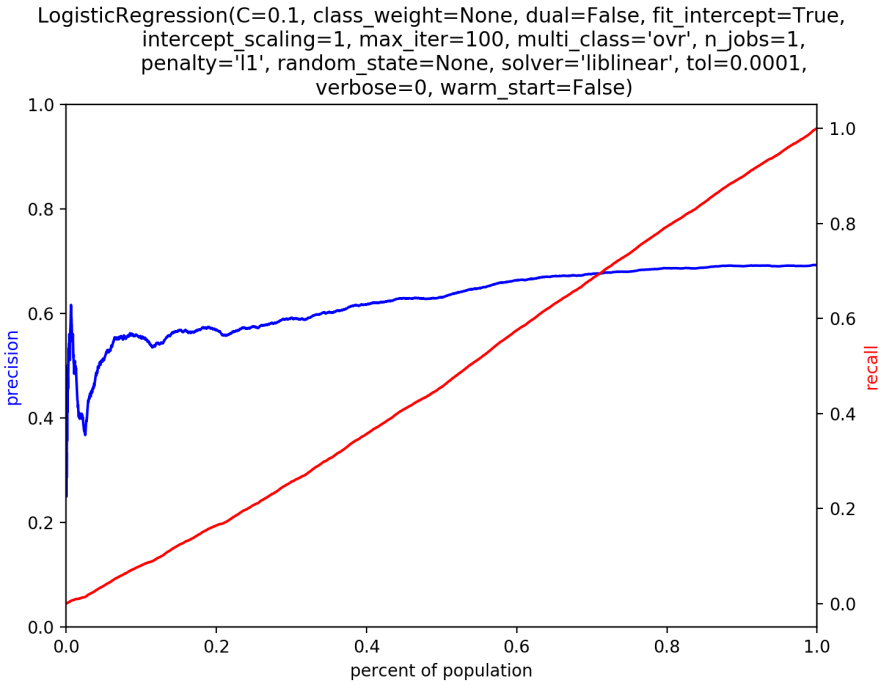

Figure 2: Logistic regression. Similar to 7, this logistic regression implementation has some more volatility at the beginning when there isn't much data available. Like the other classifiers, the results after 20% of the population is used is similar.

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. See Figure 15.

2.3 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).
To Binary - *to_binary(df,array_col)* converts the array given in the array_col from the data frame into a binary output. In this specific function, it will convert every 't' to 1 or 0 otherwise. Further updates need to be made to ensure other circumstances will be looked at.

2.4 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.

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

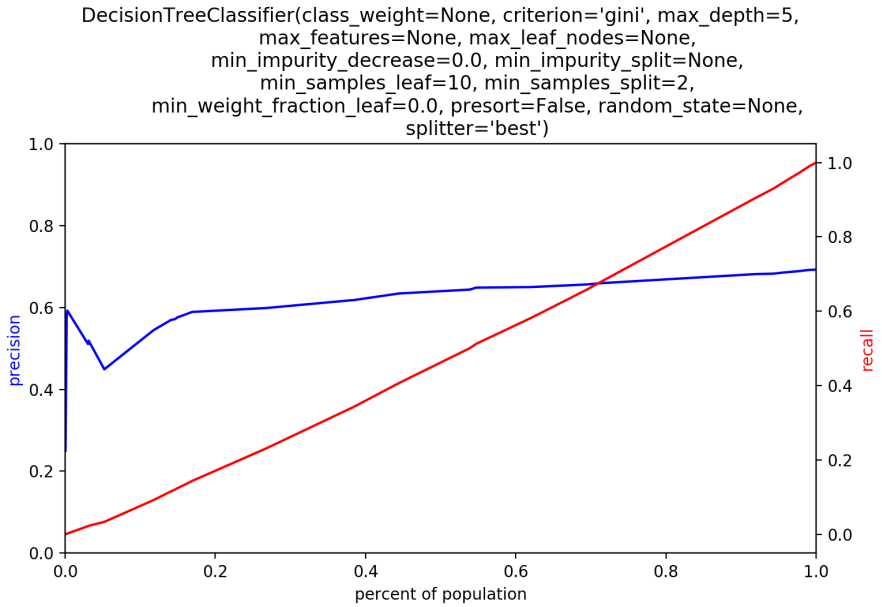

Figure 3: Decision tree classifier using the Gini criterion. Gini is intended for continuous attributes, and Entropy for attributes that occur in classes (e.g. colors). I'm assuming the large drop is because the max depth is only at 5 while in the following graph it is higher. Another difference is the min number of samples leafs is 5x greater than the result from below.

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

2.5 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.

2.6 Conclusion

In attempt to give advice on which classifier to use, considering how all classifiers gave very similar results (only a select were chosen for the purpose of brevity), it seems like I would go between bagging and random forest. Both take into account more randomnesss and are less likely to over-fit. Being that all features were used, random forest can be furthered use to analyze feature importance. Additionally, because there's an easy way we can parallelize the RF classifier (along with any other classifiers), the training of a large rf didn't take as long as SVC. Also, as previously stated, an attempt to see how certain features affect the results, a leave one out approach can be taken into account.

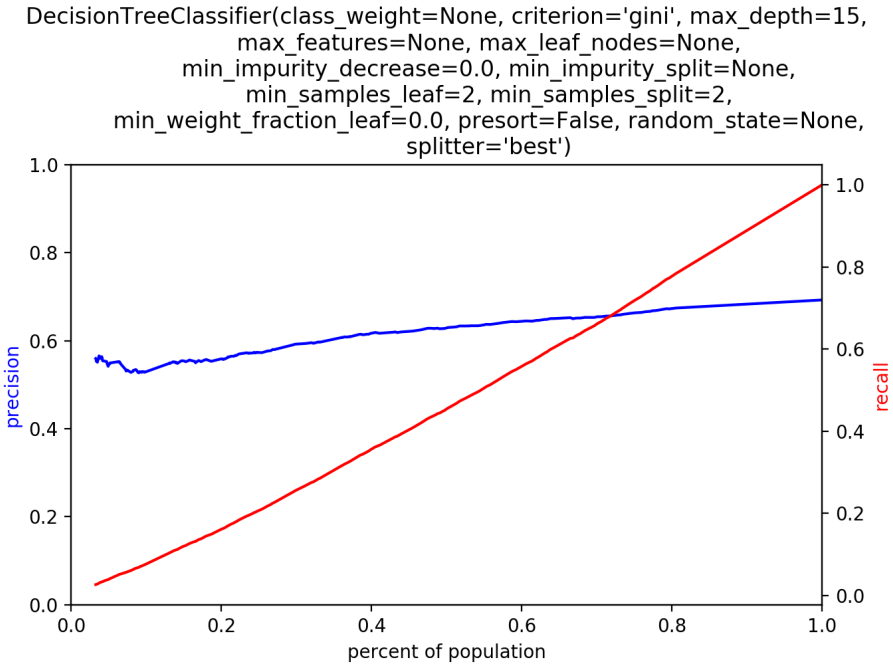

Figure 4: Decision tree classifier. Precision stays a bit more consistent here as the tree can be much deeper and the sample leaves are much smaller. Though these results might be a little better than the above graph, it could be potentially over-fitting the data.

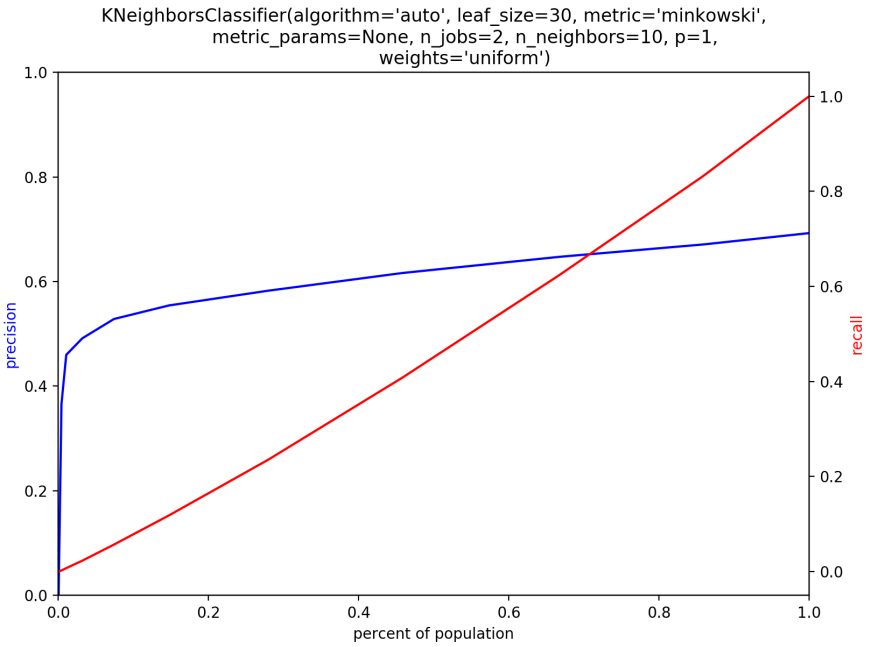

Figure 5: K-nearest neighbor with minkowski parameter that combines Euclidean space and time into a four-dimensional manifold. It performs similar to 4 but not as well with very little data.

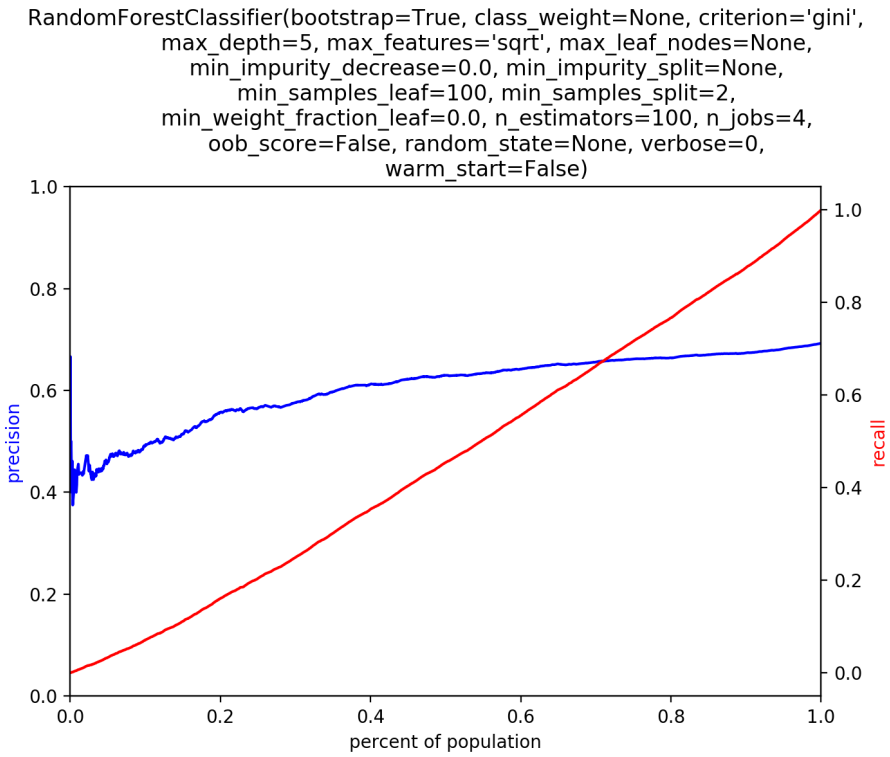

Figure 6: Random forest classifier, which employs an ensemble of decision-tree models. Though it was expected random forest would automatically do better, it seems like the parameters chosen for this instance weren't very good. The random forest in this graph had a min sample leaf of 100 without many other of the parameters set.

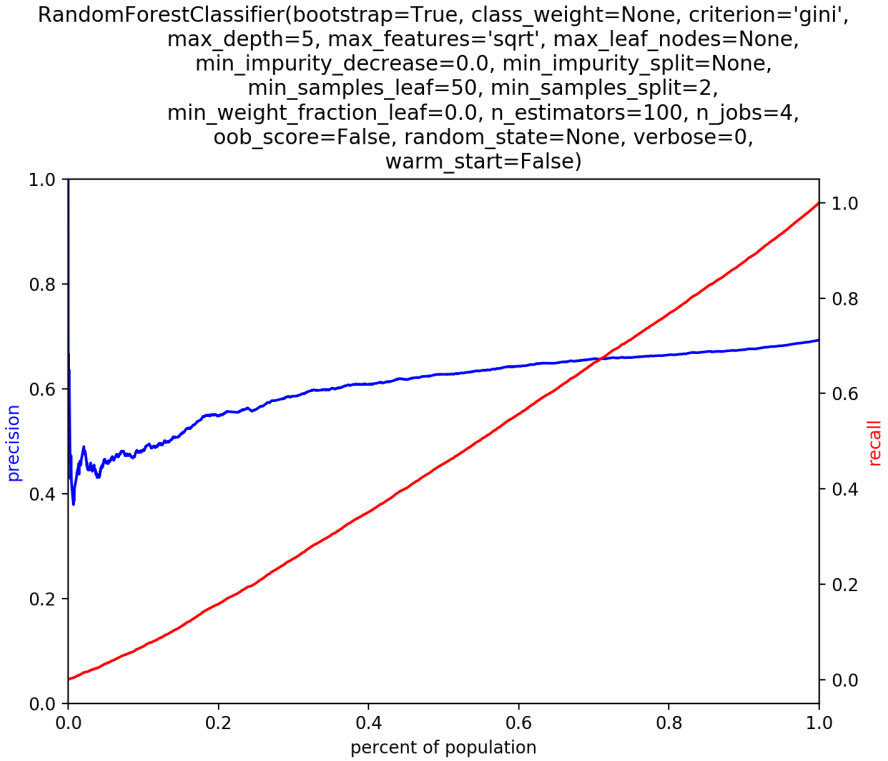

Figure 7: Random forest classifier using the Gini criterion. Gini is intended for continuous attributes, and Entropy for attributes that occur in classes (e.g. colors). The change in this one compared to the previous one, was the decrease of the minimum of sample leafs - reduced by 2x. This seemed to have caused a little more volatility in the beginning, but produced similar results in the end.

```
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
```

Figure 8: Statistics of the debt ratio data.

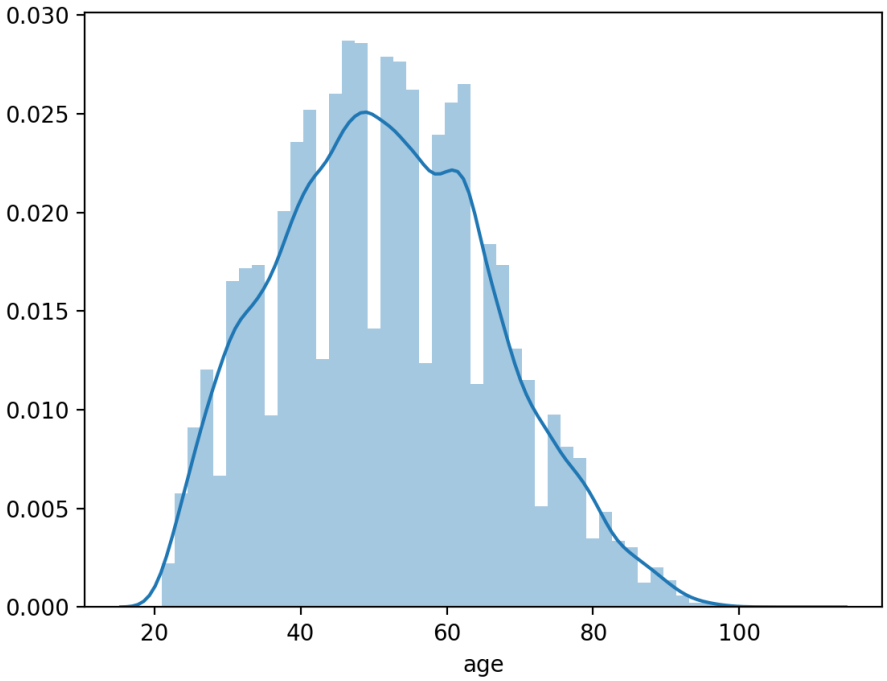

Figure 9: Age histogram.

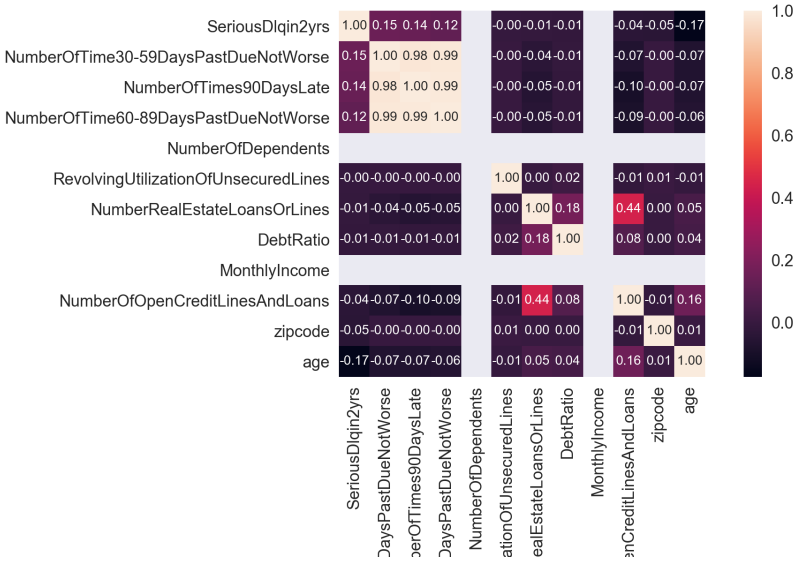

Figure 10: Correlation heat map.

Figure 11: Correlation graphs.

	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

Figure 12: Missing 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]]
```

Figure 13: Univariate analysis.

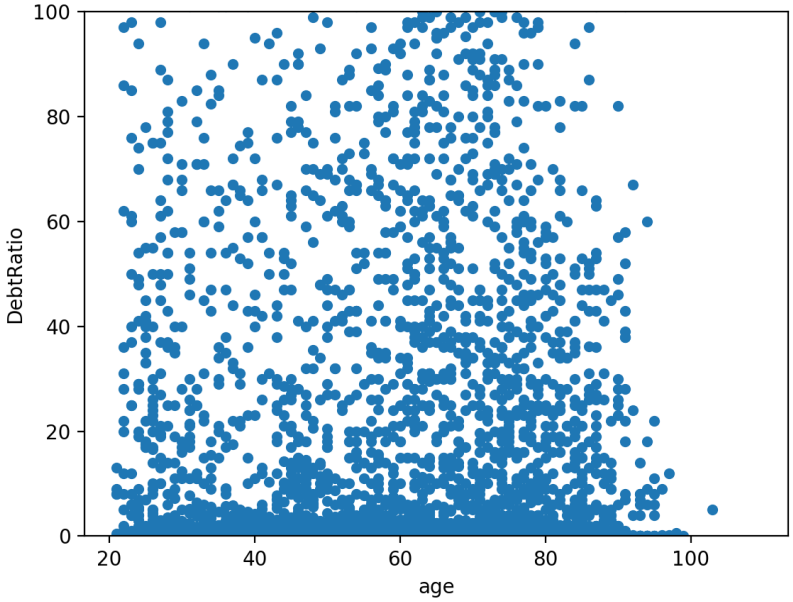

Figure 14: Bivariate Analysis.

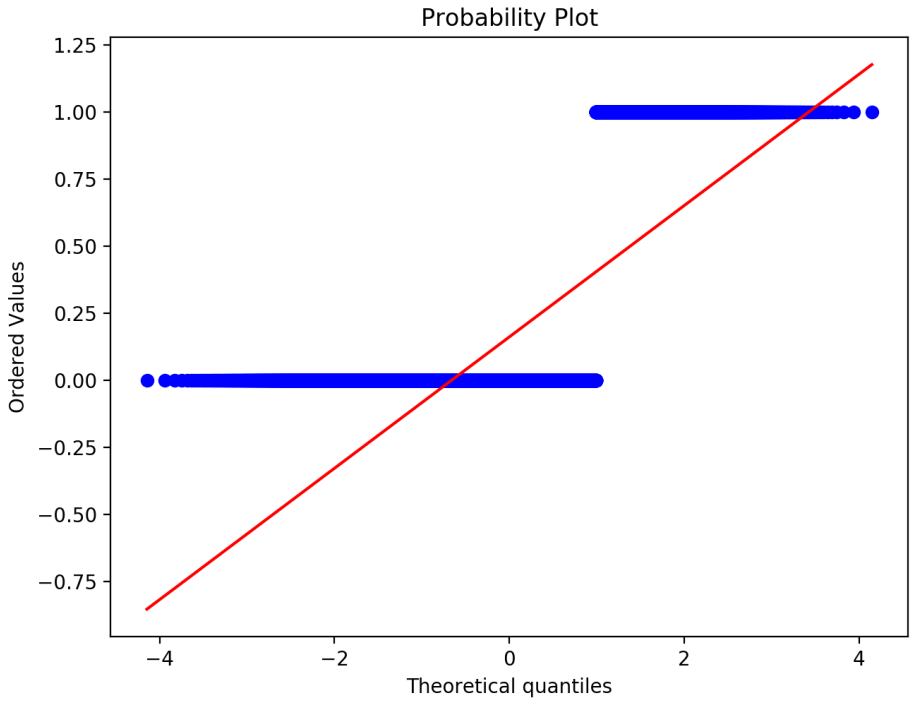

Figure 15: Normal probability plot.

Figure 16: Original Data:

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

Figure 17: Data with no nans filled with mean value:

```
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
```

```
                                      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
```

Figure 18: Logistic regression.

```
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
```

Figure 19: New mean, after filling empty values.