pipeline_demo

April 17, 2019

1 ML Pipeline Demo

Machine Learning for Public Policy
Camilo Arias
pipeline functions in pipeline.py

```
In [1]: %load_ext autoreload
        %autoreload 2
        import pipeline as ppln
        import numpy as np
```

1.1 Importing data

Function to read from csv and function to download the ZIP bowndaries of Chicago

```
In [2]: credit_df = ppln.load_credit_data('inputs/credit-data.csv')
        zip_gdf = ppln.load_zipcode_area()
```

```
WARNING:root:Requests made without an app_token will be subject to strict throttling limits.
```

1.2 Explore Data

Function to see basic summary statistics of all variables. Takes a list of columns and a list of percentiles. Default all columns and 0.25, 0.5 and 0.75 percentiles.

```
In [3]: ppln.see_summary_stats(credit_df, ['PersonID', 'SeriousDlqin2yrs', 'age', 'zipcode'],
```

	PersonID	SeriousDlqin2yrs	age	zipcode
count	41016.000000	41016.000000	41016.000000	41016.000000
mean	115800.154598	0.161400	51.683489	60623.824166
std	28112.723511	0.367904	14.746880	11.984357
min	22.000000	0.000000	21.000000	60601.000000
1%	10034.050000	0.000000	24.000000	60601.000000
25%	106539.750000	0.000000	41.000000	60618.000000
50%	119901.500000	0.000000	51.000000	60625.000000
75%	134698.250000	0.000000	62.000000	60629.000000
99%	149396.850000	1.000000	87.000000	60644.000000
max	149999.000000	1.000000	109.000000	60644.000000

```
In [4]: ppln.see_summary_stats(credit_df, ['NumberOfOpenCreditLinesAndLoans', 'NumberRealEstate
```

	NumberOfOpenCreditLinesAndLoans	NumberRealEstateLoansOrLines
count	41016.000000	41016.000000
mean	8.403477	1.008801
std	5.207324	1.153826
min	0.000000	0.000000
1%	0.000000	0.000000
5%	2.000000	0.000000
25%	5.000000	0.000000
50%	8.000000	1.000000
75%	11.000000	2.000000
95%	18.000000	3.000000
99%	25.000000	5.000000
max	56.000000	32.000000

```
In [5]: ppln.see_summary_stats(credit_df, ['RevolvingUtilizationOfUnsecuredLines', 'DebtRatio'
```

	RevolvingUtilizationOfUnsecuredLines	DebtRatio	MonthlyIncome
count	41016.000000	41016.000000	3.304200e+04
mean	6.375870	331.458137	6.578996e+03
std	221.618950	1296.109695	1.344683e+04
min	0.000000	0.000000	0.000000e+00
1%	0.000000	0.000000	0.000000e+00
5%	0.000000	0.004569	1.325000e+03
25%	0.034310	0.176375	3.333000e+03
50%	0.189730	0.369736	5.250000e+03
75%	0.667160	0.866471	8.055750e+03
95%	1.000000	2337.000000	1.450000e+04
99%	1.194705	4856.850000	2.500000e+04
max	22000.000000	106885.000000	1.794060e+06

```
In [6]: ppln.see_summary_stats(credit_df, ['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTi
```

	NumberOfTime30-59DaysPastDueNotWorse	NumberOfTimes90DaysLate \
count	41016.000000	41016.000000
mean	0.589233	0.419592
std	5.205628	5.190382
min	0.000000	0.000000
1%	0.000000	0.000000
5%	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
95%	2.000000	1.000000
99%	4.000000	4.000000
max	98.000000	98.000000

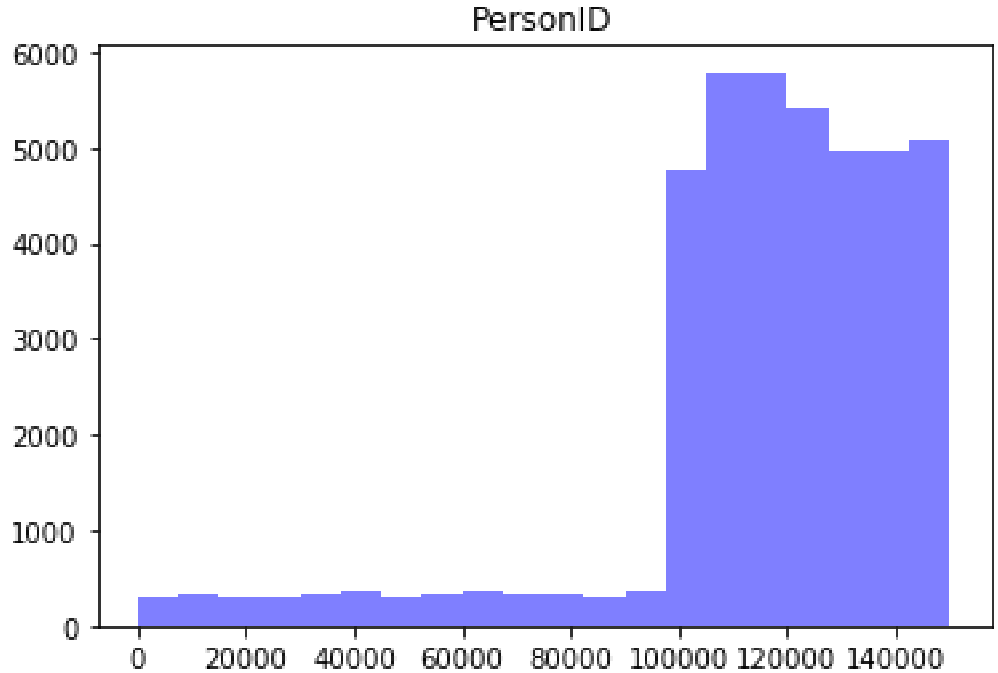
```
       NumberOfTime60-89DaysPastDueNotWorse
count                              41016.000000
mean                                   0.371587
std                                    5.169641
min                                    0.000000
1%                                     0.000000
5%                                     0.000000
25%                                    0.000000
50%                                    0.000000
75%                                    0.000000
95%                                    1.000000
99%                                    2.000000
max                                   98.000000
```

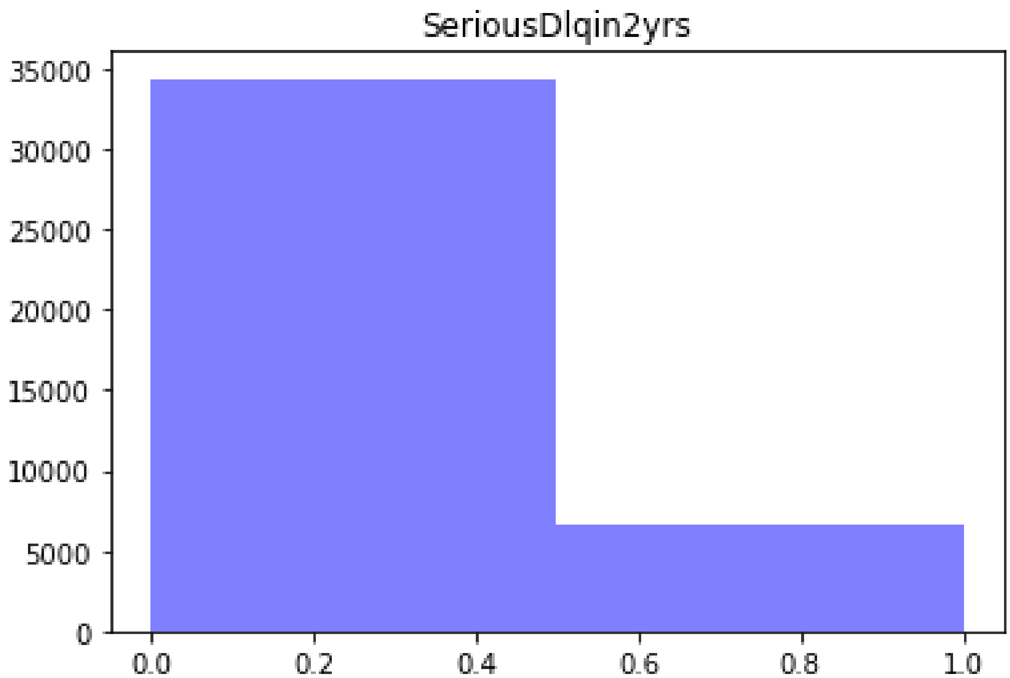
It possible to see that -NumberOfTime60-89DaysPastDueNotWorse
-NumberOfTimes90DaysLate
-NumberOfTime30-59DaysPastDueNotWorse
-RevolvingUtilizationOfUnsecuredLines
-DebtRatio
-MonthlyIncome
have some extremely high values, because the maximum value is extremely higher compared to the 99th percentile.

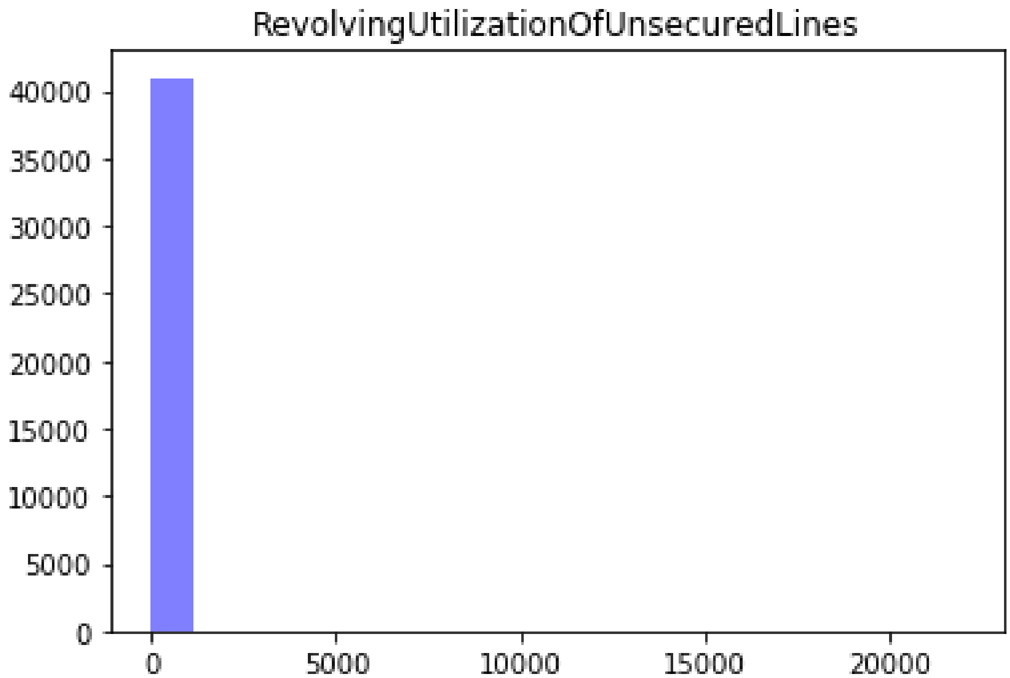
Function to see histogram all variables. Takes a list of columns and a dictionary of columns mapped to percentile range to exclude extreme values. Default all columns and all values. If categorical or string column with less than 16 unique values, bar plot is produced.

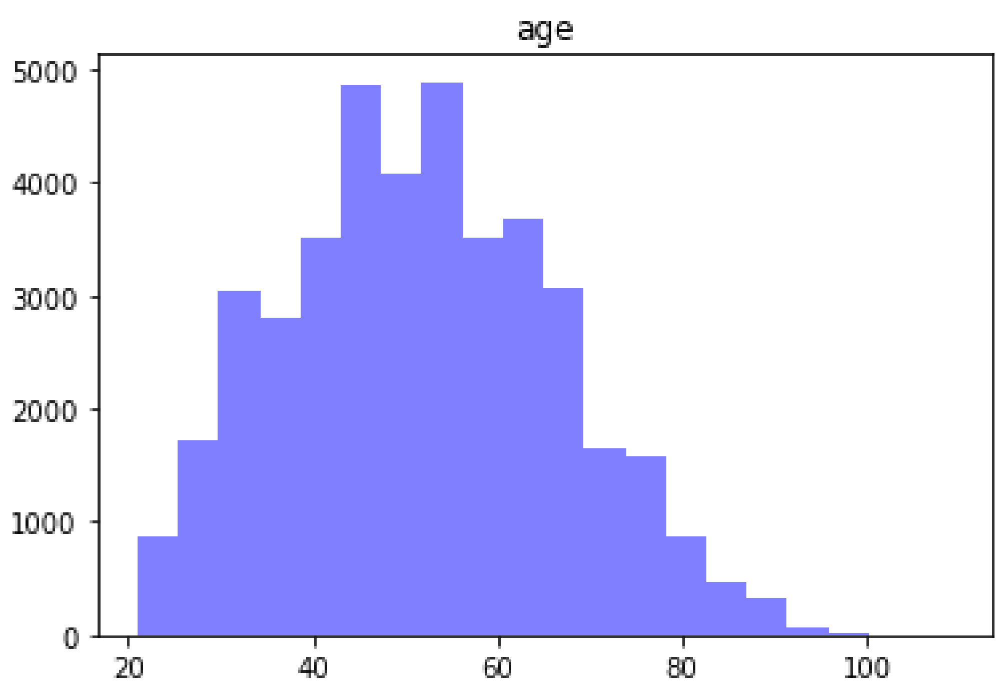
```
In [7]: ppln.see_histograms(credit_df)
```

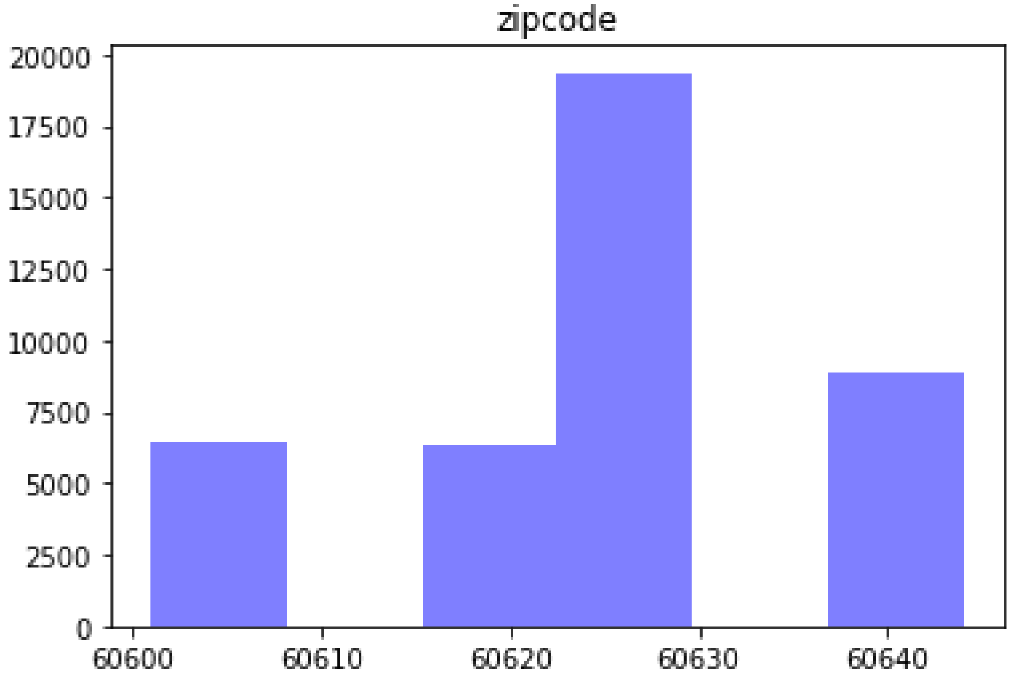
```
/anaconda3/lib/python3.6/site-packages/numpy/lib/histograms.py:754: RuntimeWarning: invalid val
  keep = (tmp_a >= first_edge)
/anaconda3/lib/python3.6/site-packages/numpy/lib/histograms.py:755: RuntimeWarning: invalid val
  keep &= (tmp_a <= last_edge)


<Figure size 432x288 with 0 Axes>
```

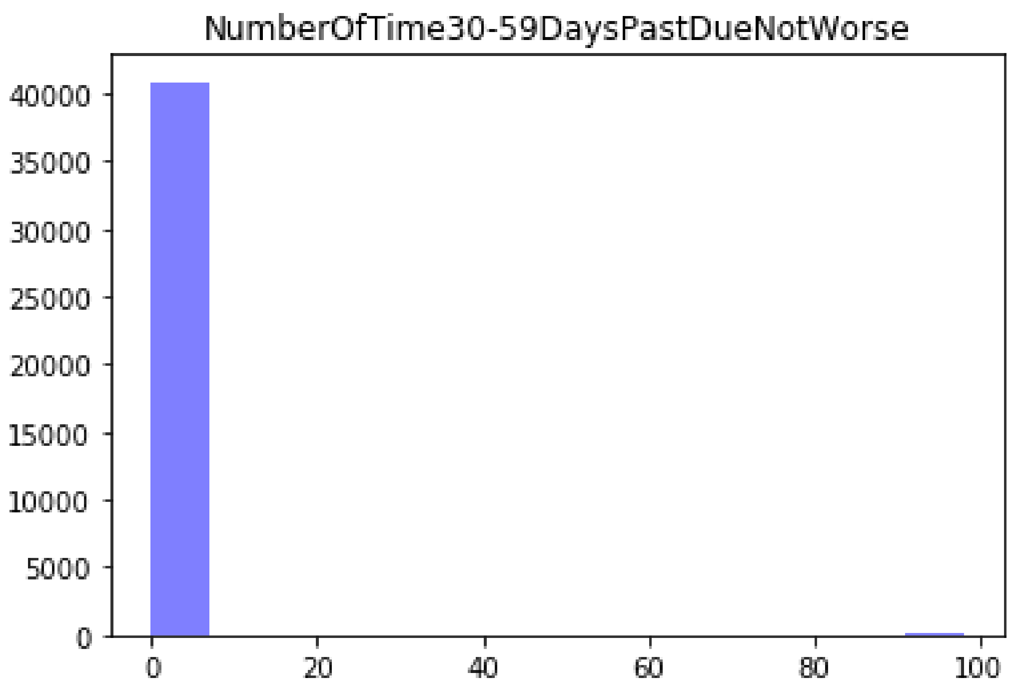

PersonID
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
120000
140000

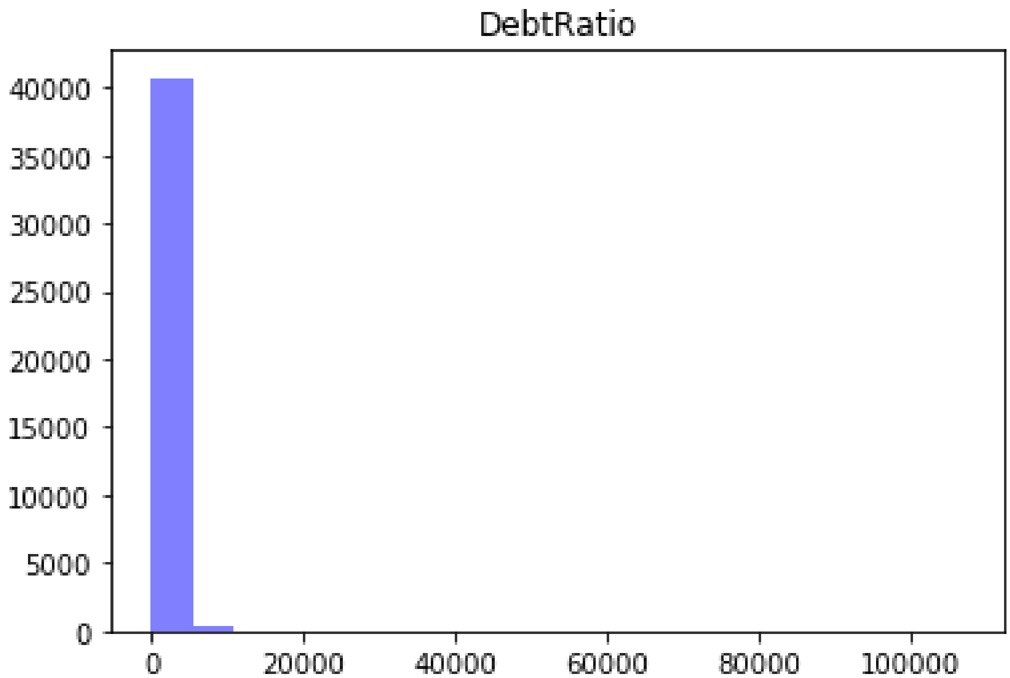

SeriousDlqin2yrs
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

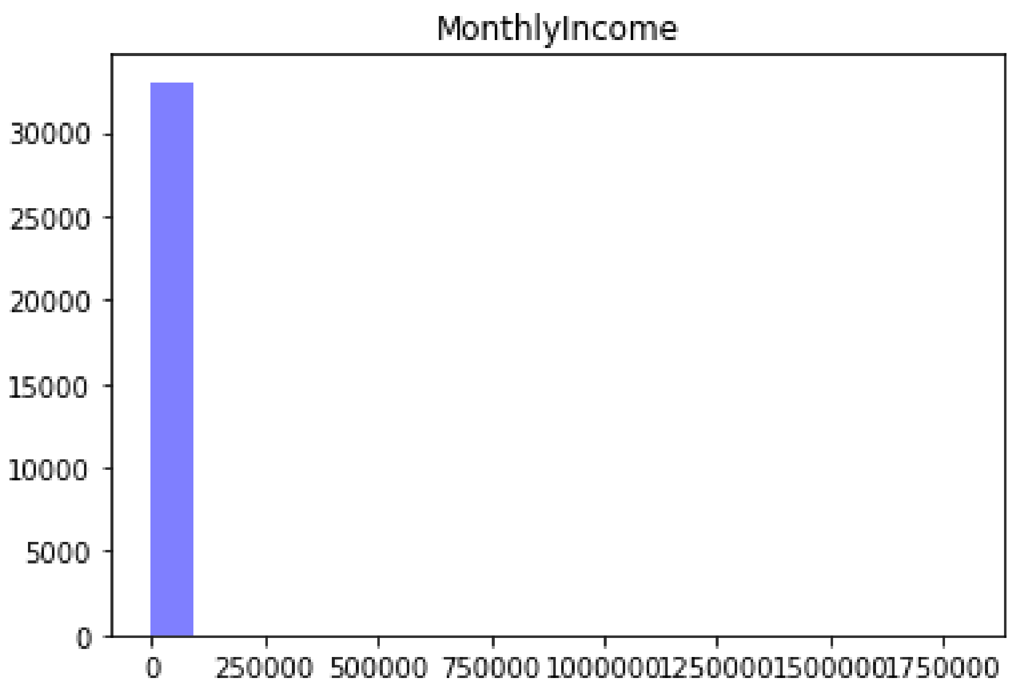

RevolvingUtilizationOfUnsecuredLines
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000

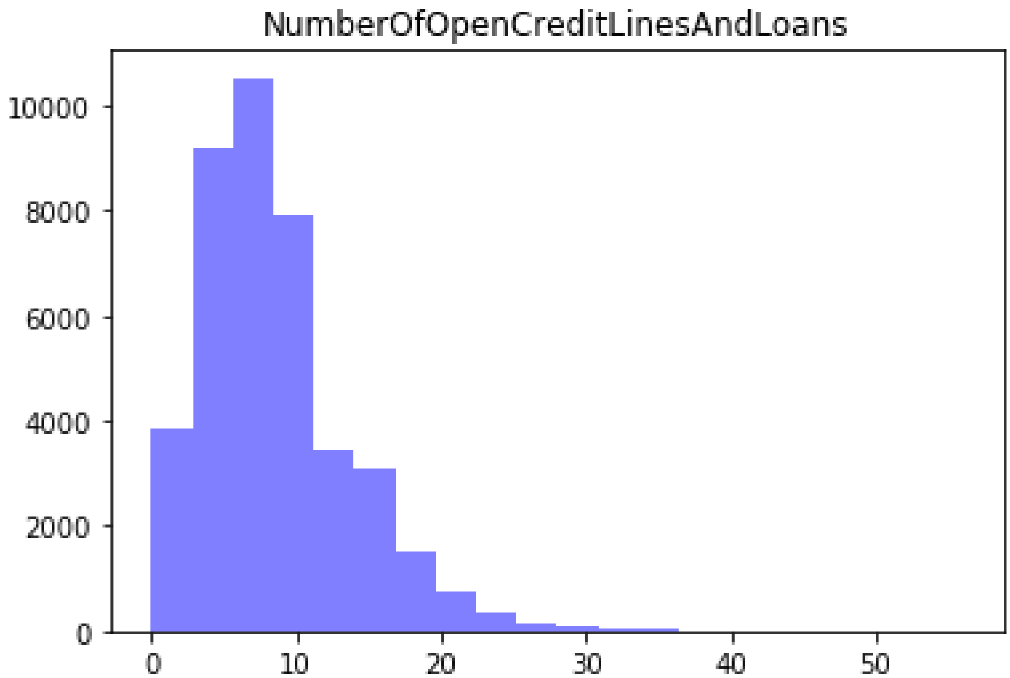

age
5000
4000
3000
2000
1000
0
20
40
60
80
100

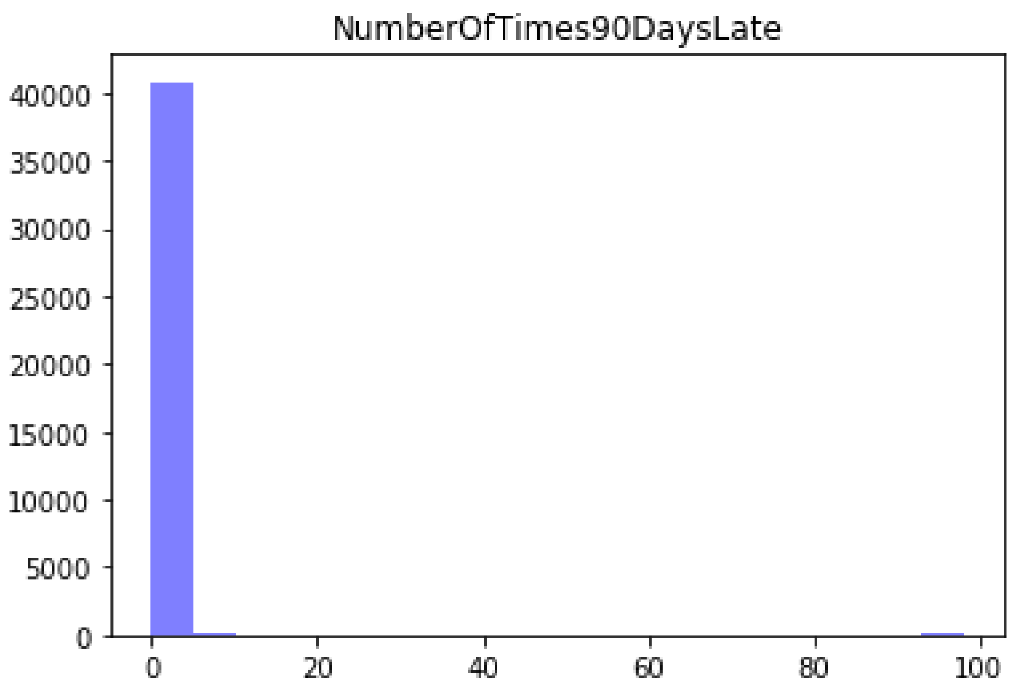

zipcode
20000
17500
15000
12500
10000
7500
5000
2500
0
60600
60610
60620
60630
60640

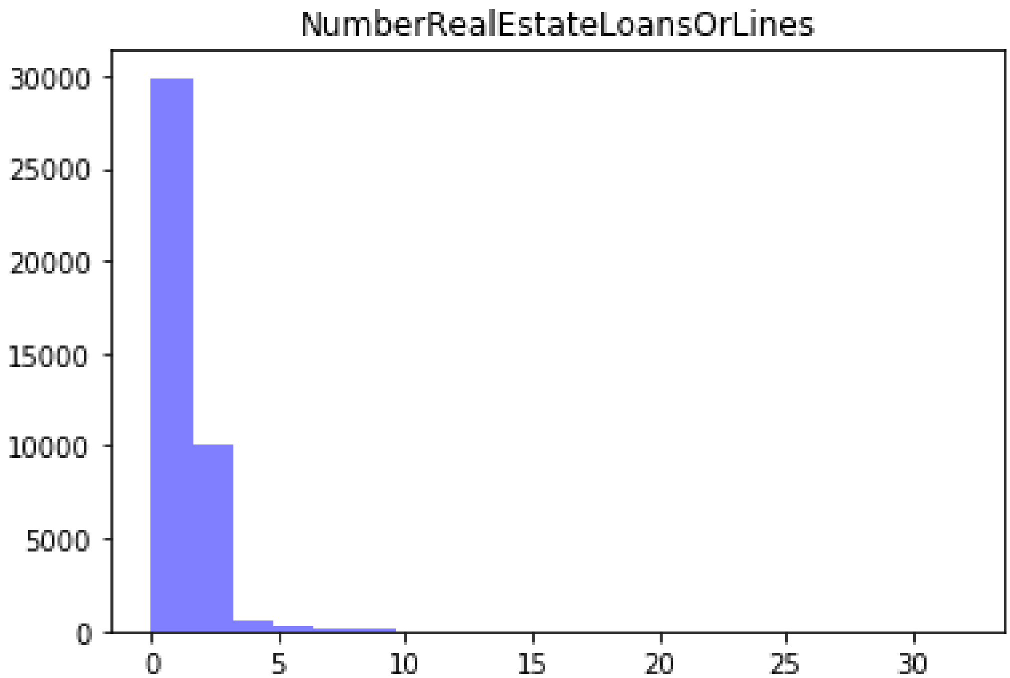

NumberOfTime30-59DaysPastDueNotWorse
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80
100

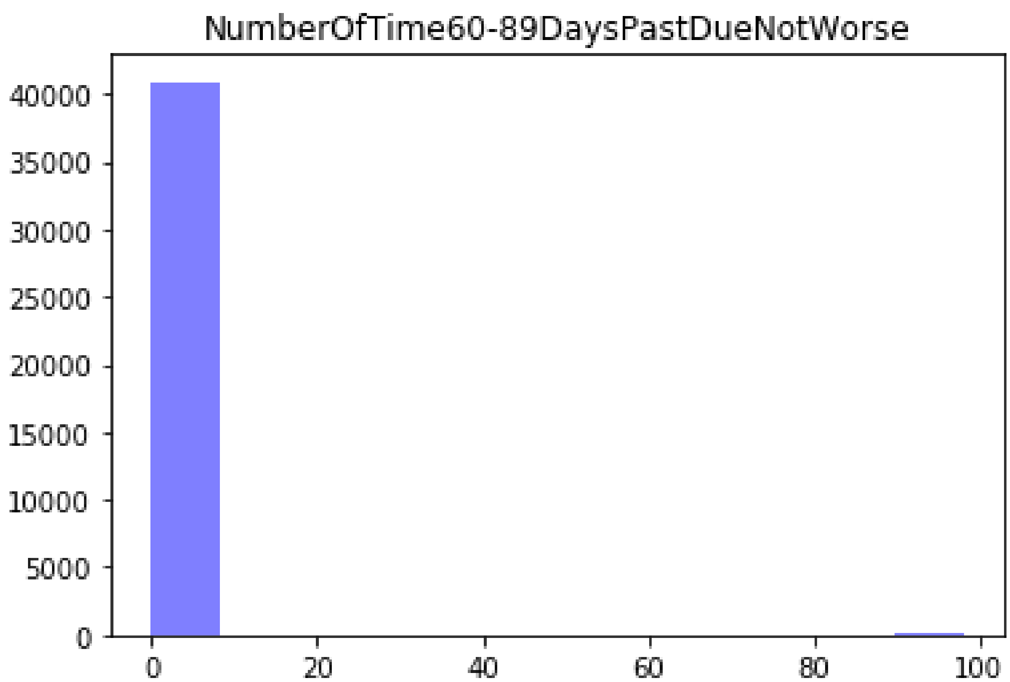

DebtRatio
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000

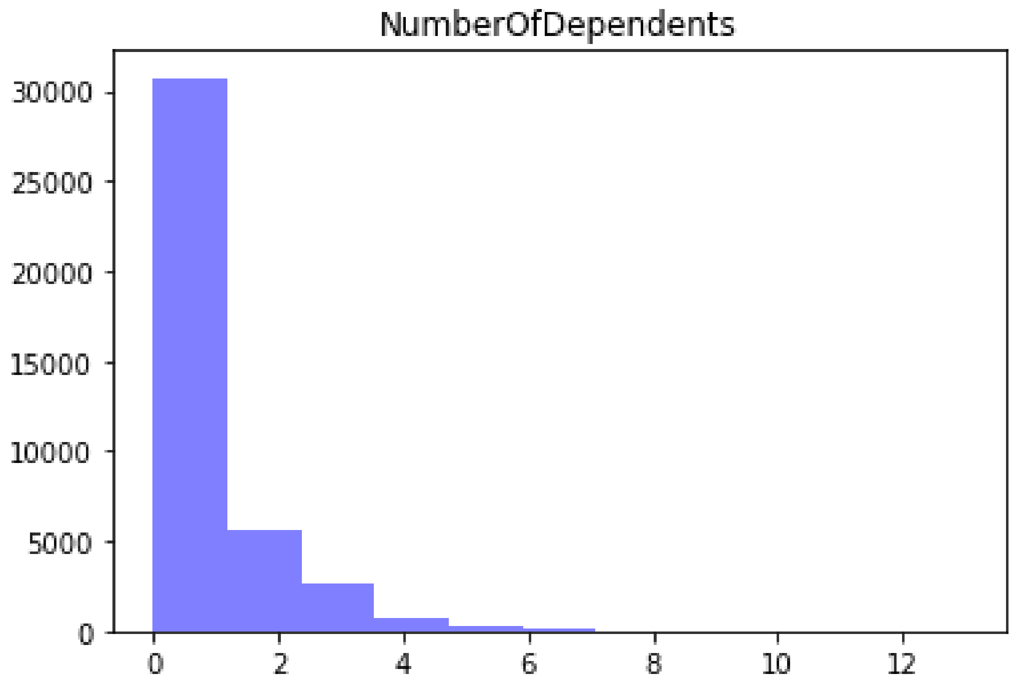

MonthlyIncome
30000
25000
20000
15000
10000
5000
0
0
250000
500000
750000
1000000
1250000
1500000
1750000


NumberOfOpenCreditLinesAndLoans
10000
8000
6000
4000
2000
0
0
10
20
30
40
50


NumberOfTimes90DaysLate
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80
100


NumberRealEstateLoansOrLines
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

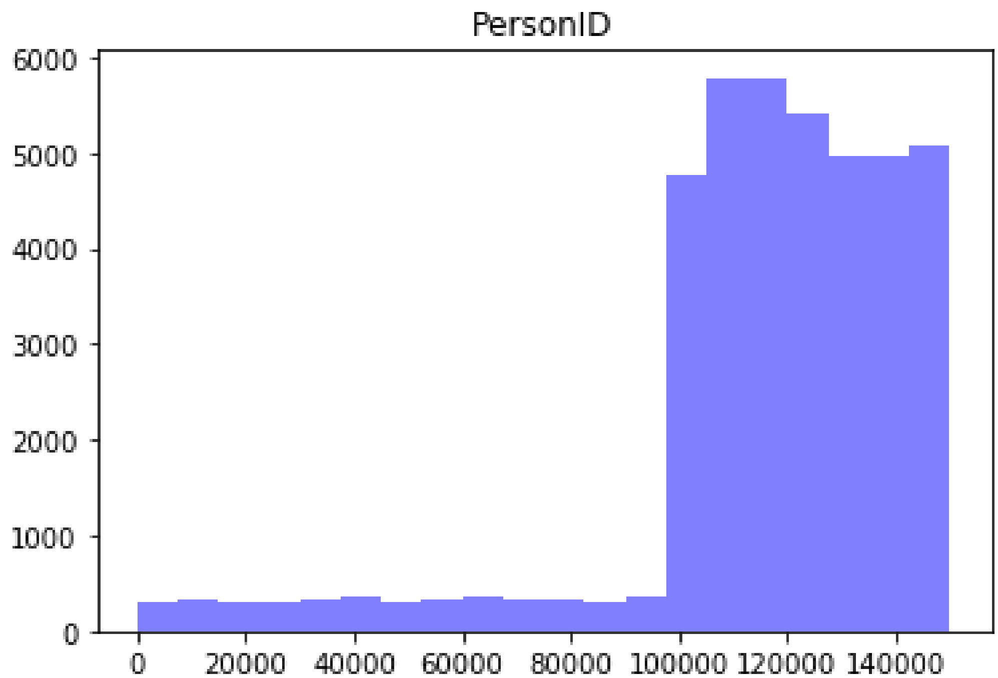

NumberOfTime60-89DaysPastDueNotWorse
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80
100

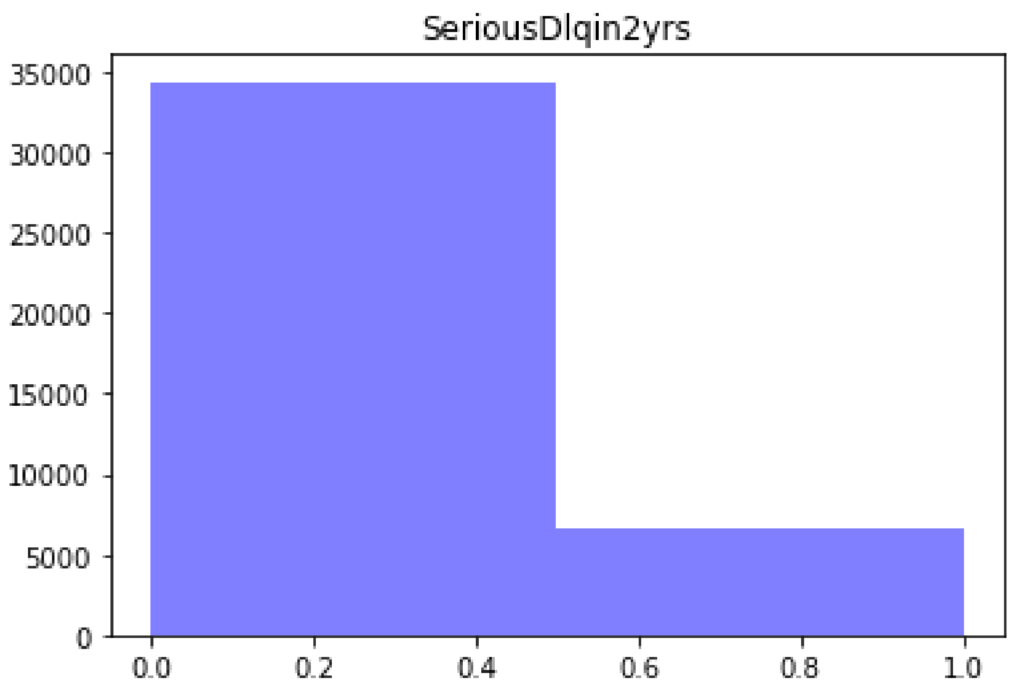


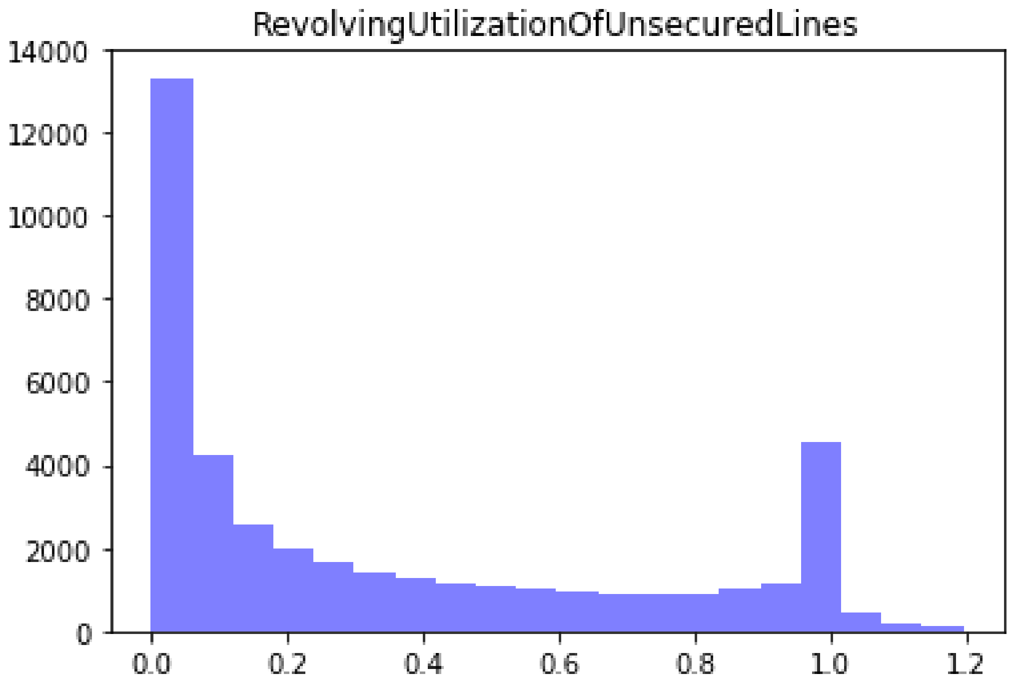
These histograms also evidence that the mentioned variables have some extremely high values. By specifying the restrict parameter, we can limit the extremely high values of these variables and see their histograms.

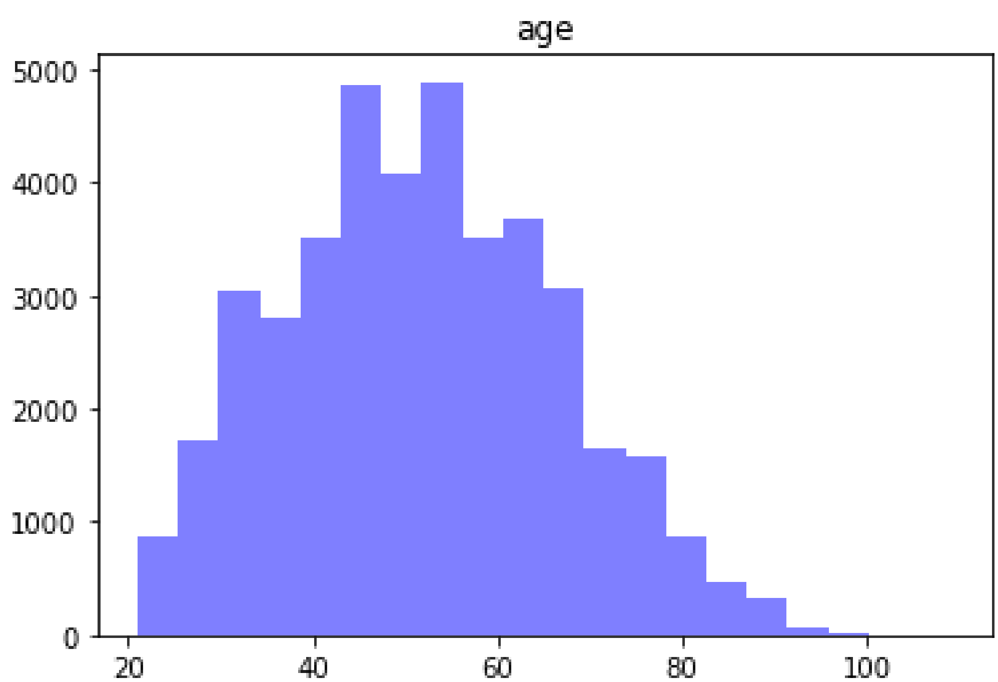
```
In [8]: restrictions = {
            'NumberOfTime60-89DaysPastDueNotWorse': [0, 0.99],
            'NumberOfTimes90DaysLate': [0, 0.99],
            'NumberOfTime30-59DaysPastDueNotWorse': [0, 0.99],
            'RevolvingUtilizationOfUnsecuredLines': [0, 0.99],
            'DebtRatio': [0, 0.99],
            'MonthlyIncome': [0, 0.99]}
        ppln.see_histograms(credit_df, restrict = restrictions)
```

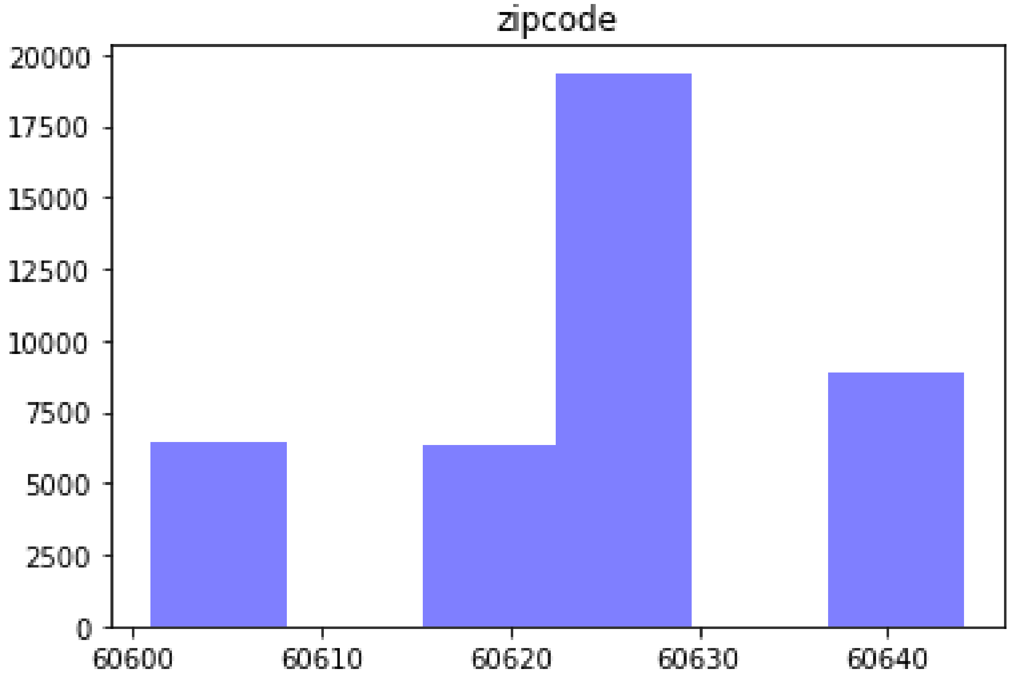
```
<Figure size 432x288 with 0 Axes>
```

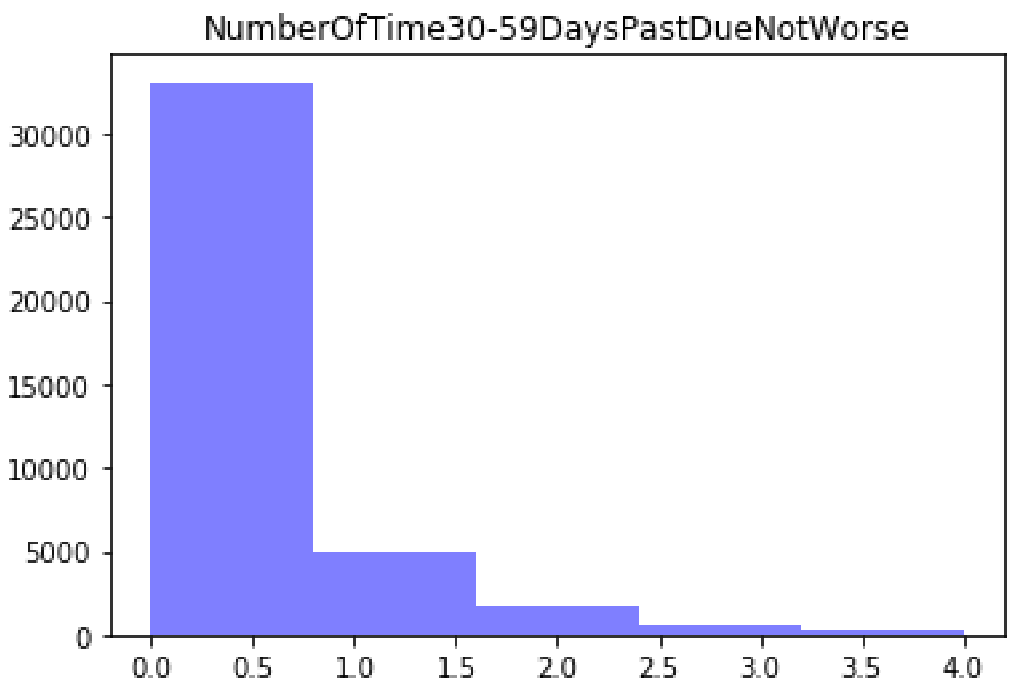

PersonID
6000
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000
120000
140000

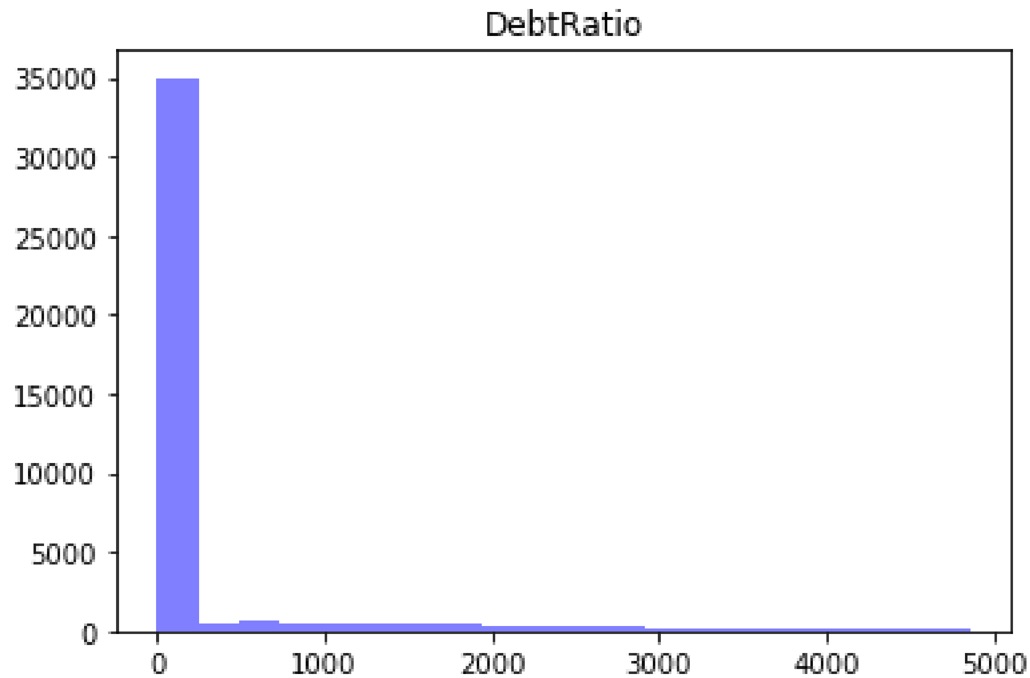

SeriousDlqin2yrs
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

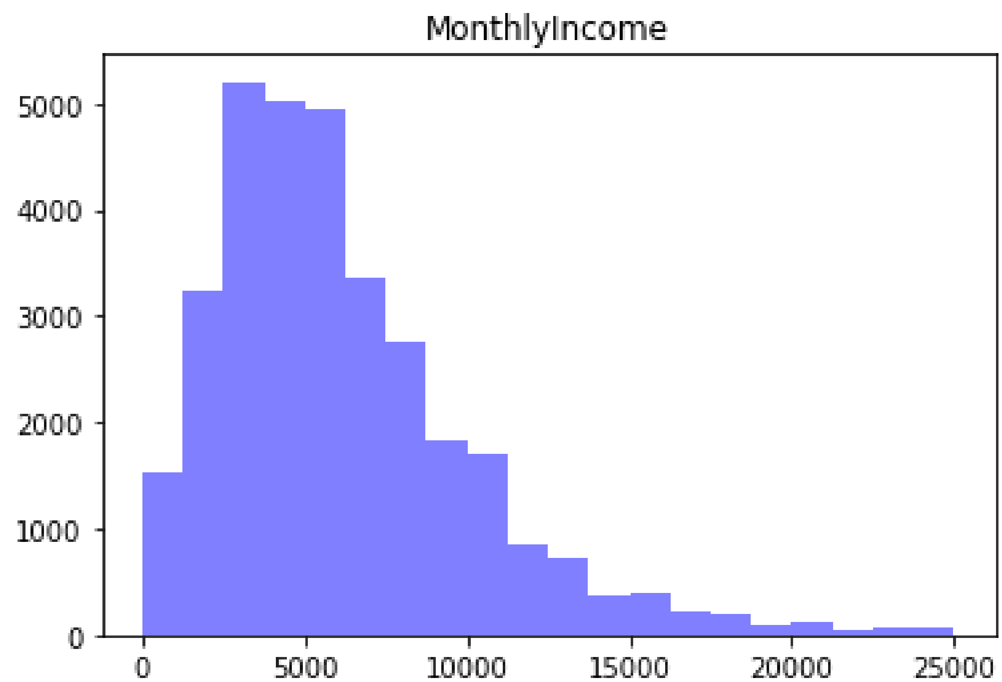

RevolvingUtilizationOfUnsecuredLines
14000
12000
10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2

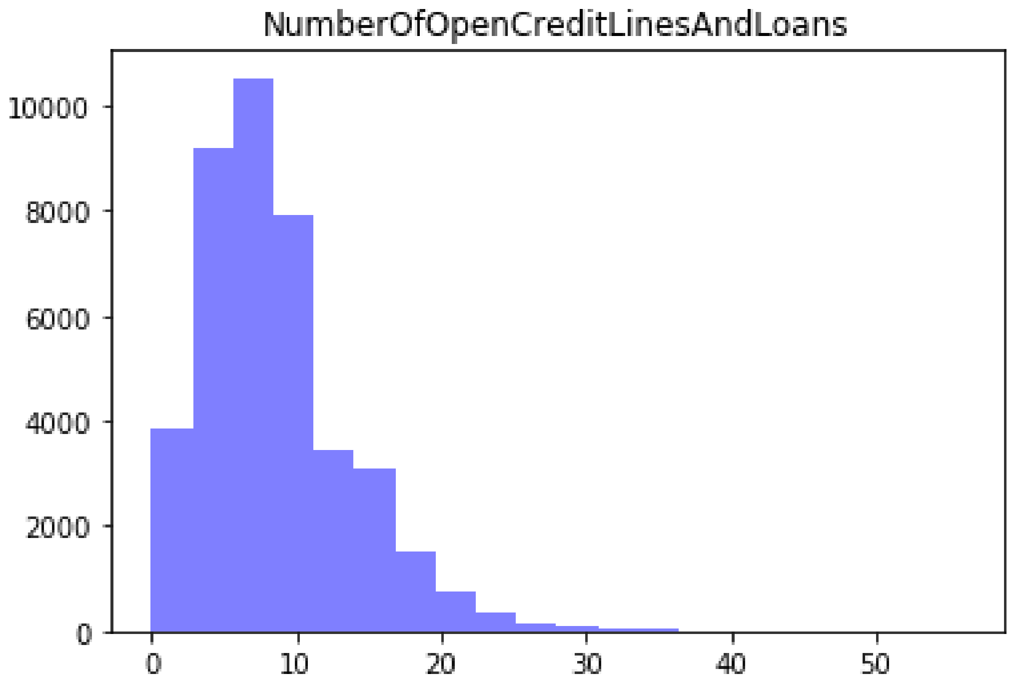

age
5000
4000
3000
2000
1000
0
20
40
60
80
100

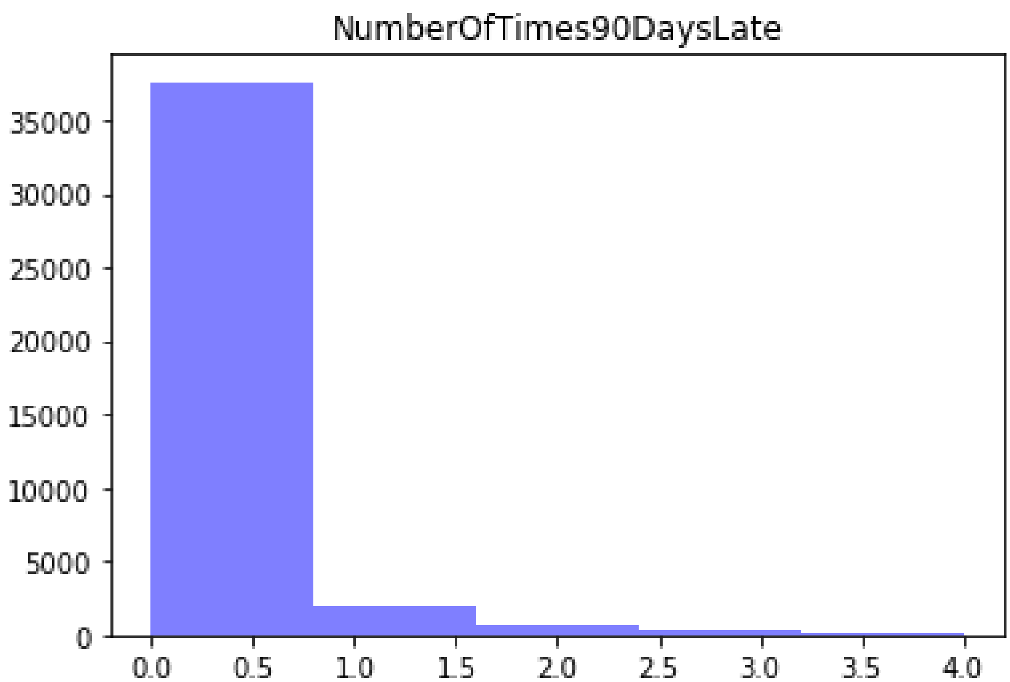

zipcode
20000
17500
15000
12500
10000
7500
5000
2500
0
60600
60610
60620
60630
60640

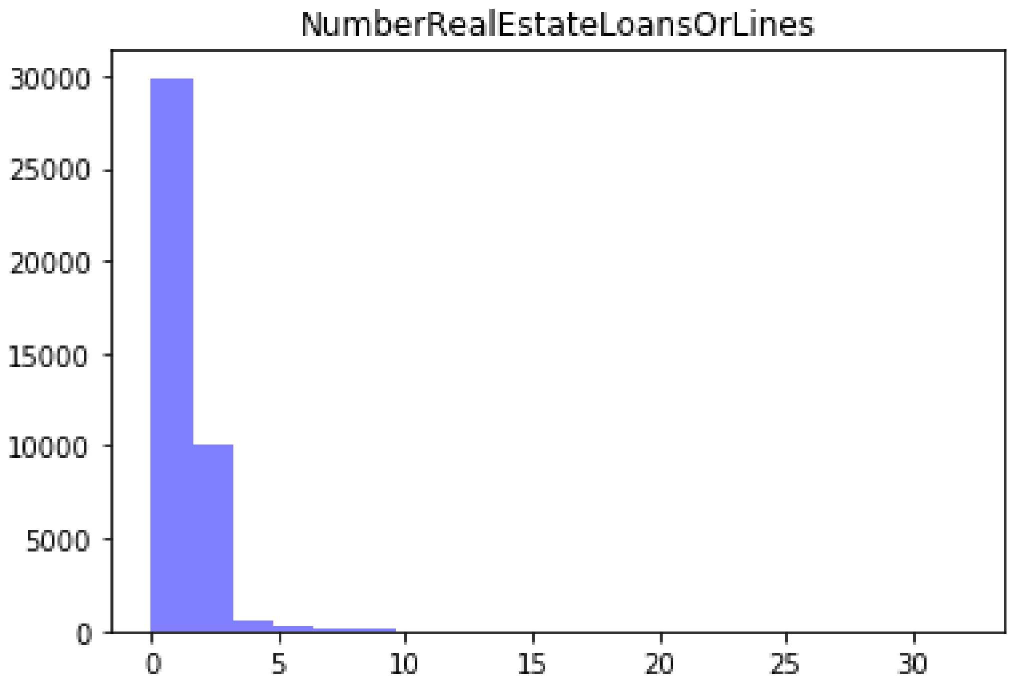

NumberOfTime30-59DaysPastDueNotWorse
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

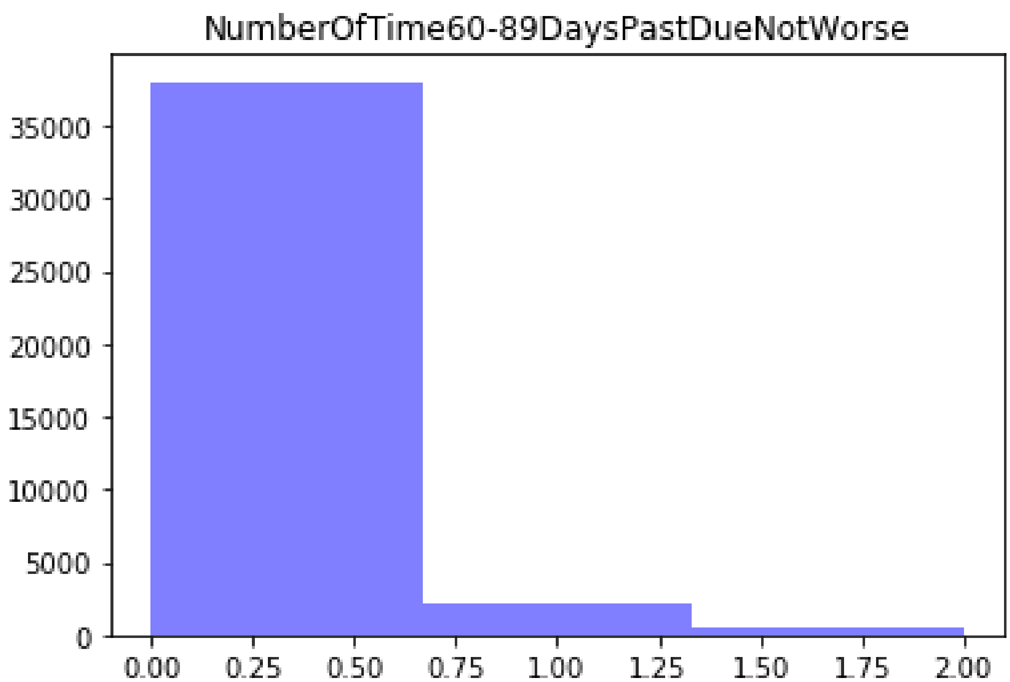

DebtRatio
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000

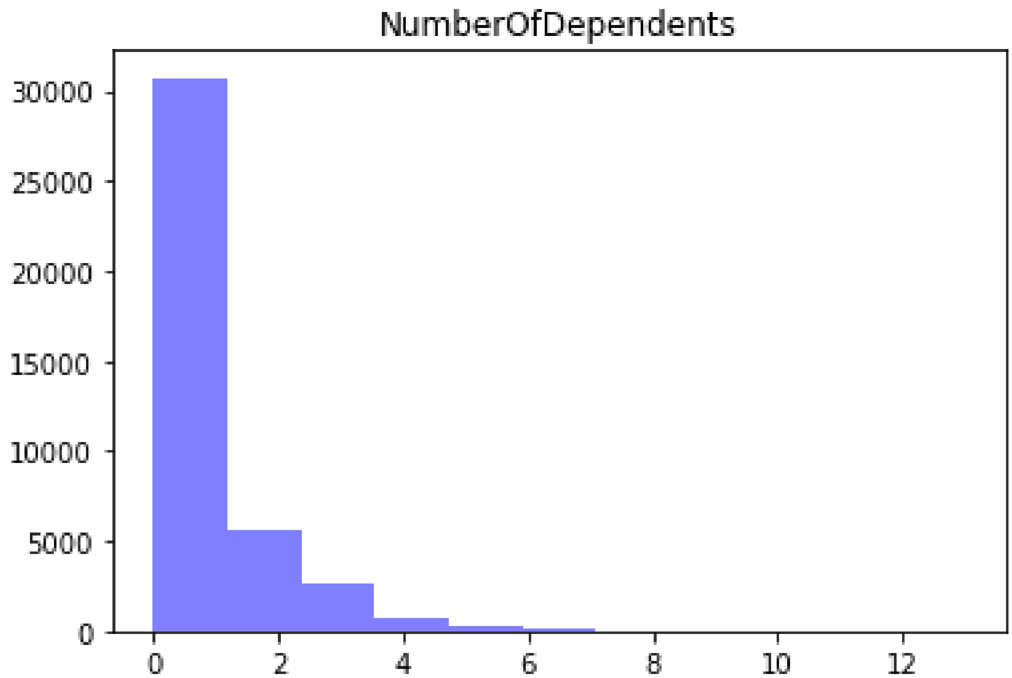

MonthlyIncome
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000


NumberOfOpenCreditLinesAndLoans
10000
8000
6000
4000
2000
0
0
10
20
30
40
50


NumberOfTimes90DaysLate
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


NumberRealEstateLoansOrLines
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30


NumberOfTime60-89DaysPastDueNotWorse
35000
30000
25000
20000
15000
10000
5000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



This gave a much tighter histogram for the variables with the outliers.

Function to create a new restricted dataframe. Takes a dictionary of restrictions, like histograms function.

From the next descriptive functions, we will use a dataframe with the columns with outliers restricted to be under the 99 percentile.

```
In [9]: credit_df_restrict = ppln.restrict_df(credit_df, restrict=restrictions)
        ppln.see_summary_stats(credit_df_restrict, ppln.OUTCOME_VAR)
        ppln.see_summary_stats(credit_df, ppln.OUTCOME_VAR)
```

```
count    39213.000000
mean         0.146482
std          0.353593
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: SeriousDlqin2yrs, dtype: float64
count    41016.000000
mean         0.161400
std          0.367904
min          0.000000
25%          0.000000
```

```
50%          0.000000
75%          0.000000
max          1.000000
Name: SeriousDlqin2yrs, dtype: float64
```

After restricting the dataframe, we lost 2803 observations, and the percentage of people who experienced 90 days past due delinquency or worse was reduced from 16% to 14.6%.

```
In [10]: ppln.summary_by_objective(credit_df_restrict)
```

```
Out[10]: SeriousDlqin2yrs                                  0             1    perc diff
         PersonID                              123483.549284  75494.701079   -38.862544
         RevolvingUtilizationOfUnsecuredLines       0.285303      0.667140   133.835685
         age                                       52.844423     45.986943   -12.976734
         zipcode                                60624.060593  60622.580432    -0.002442
         NumberOfTime30-59DaysPastDueNotWorse       0.177657      0.792479   346.072709
         DebtRatio                                266.878205    197.275259   -26.080416
         MonthlyIncome                           6241.114595   5351.716390   -14.250631
         NumberOfOpenCreditLinesAndLoans            8.440706      7.988510    -5.357331
         NumberOfTimes90DaysLate                    0.039171      0.504875  1188.913022
         NumberRealEstateLoansOrLines               0.987839      0.969359    -1.870766
         NumberOfTime60-89DaysPastDueNotWorse       0.033643      0.281163   735.723167
         NumberOfDependents                         0.736099      0.942563    28.048421
```

This function is useful to see how the average or any other statistic of the values variables differ among the different values of the objective variable. Some interesting differences in terms of the population that experienced the severe past due delinquency are:

- RevolvingUtilizationOfUnsecuredLines, is on average 133% higher.
- MonthlyIncome, is 14% lower on average.
- NumberOfTime30-59DaysPastDueNotWorse, is 346% higher on average.
- DebtRatio, is 26% lower on average.
- NumberOfTimes90DaysLate, is 1188% higher on average, from 0.04 times to 0.5 times.
- NumberOfTime60-89DaysPastDueNotWorse, is 735% higher on average, from 0.03 to 0.28 times.

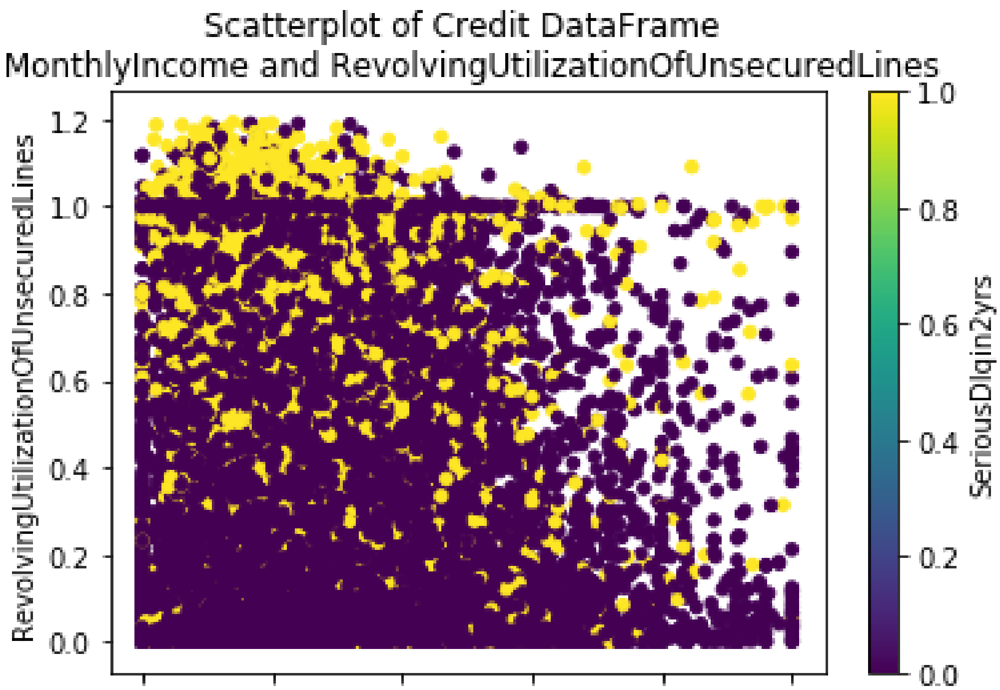
Function to see scatterplot between two variables. Takes:

- xvar
- yvar: Default outcome variable
- colorcol: Col to add color.
- logx: Plot x values in log (Default False)

- logy: Plot y values in log (Default False)
- xjitter: Add randomness to x values to avoid overlapping (Default False)
- yjitter: Add randomness to y values to avoid overlapping (Default False)

```
In [11]: ppln.see_scatterplot(credit_df_restrict, 'MonthlyIncome', 'RevolvingUtilizationOfUnse
```

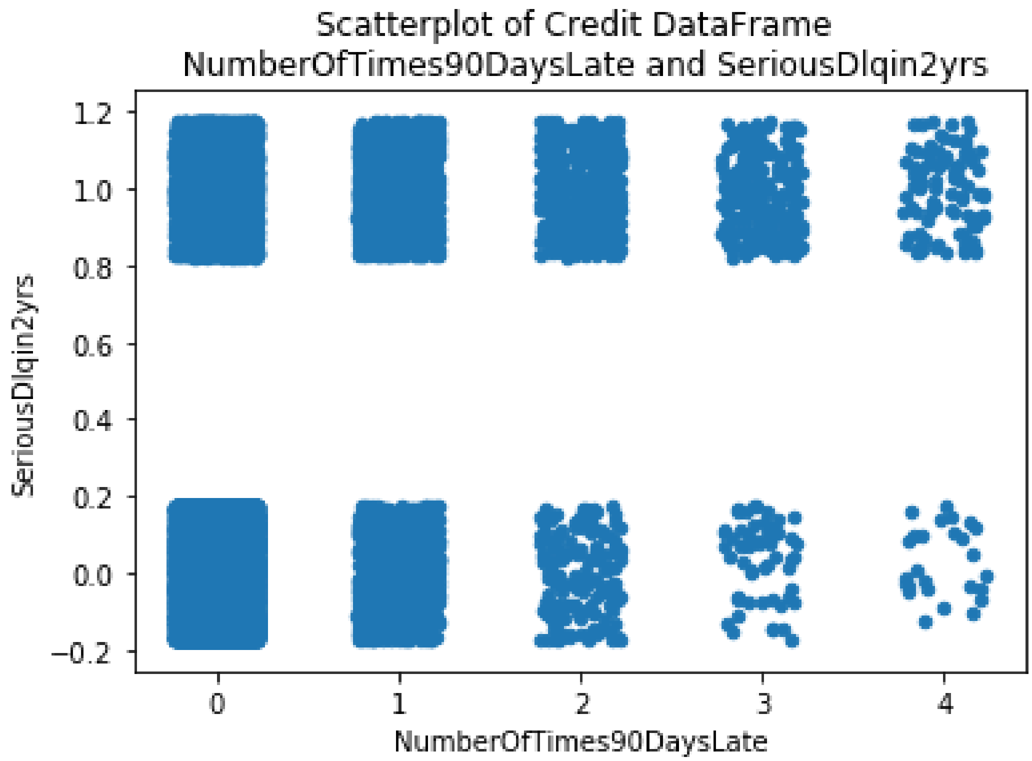
```
<Figure size 432x288 with 0 Axes>
```



This scatterplot shows the relation between Monthly Income and Revolving Utilization of UnsecuredLInes, colored by the Serious Delinquency. There is no apparent relation between income and the utilization of unsecured lines, but there can be seen more occurrences of the outcome variable under high levels of the utilization of unsecured lines, which was also seen in the past summary table.

```
In [12]: ppln.see_scatterplot(credit_df_restrict, 'NumberOfTimes90DaysLate', xjitter=True, yji
```

```
<Figure size 432x288 with 0 Axes>
```



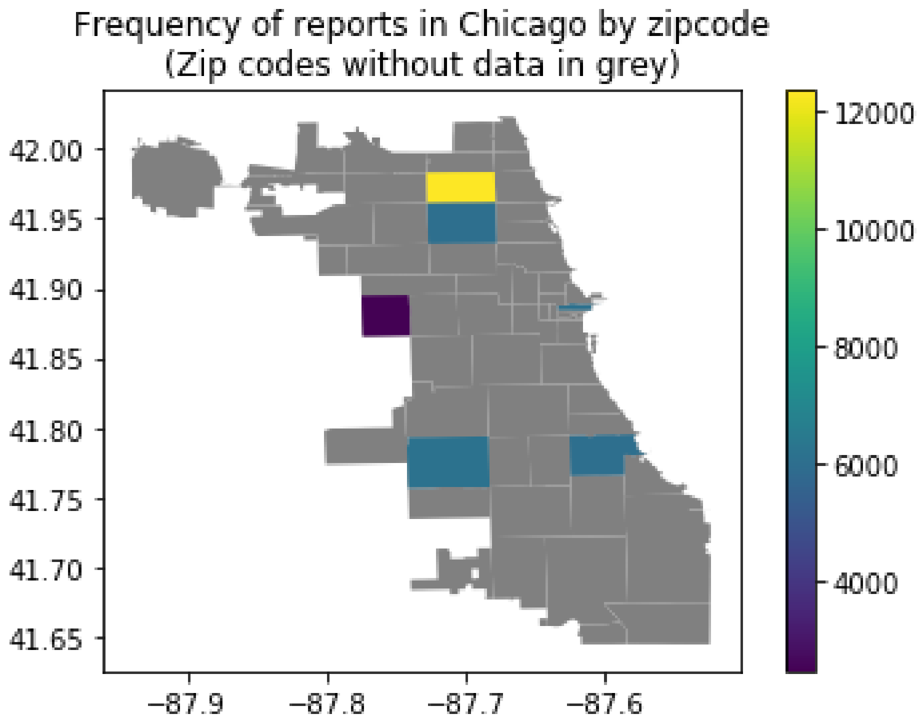
This graph is a scatterplot between the number of times 90 days late and the outcome variable. Since both variables are categorical and have few values, the jitter parameter allows seeing the relation without overlapping. It is clear how there are more occurrences of high levels of Number of Times 90 Days Late on the Serious Delinquency.

Function to map aggregated values by zip code. This function uses the zip boundaries dataframe downloaded from the Chicago Open Data Portal, merges it with the credit data and produces a map based on the aggregation specified.

Takes:

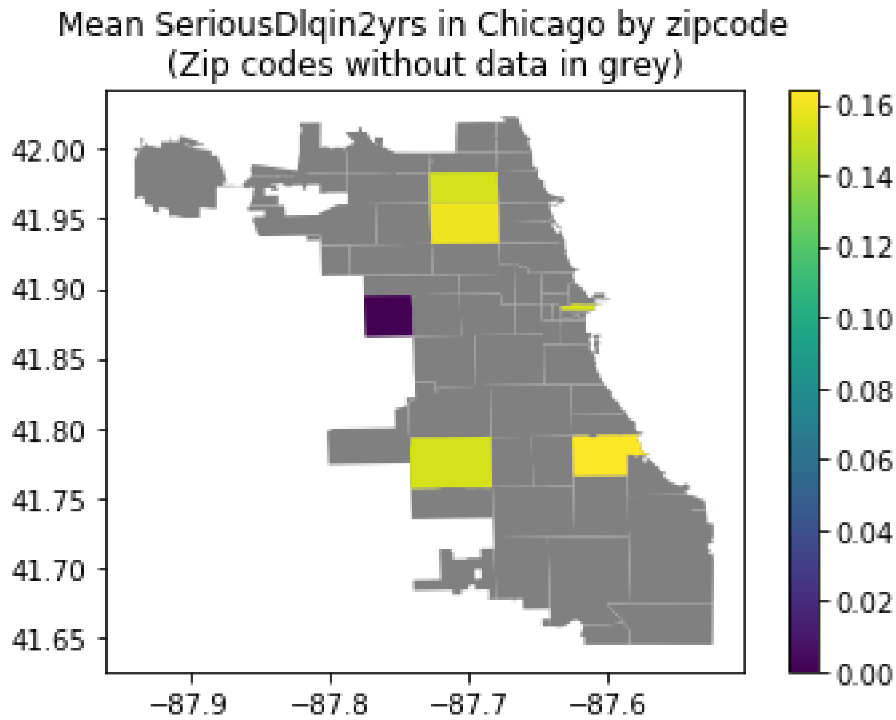
- colorcol: Column to use to color the map. Default: Outcome Var
- funct: Function to aggregate by zipcode. Default: 'mean'
- count: True to color the map by frequency. Default: False

```
In [13]: ppln.map(credit_df_restrict, zip_gdf, count = True)
```



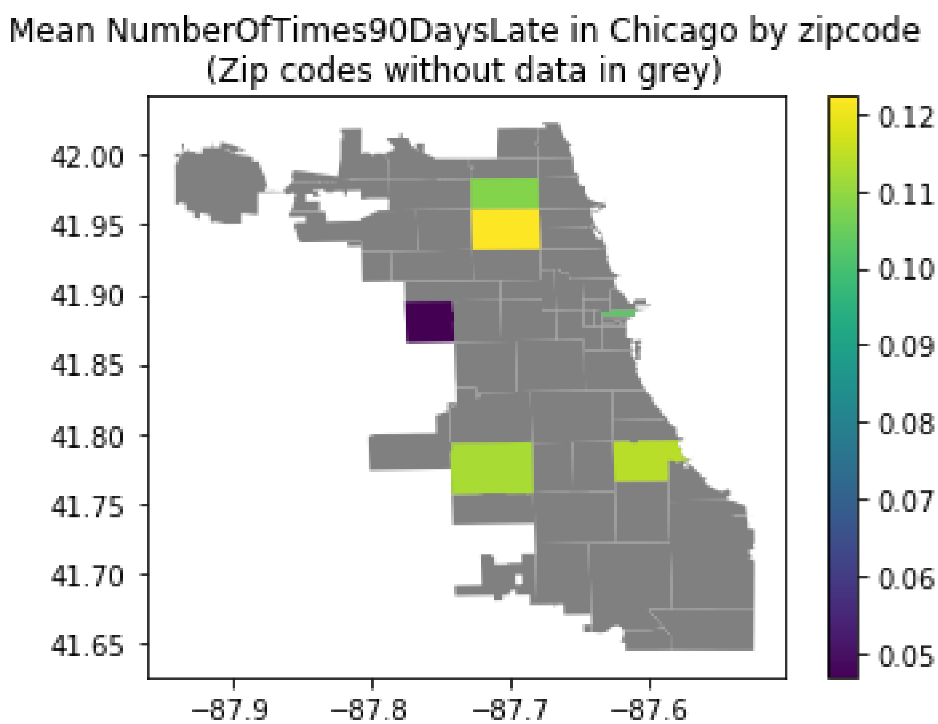
This is a map colored by frequency. As the map shows, we do not have data of all Chicago. We only have data for 6 zip codes in the city. The Zipcode from which we have most of our observations is in the north part of the city.

```
In [14]: ppln.map(credit_df_restrict, zip_gdf)
```

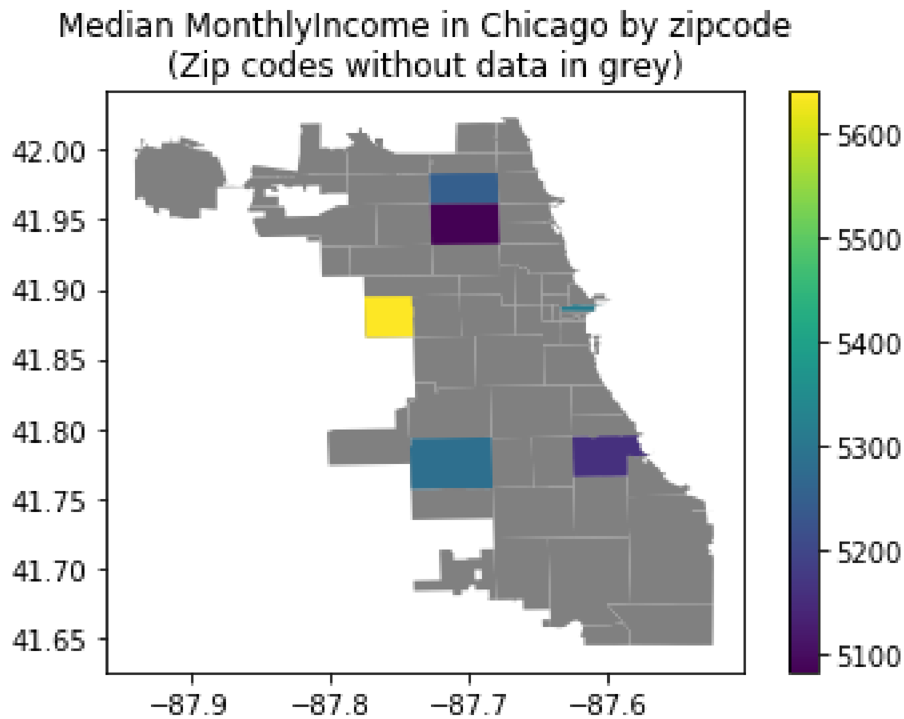


This map is colored by the average value of Serious Delinquency. Since our data is not geographically representative of the city, I would be skeptical of any interpretation of this map.

```
In [15]: ppln.map(credit_df_restrict, zip_gdf, 'NumberOfTimes90DaysLate', 'mean')
```



In [16]: ppln.map(credit_df_restrict, zip_gdf, 'MonthlyIncome', 'median')



These are more examples of using the map function, with different columns and aggregation function.

1.3 Pre-Process Data

Function fill NaN values. Takes a list of columns and function. Default all columns 'mean'.

In [17]: credit_df_restrict.isna().sum()

```
Out[17]: PersonID                                   0
         SeriousDlqin2yrs                           0
         RevolvingUtilizationOfUnsecuredLines       0
         age                                        0
         zipcode                                    0
         NumberOfTime30-59DaysPastDueNotWorse       0
         DebtRatio                                  0
         MonthlyIncome                           7344
         NumberOfOpenCreditLinesAndLoans            0
         NumberOfTimes90DaysLate                    0
         NumberRealEstateLoansOrLines               0
```

```
NumberOfTime60-89DaysPastDueNotWorse       0
NumberOfDependents                       984
dtype: int64
```

Monthly Income and Number of Dependents have 7344 and 984 NaN values respectively

```
In [18]: credit_df_restrict = ppln.fillna(credit_df_restrict)
         credit_df_restrict.isna().sum()
```

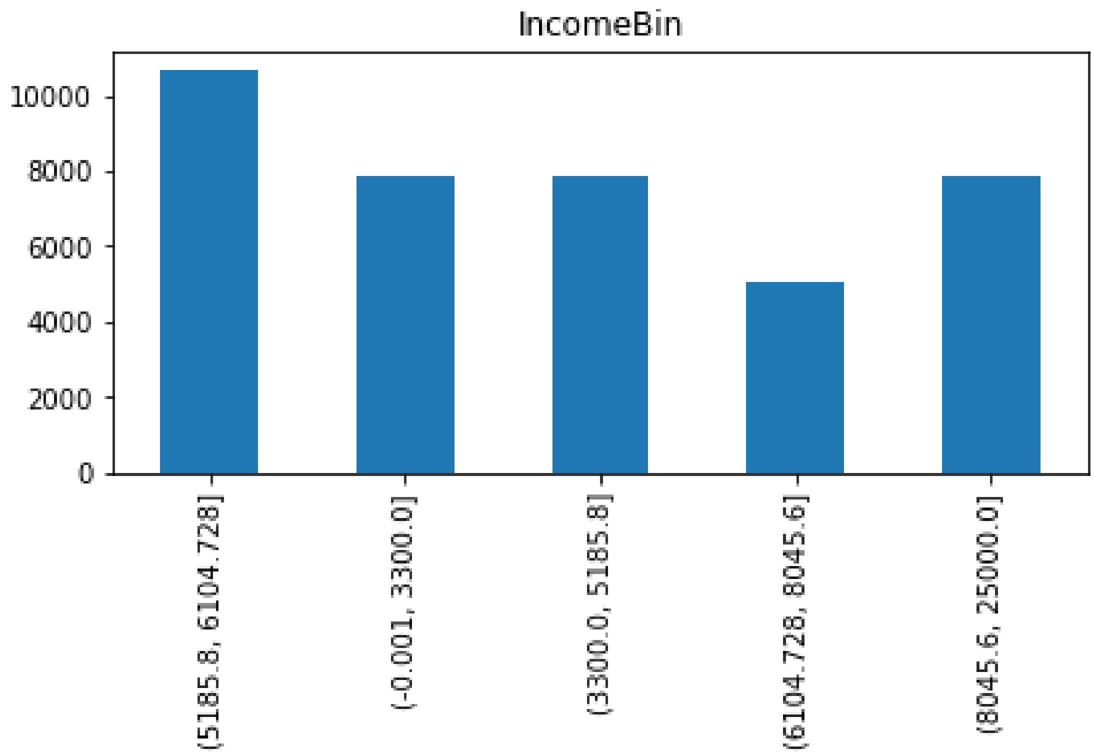
```
Out[18]: PersonID                                0
         SeriousDlqin2yrs                        0
         RevolvingUtilizationOfUnsecuredLines    0
         age                                     0
         zipcode                                 0
         NumberOfTime30-59DaysPastDueNotWorse    0
         DebtRatio                               0
         MonthlyIncome                           0
         NumberOfOpenCreditLinesAndLoans         0
         NumberOfTimes90DaysLate                 0
         NumberRealEstateLoansOrLines            0
         NumberOfTime60-89DaysPastDueNotWorse    0
         NumberOfDependents                      0
         dtype: int64
```

NaN values wele filled with the mean value of each column.

1.4 Generate Features/Predictors

Function discretize continuous variable Takes a pandas series, the number of bins and a boolean True if bins should be made equal length, otherwise are made so they contain the same number of observations, Default is False. Note: Duplicates may cause that groups are not equally sized.

```
In [19]: credit_df_restrict['IncomeBin'] = ppln.discretize(credit_df_restrict['MonthlyIncome']
         ppln.see_histograms(credit_df_restrict, ['IncomeBin'])
```

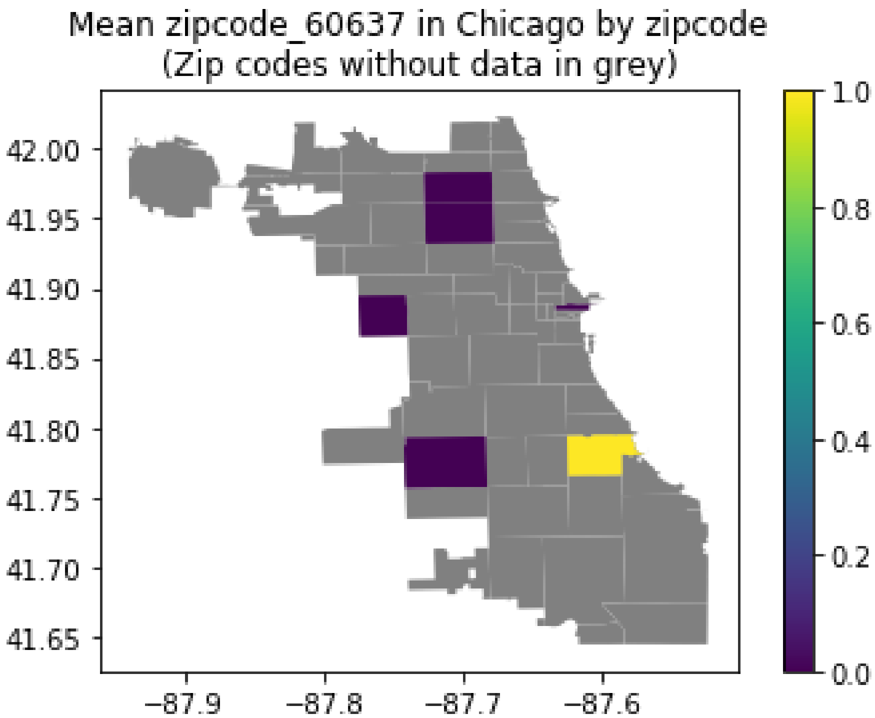


We discretized the income variable into 5 equal sized groups. In this case, the duplicates caused unequal sized bins. This was expected because we used the mean value to replace duplicate values, which is why the middle bin has more observations than the others.

Function create dummy variables from categorical Takes a pandas series, the number of bins and a boolean True if bins should be made equal length, otherwise are made so they contain the same number of observations, Default is False. Note: Duplicates may cause that groups are not equally sized.

```
In [20]: zip_dummies = ppln.make_dummies_from_categorical(credit_df_restrict['zipcode'])
         print("Dummies are:", list(zip_dummies.columns))
         credit_df_restrict = credit_df_restrict.join(zip_dummies)
         ppln.map(credit_df_restrict, zip_gdf,'zipcode_60637')
```

```
Dummies are: ['zipcode_60601', 'zipcode_60618', 'zipcode_60625', 'zipcode_60629', 'zipcode_606
```



We created dummy variables for the zipcode. As the map demonstrates, the resulting column for 'zipcode_60637' has value of 1 only for observations in Hyde Park.

1.5 Build ML classifier

Converting categorized income bins to dummies.

```
In [21]: income_dummies = ppln.make_dummies_from_categorical(credit_df_restrict['IncomeBin'])
         credit_df_restrict = credit_df_restrict.join(income_dummies)
```

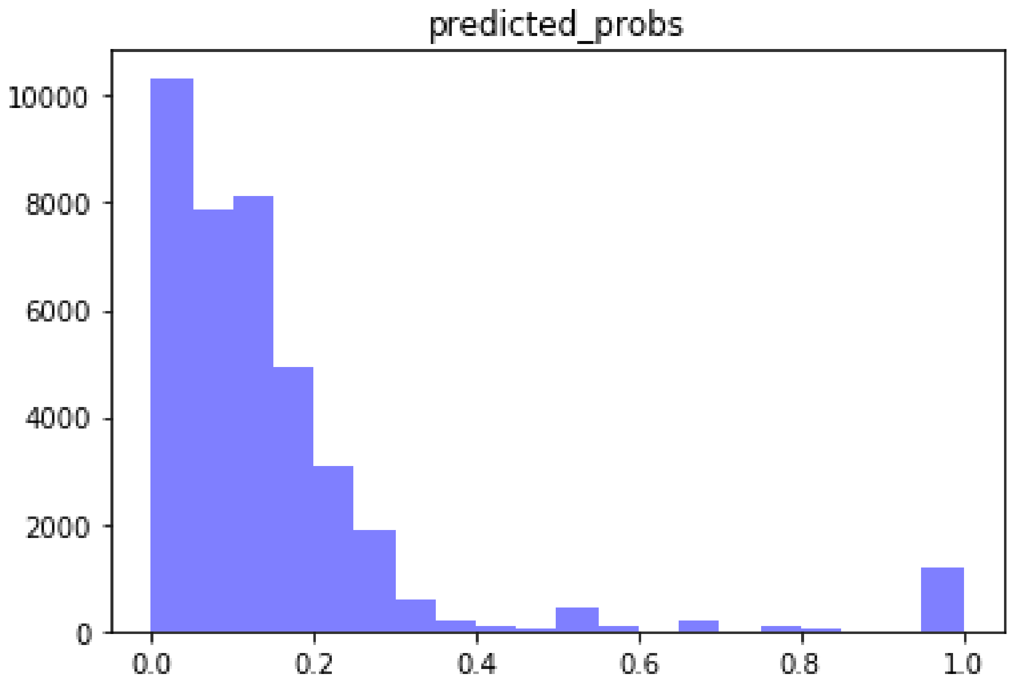
Function to create tree classifier. Takes a train dataframe, the features to use and the outcome variable. The default outcome variable is the outcome variable indicated in pipeline.py

```
In [22]: features = ['NumberOfTimes90DaysLate', 'age'] + list(zip_dummies.columns) + list(incom
         dec_tree = ppln.build_tree_classifier(credit_df_restrict, features)
```

Function to obtain prediction Takes a tree, a test dataframe and the list.

```
In [23]: credit_df_restrict['predicted_probs'] = ppln.obtain_predicted_probabilities(dec_tree,

In [24]: ppln.see_histograms(credit_df_restrict, ['predicted_probs'])
```

```
<Figure size 432x288 with 0 Axes>
```



Most of the predicted probabilities are below 0.2, which is consistent with the dataframe that has only 14% of success.

Function to obtain accuracy Takes a series of real y values, series of predicted probabilities and threshold. Default is 0.5

```
In [25]: ppln.compute_accuracy(credit_df_restrict[ppln.OUTCOME_VAR], credit_df_restrict['predic
```

```
Out[25]: correct      Correct   Incorrect
         yvar_real
         0           0.968448    0.031552
         1           0.105153    0.894847
         All         0.841991    0.158009
```

Using the threshold of 0.5, the model correctly predicted 84% of the observations. Of the non-delinquency, correctly predicted 97% and of the delinquency correctly predicted only 28%.

```
In [26]: ppln.compute_accuracy(credit_df_restrict[ppln.OUTCOME_VAR], credit_df_restrict['predic
```

```
Out[26]: correct      Correct   Incorrect
         yvar_real
         0           0.936897    0.063103
         1           0.154248    0.845752
         All         0.822253    0.177747
```

Reducing the threshold to 0.3 did increase the accuracy of the model for delinquency, but decreased overall.

Now we include income as float and NumberOfTime60-89DaysPastDueNotWorse.

```
In [27]: features_1 = ['NumberOfTimes90DaysLate', 'NumberOfTime60-89DaysPastDueNotWorse', 'age
         dec_tree1 = ppln.build_tree_classifier(credit_df_restrict, features_1)
         predicted_probs_1 = ppln.obtain_predicted_probabilities(dec_tree1, credit_df_restrict
         ppln.compute_accuracy(credit_df_restrict[ppln.OUTCOME_VAR], predicted_probs_1)
```

```
Out[27]: correct     Correct  Incorrect
         yvar_real
         0          0.998894   0.001106
         1          0.810063   0.189937
         All        0.971234   0.028766
```

The accuracy of the model increased to 97% overall, 99% for non-delinquency and 81% for delinquency, with a threshold of 0.5.