Homework 2 - Creating a Pipeline

Allison Collins

I have provided docstrings in the pipeline.py file for the different functions created as per the assignment for different stages of the pipeline; here I implement them on the credit dataset and interpret results.

In [1]:
```
#Load our pipeline file and pydot to use in visualizing the tree
import pipeline
import pydot
```

First, let's look at summary statistics. In the pipeline file, I have generated functions which produce a table with mean, std deviation, median, and max and min values across all columns; we will use the feature to remove certain columns to remove person ID, zipcodes, since those are not meaningful variables to check distribution on.

From the below chart, we can see that mean income is somewhat higher than median, indicating skewed upwards (which we can see there is a very high max value, which could indicate an extremely high earner or someone who entered in their annual income for monthly). A few other observations -- revolving utilization has a high standard deviation relative to the mean; same with number of days past due.

In [2]:
```
pipeline.calc_summary_stats('credit-data.csv',['PersonID','zipcode'])
```

Out[2]:

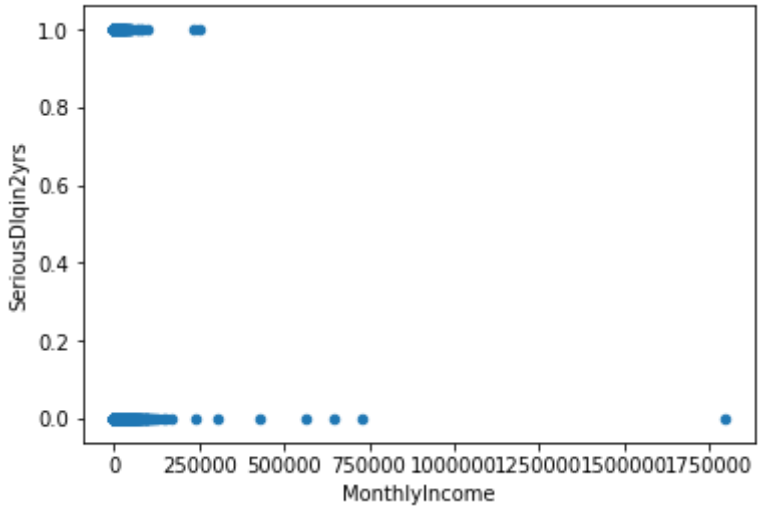
	mean	std_dev	median	max_val	min_val
SeriousDlqin2yrs	0.161400	0.367904	0.000000	1.0	0.0
RevolvingUtilizationOfUnsecuredLines	6.375870	221.618950	0.189730	22000.0	0.0
age	51.683489	14.746880	51.000000	109.0	21.0
NumberOfTime30-59DaysPastDueNotWorse	0.589233	5.205628	0.000000	98.0	0.0
DebtRatio	331.458137	1296.109695	0.369736	106885.0	0.0
MonthlyIncome	6578.995733	13446.825930	5250.000000	1794060.0	0.0
NumberOfOpenCreditLinesAndLoans	8.403477	5.207324	8.000000	56.0	0.0
NumberOfTimes90DaysLate	0.419592	5.190382	0.000000	98.0	0.0
NumberRealEstateLoansOrLines	1.008801	1.153826	1.000000	32.0	0.0
NumberOfTime60-89DaysPastDueNotWorse	0.371587	5.169641	0.000000	98.0	0.0
NumberOfDependents	0.773231	1.121269	0.000000	13.0	0.0

I also created a function that produces a simple scatterplot of a given independent and dependent variable, so we can take a glance at correlation. Let's look at whether income appears correlated, and then whether number of times 90 days late.

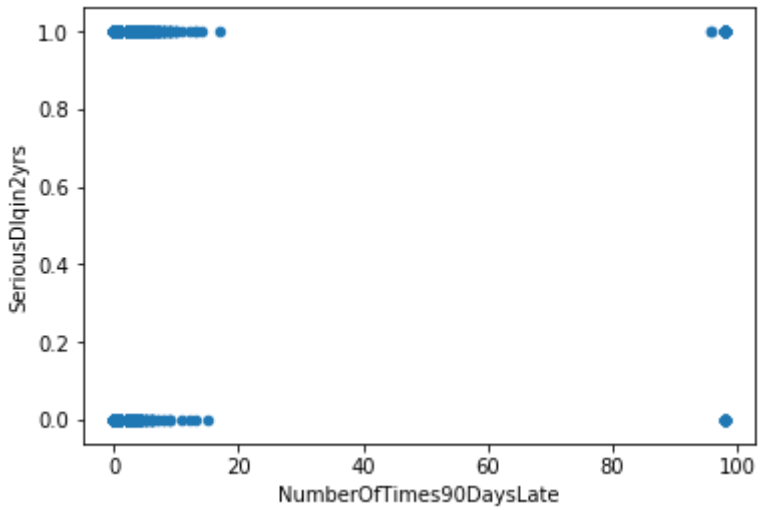
For income, we can see there is more spread in monthly income among those who did not have serious delinquency over the two years. We can also identify a potential outlier, which could be the maximum value above.

For number of times 90 days late, we can see that there is a thicker band of dots continuing farther to the right for those who exhibited serious delinquency (e.g. were late more times) than for those who did not, which intuitively makes sense.

In [3]:
```
pipeline.generate_summary_plot('credit-data.csv','SeriousDlqin2yrs','Mon
thlyIncome')
```



In [4]:
```
pipeline.generate_summary_plot('credit-data.csv','SeriousDlqin2yrs','Num
berOfTimes90DaysLate')
```



Let's now do some data pre-processing before we go to creating the decision tree. First, let's use the function written to fill in any missing values with the median, and get a pandas dataframe back.

In [5]:
```
filled_data = pipeline.fill_missing('credit-data.csv')
```

In [6]:
```
filled_data.head(1)
```

Out[6]:

	PersonID	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	zipcode	Numbe 59DaysPastDu
0	98976	0	1.0	55	60601	

Now, let's use the function in pipeline to create a binned column where we break age up into 4 groups: 20-30 (young adult), 30-45 (adult), 45-65 (middle aged), 65-109 (elderly). The binning function will return ints, since the decision tree requires numbers.

In [7]:
```
filled_data = pipeline.create_binned_col(filled_data,'age', [20,30,45,65
,109])
```

Now, let's use the function in pipeline to make number of dependents into a binary variable - whether or not one has a dependent at all.

In [8]:
```
#Check the unique values in the column to create our mapping dictionary
filled_data.NumberOfDependents.unique()
```

Out[8]:
```
array([ 0.,  2.,  1.,  4.,  3.,  8.,  5.,  6.,  7.,  9., 13.])
```

In [9]:
```
#Create the dictionary to map to binary 0,1
dependents_dict = {0.0: 0, 2.0: 1, 1.0: 1, 3.0: 1, 4.0: 1, 5.0: 1, 6.0:
1, 7.0: 1, 8.0: 1, 9.0: 1, 13.0: 1}
```

In [10]:
```
filled_data = pipeline.create_binary_col(filled_data, 'NumberOfDependent
s', dependents_dict)
filled_data.head(1)
```

Out[10]:

	PersonID	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	zipcode	Numbe 59DaysPastDu
0	98976	0	1.0	55	60601	

Now, we can create the decision tree. We will allow all potential predictors to be used, except Person ID (since it is meaningless, as it as an ID number). We will use graphviz to visualize the tree below.

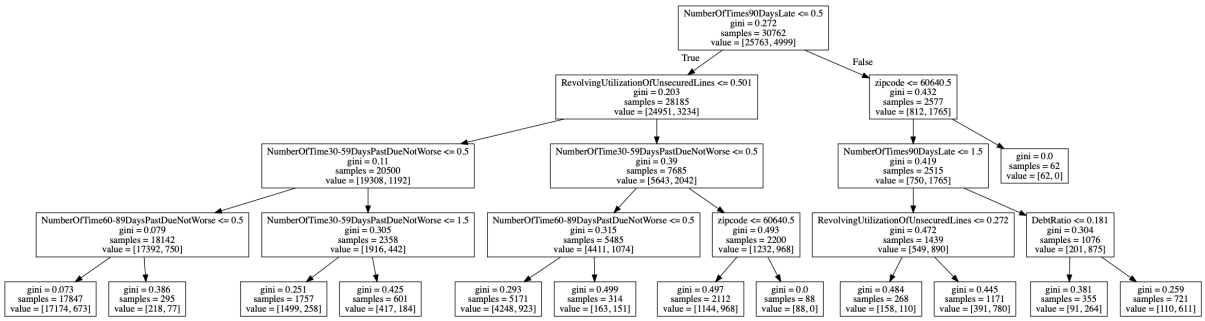
In [11]:
```
y_test, y_predict = pipeline.create_decision_tree(filled_data, 'SeriousDlqin2yrs', 4, 100,['RevolvingUtilizationOfUnsecuredLines',
       'age_binned', 'zipcode', 'NumberOfTime30-59DaysPastDueNotWorse',
'DebtRatio',
       'MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans',
       'NumberOfTimes90DaysLate', 'NumberRealEstateLoansOrLines',
       'NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfDependents_binary'])
```

In [12]:
```
(graph,) = pydot.graph_from_dot_file('tree.dot')
```

In [13]:
```
graph.write_png('tree.png')
```

In [14]:
```
from IPython.display import Image
Image("tree.png")
```

Out[14]:



We can see that the tree first split on number of times 90 days late. For those for whom this was true that it was <.5, revolving utilization was the next most significant predictor, followed by the number of times 30-59 days past due. For simplicity I implemented a max depth of 4, but this is a parameter which can be changed (as is the minimum number of entries required to make a split).

For those for whom they were later >.5 time, the next most important criteria was different -- this time it was zip code. We can trace down the paths to classify individuals.

Now we can evaluate the file using the accuracy feature of sklearn, which is wrapped into a function in the pipeline file. Per below, we can see that the model has ~87% accuracy, although as we have discussed in class, this may not be the best measure.

In [15]:
```
accuracy = pipeline.evaluate_decision_tree(y_test, y_predict)
print(round(accuracy*100,2),'%')
```

```
87.3 %
```