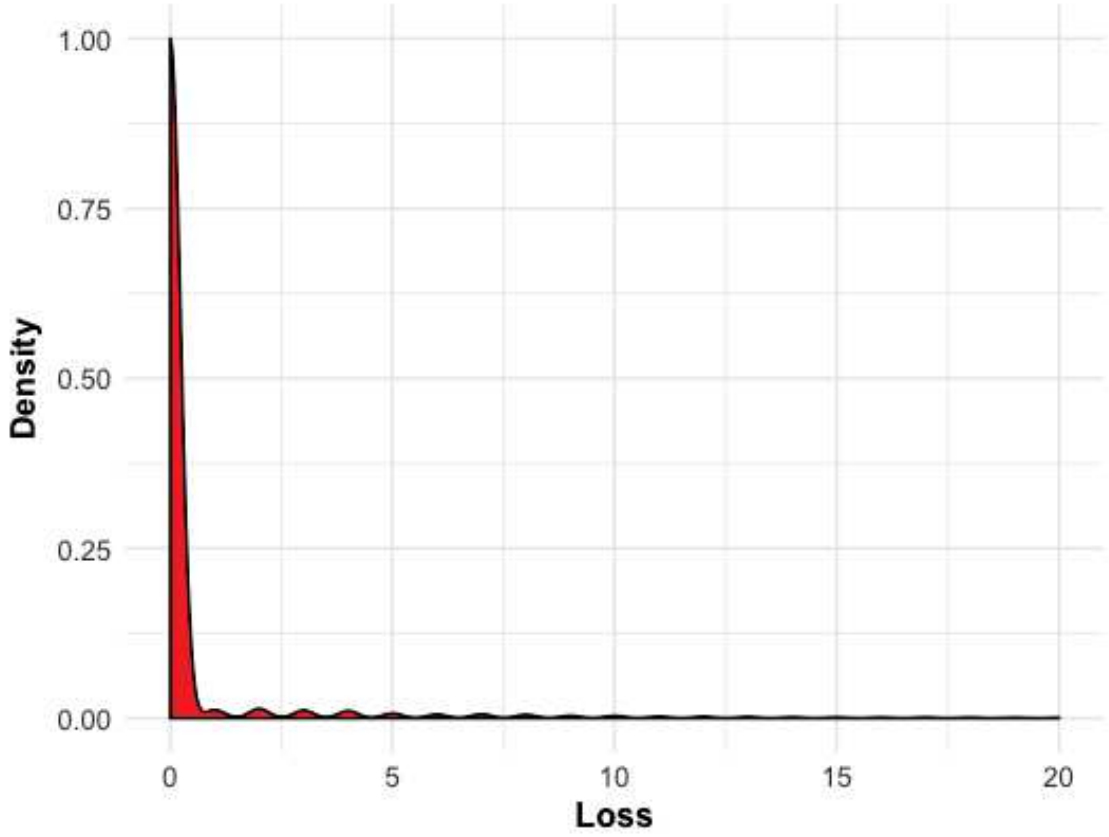


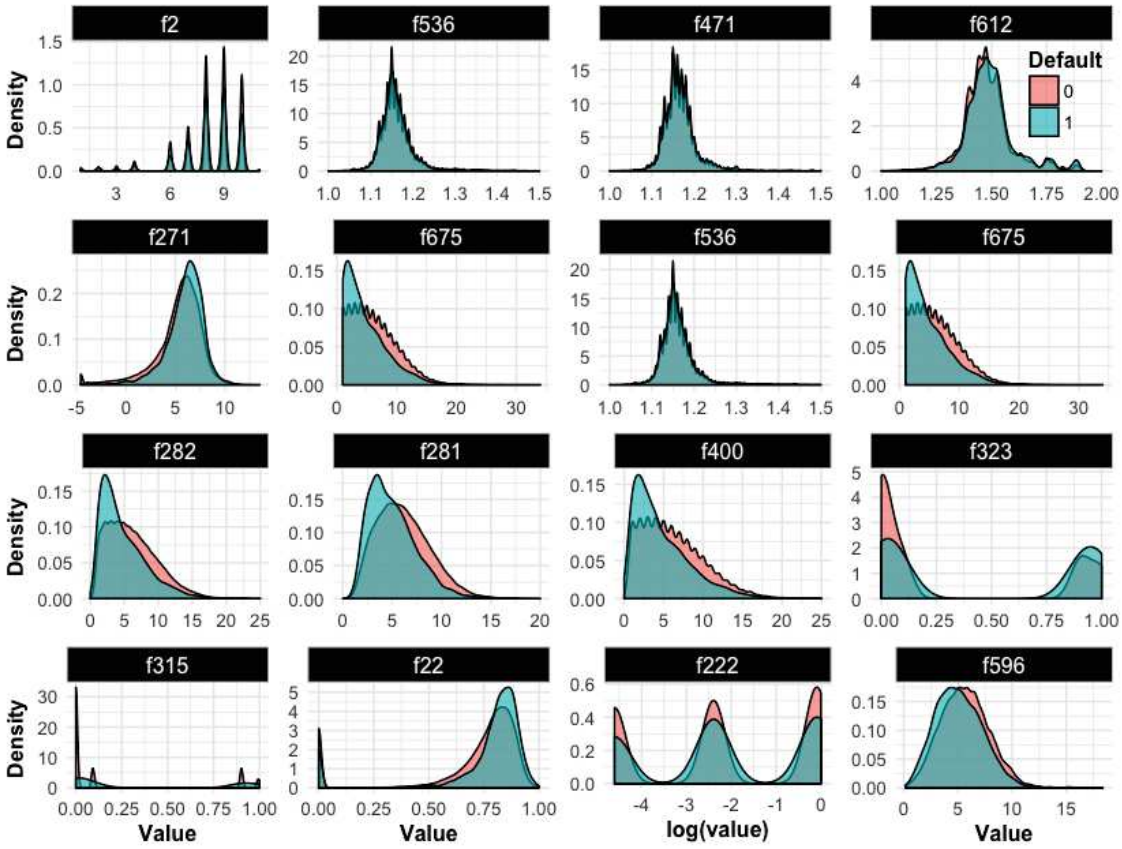

## 1.2 Feature Selection

By far the most challenging part of this project for us was feature selection. Because there were over 700 features we had to sift through them to find the ones that were best for our model. Initially we did tried to use all 770 features in our model however we quickly realized this was not feasible because all of our models predicted 0 (no one would default).

Our first round of feature selection was done in R. We preprocessed the loss to be a boolean variable. 0 for paying back your loan, and 1 for not paying back in some amount. We then made a series of density graphs of various features. We would look at these density graphs to find features with a small amount of overlap. This would indicate that the features could be useful in classification because there non overlapping values could be used to distinguish between the loss.

```
In [8]: from IPython.display import Image
        Image(filename='Rplot03.png')
```

```
Out[8]:
```




Other feautres we used in our model came from the correlation matrix mentioned above. We sorted the matr ix by features that correlated with loss and used those features that were most highly correlated.

The last set of features and arguably most important came from pairs of highly correlated features. From looking at Piazza, we realized that highly correlated feature pairs could be important because they could correspond to metrics such as loan payments over time, or other time dependent features.

We took these pairs of features and made our own new processed features from them. We tried subtraction, division, and multiplication of pairs in our models. Via brute force we settled on subtraction of these pairs to be the most significant in our model.

In the end the features in our model came from the ones selected above. We had discovered many features that could be included so we had to brute force the selectioon. We actually wrote one script in R that would run our models continuously with different features and record added features which gave the most improvement.

As a side note we also tried Principle Component Analysis (PCA) to reduce our features to just 10. However, this didn't work particularly well in our models. In addition running PCA for a large number of features such as 20 or 50 was computationaly expensive.

## 1.3 Splitting up the data

We decided to split the training data into 80% for training and 20% for the test. We used a random split of the data, but with a set seed so we could reproduce the result during development.

## 1.4 Pipeline

Like mentioned above we initially tried to use just simple regression models. However, because we only predicted people would pay back there loans we decided a two-step model approach was necessary. Based

on the histogram with most people paying back there loans we decided we should first use a classifier that would predict whether someone would pay back their loan or not. Then we could use a regression model trained on people who default after that would predict the loss.

### 1.5 Classifications Tried

We tried many different classifications. The problem we repeatedly ran into was always predicting a variable would pay back there loan. At first, this was deceptive because we thought we had a very accuracy (90%) but that was only because the majority of people pay back their loan. To better evaluate our accuracy, we separated the accuracy of predicting not default and default in order to see how well our model truly performed. Not surprisingly, for bad models, we would have high 0 (nondefault accuracy) and very low 1 (default accuracy).

The first model we tried was the logistic regression, however, this did not classify ones very well at all. Our big breakthrough came from the `sk.ensemble` library. The models in this library, particularly the decision tree models, proved to be very effective. We think this is because these models are much more expressive than logistic regression and are therefore able to classify better on our data. After trying a few of these models such as Gradient Boosted Tree, SVM, etc. we found Random Forest was the best. It particularly performed well when we used many trees (200) but it still generalized very well to our out of sample error. With Random Forest we achieved around a 98% accuracy, with a 1 accuracy of 97%, and a 0 accuracy of 98%.

### 1.6 Regressions Tried

Just like classification, we tried many regression models. We started with the simple models such as linear regression. However, more powerful models seemed significantly more effective. We again used the ensemble package from sklearn that has RandomForest regression and GradientBoosingRegressor as different types of regression models. These regressions worked more accurate compared to the simple linear regression.

For features in our regression model, we roughly used the same as our classification. Adding slightly more from trial and error and our running R script that picked out useful features.

### 1.7 Contributions

- Hooman - Wrote the report, worked on regression models
- Yuan - Helped to write the report, worked on regression models, data cleaning
- Ashkan - Did the R code! Worked on various models and report
- Sasha - Worked on the python code, and the models for the report as well