Loan Prediction

Machine Learning Engineer Nanodegree

Capstone Project

Elvin Rey Magsino
September 5, 2017

Definition

Project Overview

The insurance domain is one of the largest consumers of data science analytics. This industry is founded on forecasting and estimating the value/impact of future events. Insurance has been using established predictive modeling practices for some time now, with predictive modeling being used especially in claims loss prediction and pricing. The opportunity to use machine learning techniques, even across new insurance operations, has never been greater as new data sources and big data are more readily available. Machine learning allows insurance companies to yield higher predictive accuracy since it can fit data using more flexible and complex models. ML takes advantage of heavy data analytics and is capable of computing datasets that are seemingly unrelated in a structured, semi-structured, or non-structured environment. Insurers can use ML to better understand the core values of insurance such as risk, claims, and customer experience. Critical areas where machine learning can greatly benefit insurance includes estimating losses, monitoring and detecting fraud, claims processing, and pricing risk.

Gary Reader, KPMG's Global Head of Insurance, one stated,

> *For the insurance sector, we see machine learning as a fundamental game-changer since most insurance companies today focus on three main objectives: improving compliance, improving cost structures and improving competitiveness. Machine learning can form at least part of the answer to all three.*[1]

In a business sense, this is why ML drives so much value:

- *Machine learning delivers more accurate predictions than traditional analysis or human judgment.*
- *Modern techniques make these predictions easy to understand and transparent.*
- *With better predictions, managers make smarter decisions.*
- *Smarter decisions produce more revenue, lower costs, and a better bottom line.*[1]

For this project, we'll be analyzing Dream Housing Finance (DHF), an insurance company which deals specifically with home loans. To calculate loan eligibility, Dream Housing Finance asks potential customers a few questions ranging from income level to loan amount. This insurance company has a presence across all types of property areas – urban, semi urban, and rural. Machine Learning techniques will be implemented to determine the customer's loan eligibility.

Project Statement

When a customer provides their personal details (including gender, marital status, education, number of dependents, income, education level, loan amount, loan term, and credit history), DHF will automatically generate the customer's loan eligibility. Since the output of the algorithm is either a yes or no, this poses a binary classification problem. DHF will need to implement machine learning algorithms to accurately predict whether a certain customer will be approved for a home loan.

Using various known Machine Learning models, we will determine which algorithm is best to use to determine whether a customer will be approved for a home loan. Compared to the learning models that we implement, the desired model will grant us relatively high scores for accuracy, F1-score, and Cross-Validation. Once we find a model that achieves satisfactory scores on accuracy, F1-score, and Cross Validation, we can conclude that the model is the most suitable model for our problem.

Metrics

We'll be looking at the accuracy of the target Boolean variable, Loan_Status, to quantify the performance of both the benchmark model and the solution model. Since the distribution of the target variable is unbalanced (of the 615 requests, 422 were granted for a total of 68.73% success), we would need a strategy to deal with this unbalanced data. The evaluation metric will be an accuracy score – the percentage of the loan approval that the algorithm correctly predicts. However, due to unbalanced data, we're going to look at different performance measures that can give more insight to the accuracy of the model than traditional classification accuracy – this includes precision, recall, and the F1-Score (a weighted average of precision and recall).

We'll be interested in the F1 score as the measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results, and r is the number of correct positive results divided by the number of positive results that should have been returned. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.[4]

(The equation used for calculating the F1 score)

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

We'll also be calculating the cross-validation score. Cross-validation score is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. Since our model is trying to predict whether new customers will be approved for a loan, cross-validation score aims to estimate how accurately a predictive model will perform in practice. Cross-validation combines measures of fit (prediction error) to derive a more accurate estimate of model prediction performance. Simply, a higher cross-validation score means better generalizability and since our model is trying to predict outcomes, cross-validation would be another great metric.

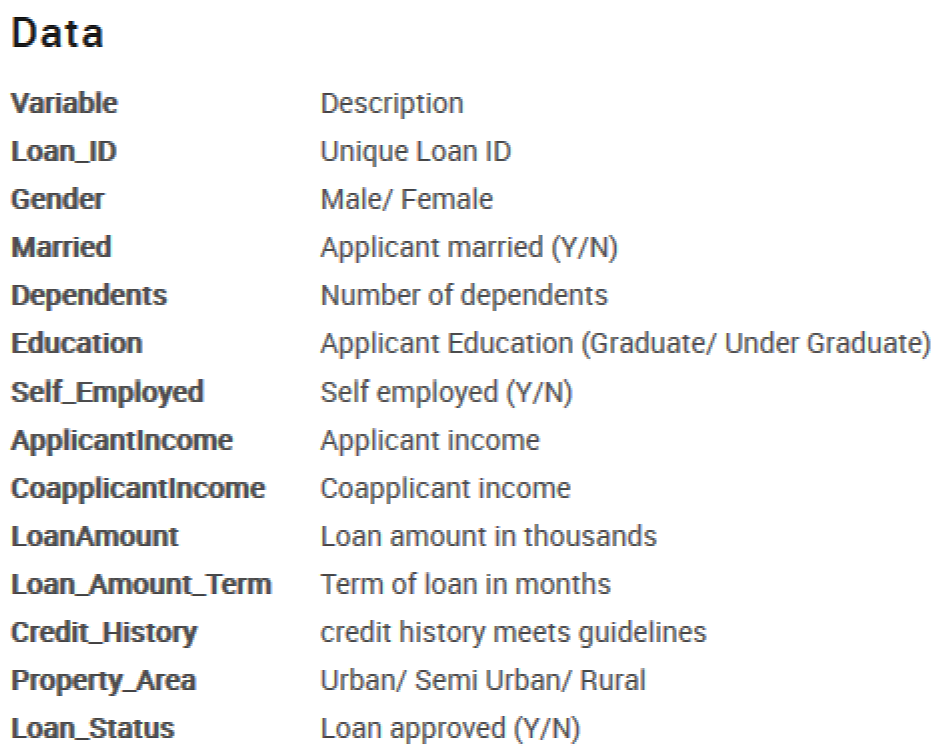
Analysis

Data Exploration

Data was extracted from the loan prediction dataset available through Analytics Vidhya[3]. This project will consider a dataset consisting of 615 rows and 14 categories (description, unique load ID, male/female, applicant married (Y/N), number of dependents, applicant education (graduate/under graduate), self-employed, applicant income, coapplicant income, loan amount in thousands, term of loans in months, credit history meets guidelines, urban/semi urban/rural, loan approved (Y/N). Each row represents one customer whose target variable, loan approved (Y/N), is what the algorithm will be predicting. The target variable can take two values, either yes or no; therefore, this will be a binary classification problem.

Since the distribution of the target variable is unbalanced (of the 615 requests, 422 were granted for a total of 68.73% success), we would need a strategy to deal with this unbalanced data. The evaluation metric will be an accuracy score – the percentage of the loan approval that the algorithm correctly predicts. However, due to unbalanced data, we're going to look at different performance measures that can give more insight to the accuracy of the model than traditional classification accuracy – this includes precision, recall, F-Score (a weighted average of precision and recall), and cross-validation score.

Here are the features of the dataset in a simple table format:

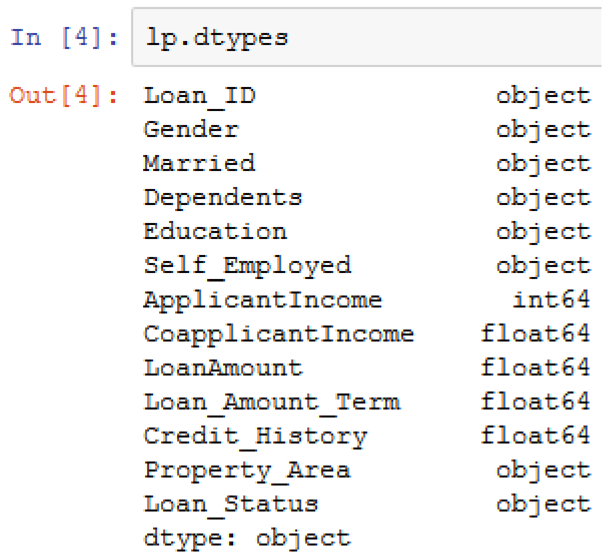


Data

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

After initial analysis of the data, we notice that there are missing and incomplete data. We see that for numerical data, we're missing some data in the following fields: LoanAmount (22 records), Loan_Amount_Term (14 records), and Credit_History (50 records). For categorical data, we're also missing some data in the following fields: Gender (13 records), Dependents (15 records), Self_Employed (32 records). We need some process to fix this missing data.

Here are the data types of the dataset in a simple table format:

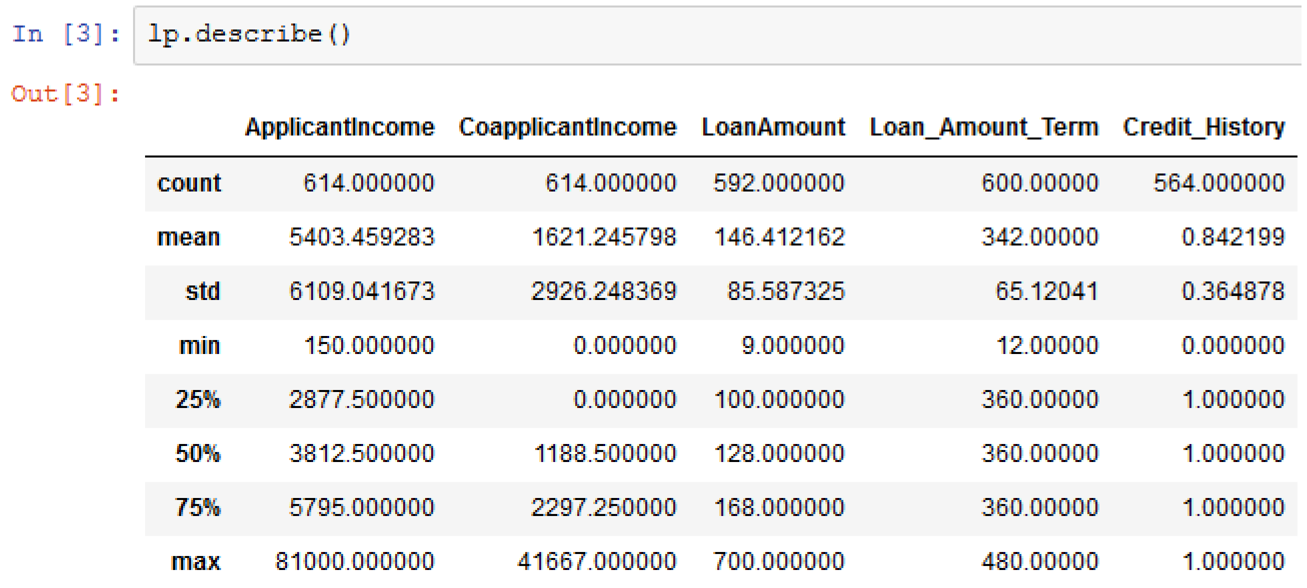


```
In [4]: lp.dtypes

Out[4]: Loan_ID                 object
        Gender                  object
        Married                 object
        Dependents              object
        Education               object
        Self_Employed           object
        ApplicantIncome          int64
        CoapplicantIncome      float64
        LoanAmount             float64
        Loan_Amount_Term       float64
        Credit_History         float64
        Property_Area           object
        Loan_Status             object
        dtype: object
```

We will also need to address outliers in the dataset. We'll discuss these outliers and incomplete/missing data in the following section.

Exploratory Visualization

We first describe the data to see any abnormalities.



```
In [3]: lp.describe()
```

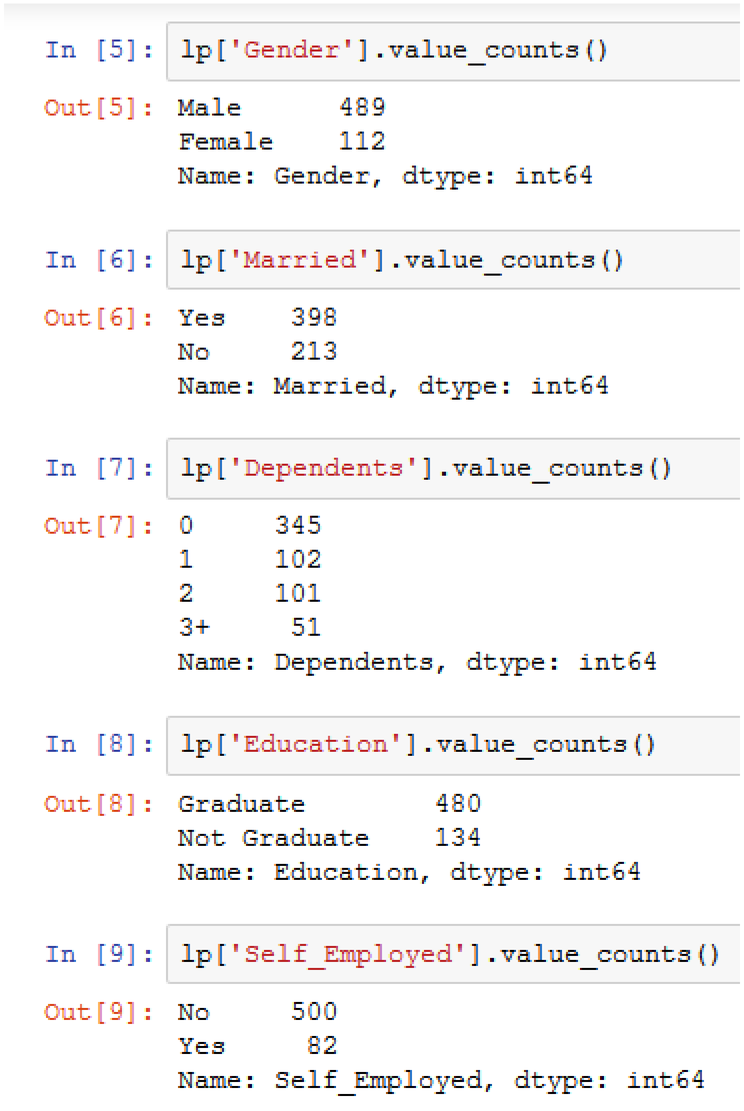
```
Out[3]:
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

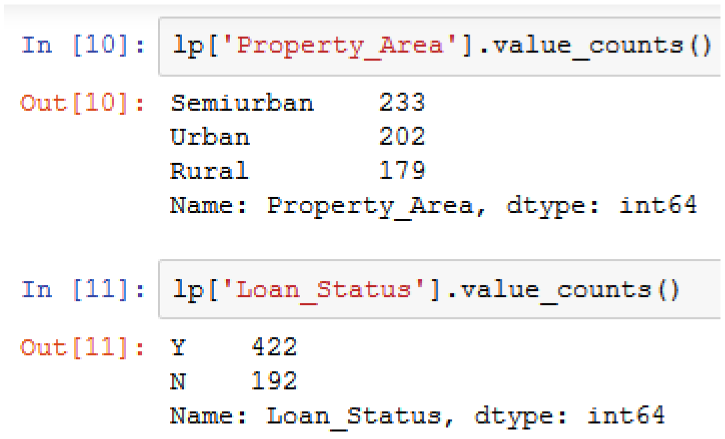
If we're looking at averages, we see that average ApplicantIncome is 5,403; average CoapplicantIncome is 1,621; average LoanAmount is 146, average Loan_Amount_Term is 342, and average Credit_History is 84% (i.e. 84% of applicants meet credit guidelines).

If we compare the mean to the 50% score, we can get an idea of how data is skewed. The further off the mean is to the 50% score, the more skewed the data could be.

Since we have non-numerical data as well, we view their frequency to see if they make some sense.



```
In [5]:  lp['Gender'].value_counts()

Out[5]:  Male      489
         Female    112
         Name: Gender, dtype: int64

In [6]:  lp['Married'].value_counts()

Out[6]:  Yes    398
         No     213
         Name: Married, dtype: int64

In [7]:  lp['Dependents'].value_counts()

Out[7]:  0     345
         1     102
         2     101
         3+     51
         Name: Dependents, dtype: int64

In [8]:  lp['Education'].value_counts()

Out[8]:  Graduate        480
         Not Graduate    134
         Name: Education, dtype: int64

In [9]:  lp['Self_Employed'].value_counts()

Out[9]:  No     500
         Yes     82
         Name: Self_Employed, dtype: int64
```



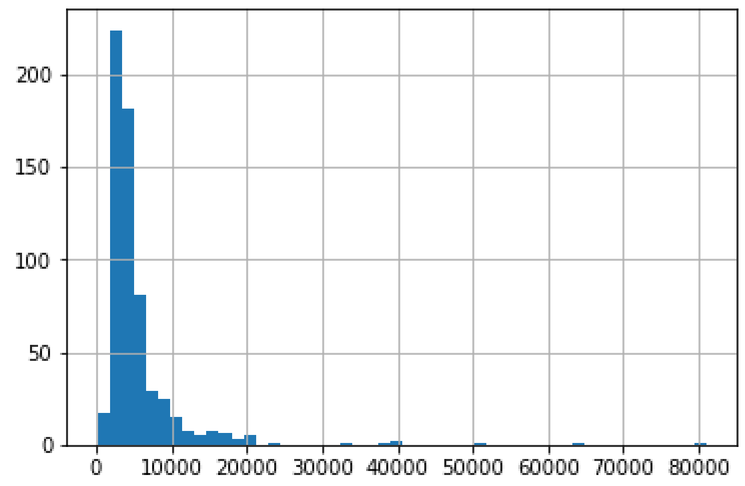
```
In [10]:  lp['Property_Area'].value_counts()

Out[10]:  Semiurban    233
          Urban        202
          Rural        179
          Name: Property_Area, dtype: int64

In [11]:  lp['Loan_Status'].value_counts()

Out[11]:  Y    422
          N    192
          Name: Loan_Status, dtype: int64
```

We can see that most applicants are male, have no dependents, have graduated, are not self-employed, and live in an either urban or semi-urban environment. Also, we seem to have a high rate of loan approval.

We next look at some histograms of some fields to see if there are any extreme values.

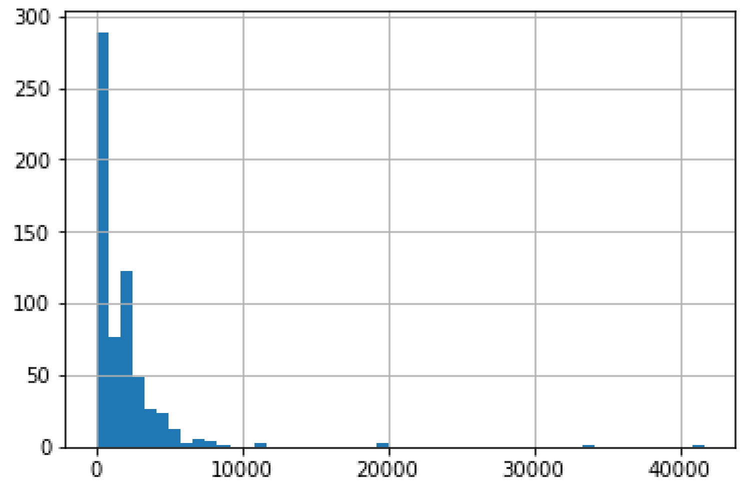
Applicant Income:

```
In [12]: lp['ApplicantIncome'].hist(bins=50)

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x992b2b0>
```



Coapplicant Income:

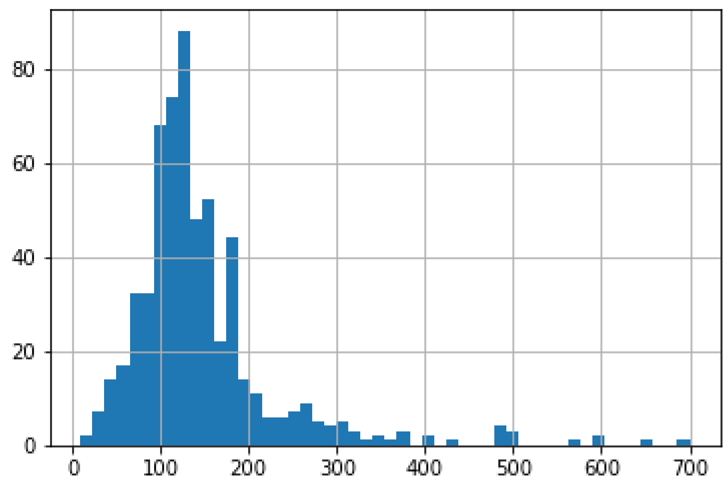
```
In [13]: lp['CoapplicantIncome'].hist(bins=50)

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x99ecd68>
```



Loan Amount:

```
In [14]: lp['LoanAmount'].hist(bins=50)
```

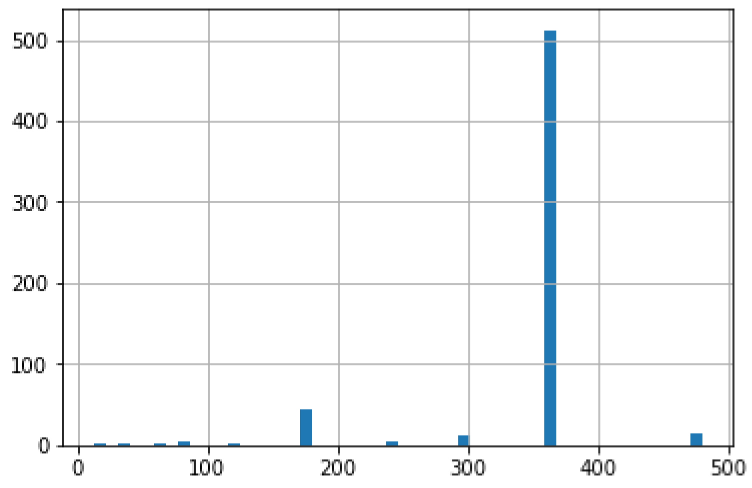
```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x9d00f60>
```



Loan Amount Term:

```
In [15]: lp['Loan_Amount_Term'].hist(bins=50)
```

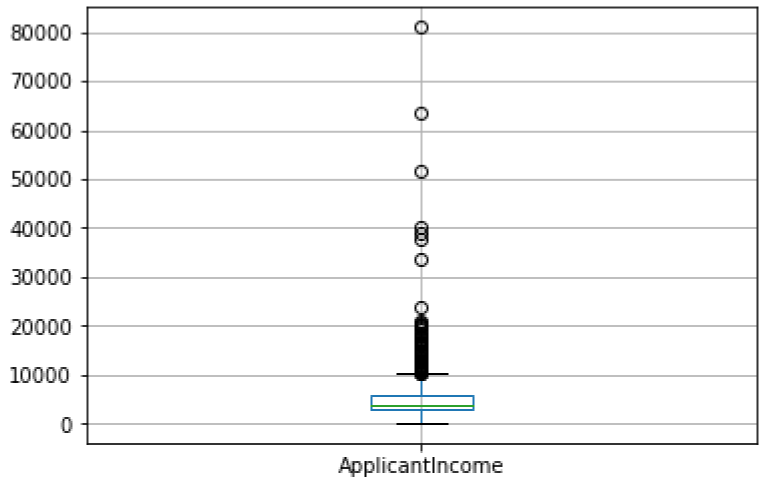
```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x9f25780>
```



Since Credit_History isn’t continuous, we don't have a need to plot that data.

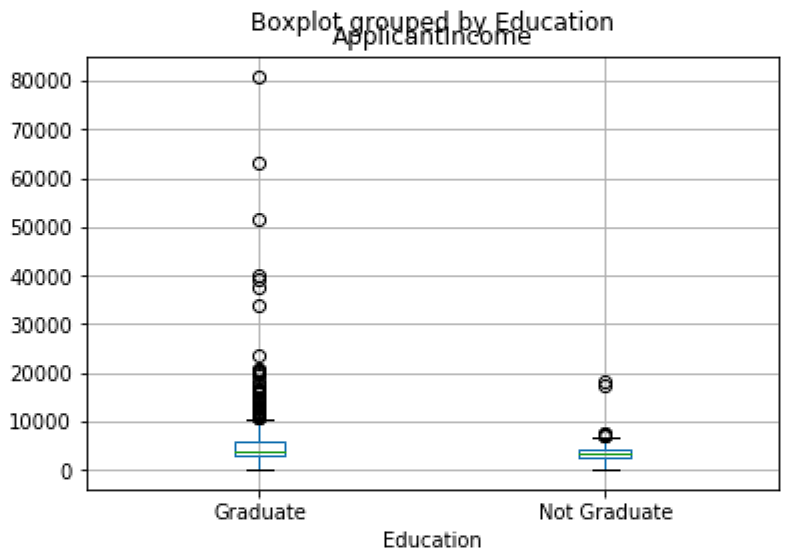
Since we are trying to determine which factors can most easily predict whether an individual will get approved for a loan, I'll go ahead and analyze fields which I intuitively think are a contributing factor. The main factor I would consider to be ApplicantIncome. Let's plot a histogram and a boxplot for ApplicantIncome so we can determine if there are any outliers or extreme values.

```
In [17]: lp.boxplot(column='ApplicantIncome')

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0xa106940>
```



The graph above shows that there are outliers in ApplicantIncome. Since I believe ApplicantIncome is heavily influenced by Education, let's segregate this data by Education. We'll use another box plot.

```
In [18]: lp.boxplot(column='ApplicantIncome', by = 'Education')

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0xa2ee588>
```

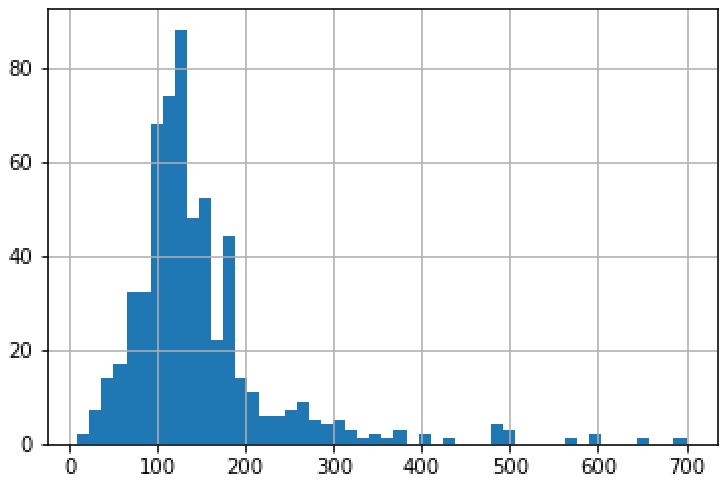


The graph above shows us that there really is no big difference between the mean incomes of graduates and non-graduates. We can see, however, that more graduates tend to apply for a loan (as also evidenced by the value count of the Education field (480 graduate vs. 134 non-graduates).

Let's take a look at LoanAmount now. We'll plot a histogram and a box plot.

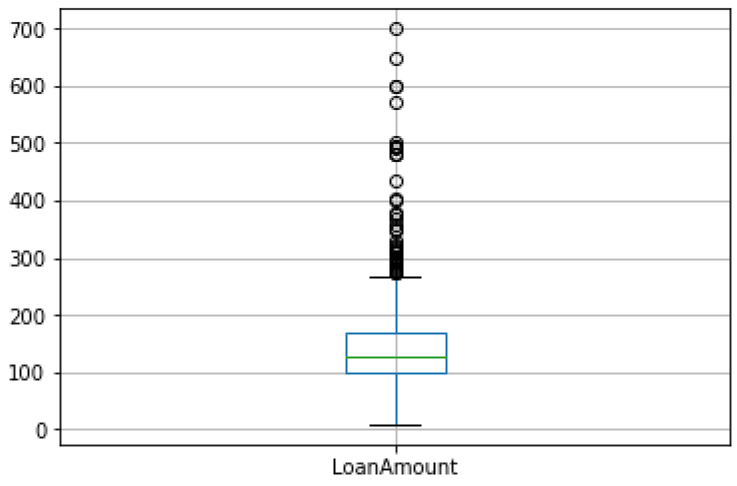
```
In [19]: lp['LoanAmount'].hist(bins=50)
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0xa351400>
```



```
In [20]: lp.boxplot(column='LoanAmount')
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0xb507b00>
```



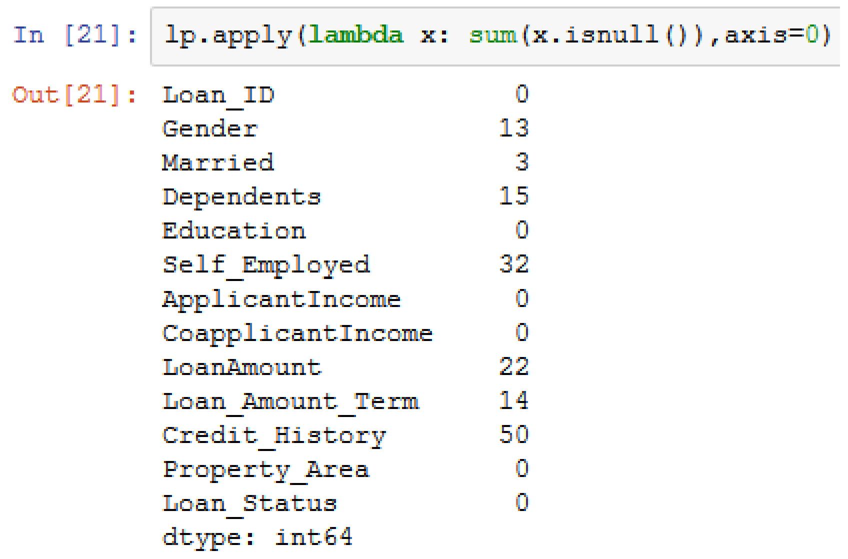
Similar to ApplicantIncome, we can also see that LoanAmount has outliers and extreme values.

Earlier we had already established that LoanAmount has missing values. So, in addition to both ApplicantIncome and LoanAmount having extreme values, LoanAmount also has missing values. We need to find a way to fix this. We obviously cannot add missing data to the data set - fortunately, we have other options.

We should refrain from using models until the data is 'clean'. We've noticed that there's missing data so we need to estimate some value for the missing fields. In particular we found both ApplicantIncome and LoanAmount to contain extreme values and LoanAmount to contain missing values.

To get a better grasp of the data, we will/should also look at the other fields to see if they contain any useful information.

Let's try to find missing variables in our whole data set - so we can get a picture of just what exactly is missing.



```
In [21]: lp.apply(lambda x: sum(x.isnull()),axis=0)

Out[21]: Loan_ID               0
         Gender               13
         Married               3
         Dependents           15
         Education             0
         Self_Employed        32
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount           22
         Loan_Amount_Term     14
         Credit_History       50
         Property_Area         0
         Loan_Status           0
         dtype: int64
```

We see here that we have missing data for a good number of variables in our data set. In the next section (Data Preprocessing), we'll explore the options to fill in these missing data.

Algorithms and Techniques

For this project, I will be utilizing at least three models. One model I will implement is logistic regression. Since this is the most basic type of regression and commonly used in predictive analysis, its results will let us examine a few things. Does a set of predictor variables do a good job in predicting an outcome variable? Which variables in particular are significant predictors of the dependent variable? And in what way do they impact the dependent variable?

However, since logistic regression is sensitive to outliers, this could significantly swing the regression results. Additionally, logistic regressions tend to overfit the data and can begin to model the random noise in the data, rather than just the relationship between the variables.

After logistic regression, we'll apply a better modeling technique – decision trees. Decision trees are known to provide higher accuracy than regression models. Unlike logistic models, decision trees can map non-linear relationships quite well.

We'll also end with random forests since it's capable of both regression and classification tasks – it also undertakes dimensional reduction methods, treats missing values, outlier values, and other steps of data exploration, and does a fairly good job. It is a type of ensemble model where a group of weak models combine to perform a powerful model.[6] I suspect random forest to yield the best results due to being able to work with all the features in the dataset and returning a feature importance matrix which we can use to select important features. Additionally, random forest has methods for balancing errors in data sets where classes are imbalanced.

Using these methods will allow us to compare their accuracy and cross validation scores and select the most appropriate model to predict loan approval.

Since I initially suspect ApplicantIncome to be the driving factor in Loan Approval, we'll see just how well these models predict Loan Approval Outcome based on results from ApplicantIncome. However, we will come to see that there are a few more variables that are just as important in determining the Loan Approval outcome, which we will explore in later sections (Implementation).

Benchmark

Since I'll be using three separate models and hypothesize that logistic model will produce the lowest accuracy, F1, and Cross-Validation score, we'll be using the results of Logistic Regression as the benchmark. Since Logistic Regression is the simplest of the models we'll be using, I hypothesized that this model will result in lower scores. But as we'll see, more complicated models do not necessarily mean better scores. We will compare the scores across the algorithms, with the different variables we'll be experimenting with, which should improve as more advanced techniques are introduced. Later, we'll discuss the scores of these algorithms and arrive at a conclusion as to which model and which variables are highly likely to impact the outcome of LoanApproval.

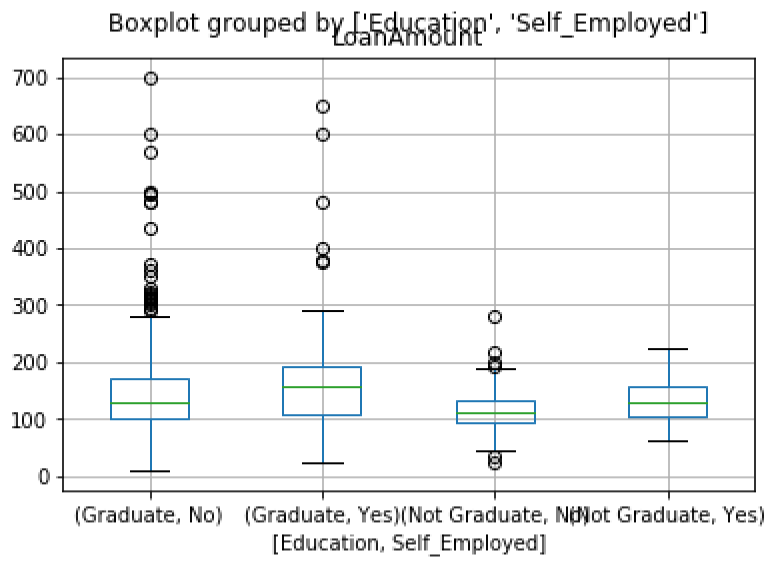
Methodology

Data Preprocessing

Continuing from out discussion above, since we're concerned with filling the values for LoanAmount (since that's the field that contains blanks), we need to try to find some solution that would fill those blanks. One solution would be to use the mean of the data. In this scenario, since we have an intuitive understanding of what can contribute to loan amount (that is, education level and, possibly, whether or not they're self-employed), let's see if there is some trend that we can use to adjust our blank values accordingly. Let's create a boxplot to see if that trend exists:

```
In [22]: lp.boxplot(column='LoanAmount', by =['Education','Self_Employed'])
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0xb551c18>
```



The graph above shows that there's some variation in the median of loan amount for each of these groups. We'll use this to impute the values for LoanAmount. Since we're looking at these two categories, education and self-employed, let's make sure that these two categories do not also have blank values.

Let's look at the count of Self_Employed.

```
In [23]: lp['Self_Employed'].value_counts()
```

```
Out[23]: No      500
         Yes      82
         Name: Self_Employed, dtype: int64
```

I'm going to make a judgment call and assign the blank values to No since a majority of applicants belong to the No group (roughly 86%)

```
In [24]:  lp['Self_Employed'].fillna('No',inplace=True)
```

Education has no blank values so we're fine using that.

We'll code some function where we can extract the median value for all groups of values in Self_Employed and Education [(Graduate, No), (Graduate, Yes), (Not Graduate, No), (Not Graduate, Yes)].

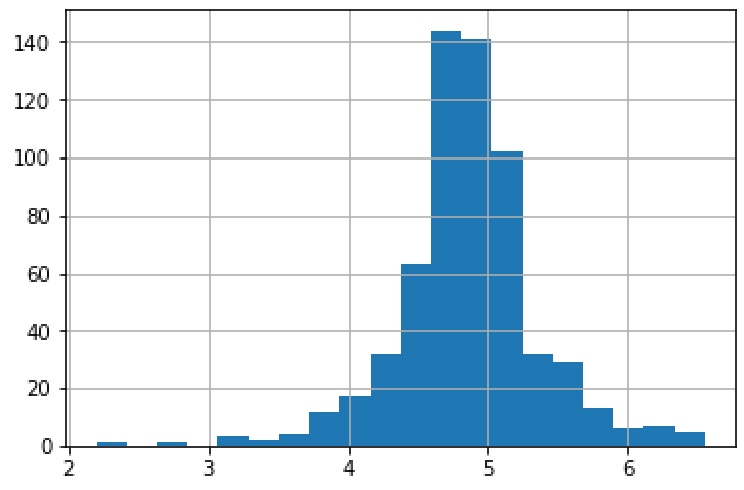
```
In [25]: #Create Pivot Table to extract these values
         extractUniqueValues = lp.pivot_table(values='LoanAmount', index='Self_Employed' ,columns='Education', aggfunc=n.median)

         #Return these values
         def replacement(x):
          return extractUniqueValues.loc[x['Self_Employed'],x['Education']]

         #Replace missing values
         lp['LoanAmount'].fillna(lp[lp['LoanAmount'].isnull()].apply(replacement, axis=1), inplace=True)
```

The code block above replaces the missing values in LoanAmount with the median of each unique combination of categories for Self_Employed and Education.

At this point, we are done filling in blank values for the fields we are analyzing (ApplicantIncome and LoanAmount)

Now, we'll look at outlier data. We'll apply a log function to both ApplicantIncome and LoanAmount to 'normalize' the curve.

```
In [26]:  lp['LoanAmount_Normalized'] = n.log(lp['LoanAmount'])
          lp['LoanAmount_Normalized'].hist(bins=20)

Out[26]:  <matplotlib.axes._subplots.AxesSubplot at 0xa373438>
```



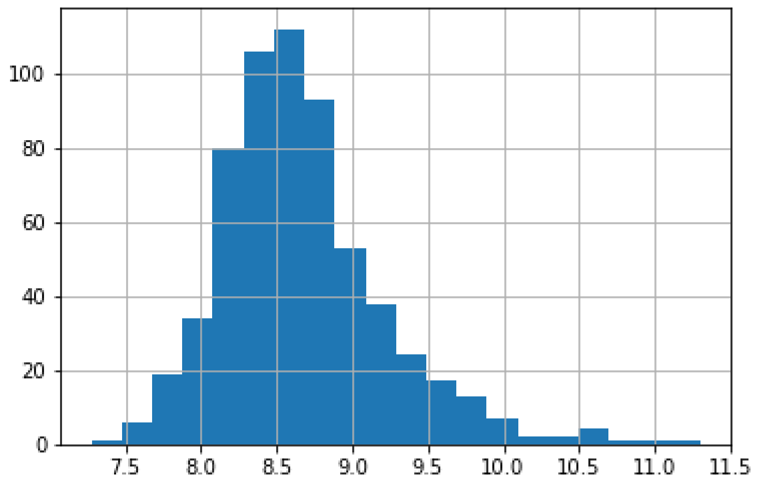
After 'normalizing' the LoanAmount, we see that this distribution seems more average.

For ApplicantIncome, a reason for the outliers may be because the applicants have coapplicants with higher incomes. We'll also normalize this, but we'll include the income from the coapplicants as well.

We didn't do any further transformations on LoanAmount because although there were outliers for LoanAmount, we realize that this is possible due to the need of requesting high house loans.

We'll create a new field containing the sum of both applicant and coapplicant, then apply the log function to normalize it.

```
In [27]: lp['TotalIncome'] = lp['ApplicantIncome'] + lp['CoapplicantIncome']
         lp['TotalIncome_Normalized'] = n.log(lp['TotalIncome'])
         lp['TotalIncome_Normalized'].hist(bins=20)

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xb7f4d68>
```



The graph above shows a much better distribution of ApplicantIncome.

At this point, we need to solve the missing data for Gender, Married, Dependents, Loan_Amount_Term, and Credit_History. For each of these categories, we'll be using different techniques - simply so we can practice the various options for handling missing data.

- Gender - Arbitrarily assign the 13 missing data as Male since the majority of applicants are male.
- Married - Arbitrarily assign the 3 missing data as Married since the majority of applicants are Married.
- Dependents - Arbitrarily assign the 15 missing data as 0 since the majority of dependents are 0.
- Loan_Amount_Term - Since the mean is close to 50% of the data, we'll replace all missing values with the mean.
- Credit_History - I'll assume that if the Loan_Status was granted, they had good credit and if the Loan_Status was denied, they have bad credit. I.E. Loan_Status = N then Credit_History = 0 and if Loan_Status = Y then Credit_History = 1. To do this, I'm going to encode Loan_Status to a numerical value and then fill Credit_History with the corresponding Loan_Status values.

Here are the transformations for the above variables:

```
In [28]: #Gender
         lp['Gender'].fillna('Male', inplace=True)
         #Married
         lp['Married'].fillna('Yes', inplace=True)
         #Dependents
         lp['Dependents'].fillna('0', inplace=True)
         #Loan_Amount_Term
         lp['Loan_Amount_Term'].fillna(lp['Loan_Amount_Term'].mean(), inplace=True)
         #Credit_History
         lp['Loan_Status'] = lp['Loan_Status'].map(lambda x: 1 if x == 'Y' else 0) #Encode first
         lp['Credit_History'].fillna(lp['Loan_Status'], inplace = True) #Then replace
```

Let's now verify that there are no blanks in the data and that no other rows were deleted.

```
In [29]: lp.apply(lambda x: sum(x.isnull()),axis=0)
```

```
Out[29]: Loan_ID                    0
         Gender                     0
         Married                    0
         Dependents                 0
         Education                  0
         Self_Employed              0
         ApplicantIncome            0
         CoapplicantIncome          0
         LoanAmount                 0
         Loan_Amount_Term           0
         Credit_History             0
         Property_Area              0
         Loan_Status                0
         LoanAmount_Normalized      0
         TotalIncome                0
         TotalIncome_Normalized     0
         dtype: int64
```

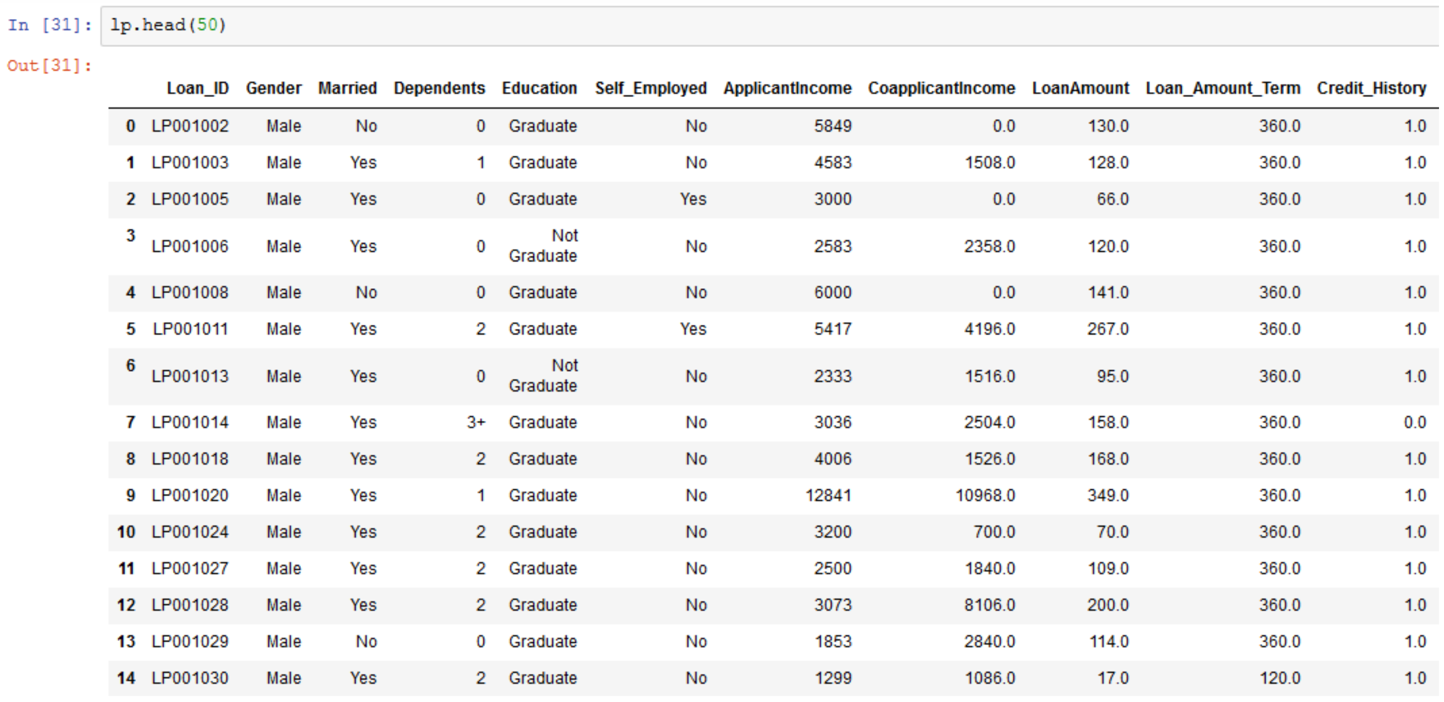
Let's describe the table now and see how much has changed.

```
In [30]: lp.describe()
```

```
Out[30]:
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	LoanAmount_Normalized	TotalIncome
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	145.764658	342.000000	0.833876	0.687296	4.857146	7024.705081
std	6109.041673	2926.248369	84.145700	64.372489	0.372495	0.463973	0.496392	6458.663872
min	150.000000	0.000000	9.000000	12.000000	0.000000	0.000000	2.197225	1442.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000	0.000000	4.607658	4166.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000	1.000000	4.852030	5416.500000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000	1.000000	5.104426	7521.750000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000	1.000000	6.551080	81000.000000

Let's print a few records and see how much has changed.



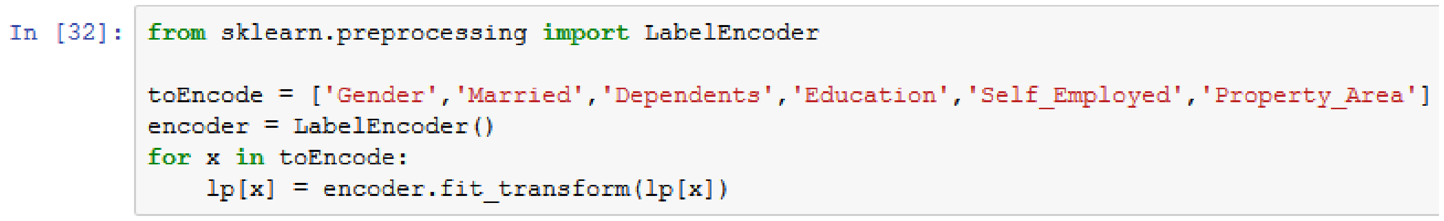
```
In [31]: lp.head(50)
Out[31]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	130.0	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.0	360.0	0.0
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.0	360.0	1.0
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.0	360.0	1.0
10	LP001024	Male	Yes	2	Graduate	No	3200	700.0	70.0	360.0	1.0
11	LP001027	Male	Yes	2	Graduate	No	2500	1840.0	109.0	360.0	1.0
12	LP001028	Male	Yes	2	Graduate	No	3073	8106.0	200.0	360.0	1.0
13	LP001029	Male	No	0	Graduate	No	1853	2840.0	114.0	360.0	1.0
14	LP001030	Male	Yes	2	Graduate	No	1299	1086.0	17.0	120.0	1.0

We have completed all steps required to ensure that the data is 'clean'. We will now implement our algorithms to see which model provides the best scores.

Implementation

We're now going to use scikit-learn to create predictive models with our data. We also need to convert all our data to numeric values. We'll do this by encoding the categorical data using LabelEncoder.



```
In [32]: from sklearn.preprocessing import LabelEncoder

         toEncode = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area']
         encoder = LabelEncoder()
         for x in toEncode:
             lp[x] = encoder.fit_transform(lp[x])
```

Let's print the set to see what it looks like now.

```
In [33]: lp.dtypes

Out[33]: Loan_ID                       object
         Gender                         int64
         Married                        int64
         Dependents                     int64
         Education                      int64
         Self_Employed                  int64
         ApplicantIncome                int64
         CoapplicantIncome            float64
         LoanAmount                   float64
         Loan_Amount_Term             float64
         Credit_History               float64
         Property_Area                  int64
         Loan_Status                    int64
         LoanAmount_Normalized        float64
         TotalIncome                  float64
         TotalIncome_Normalized       float64
         dtype: object
```

We're going to insert a few models from scikit-learn - a logistic regression model, a decision tree model, and a random forest model.

We will take a look at accuracy score, but more importantly we'll be looking at F1 score since this is a weighted average of precision and recall. F1 score is a better metric for imbalanced data. We'll also implement K-Fold cross validation to help us know the effectiveness of model performance.

We created a function to test the classification model and access performance. This will allow us easy access to the score we need.

```
In [34]: #Import scikit learn models:
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.tree import DecisionTreeClassifier
         from sklearn import metrics

         #Import K-Fold cross validation
         from sklearn.cross_validation import KFold

         #Creation a function to test the classification model and access performance:
         def performance_model(model, data, predictors, outcome):
             model.fit(data[predictors],data[outcome]) #Fit
             predictions = model.predict(data[predictors]) #Predict
             #Print accuracy
             print ("Accuracy : ", metrics.accuracy_score(predictions,data[outcome]))
             #Print F1 score
             print ("F1 Score : ", metrics.f1_score(predictions,data[outcome]))

             #Perform KFold cross-validation with 10 folds
             kf = KFold(data.shape[0], n_folds=10)
             error = []
             for train, test in kf:
                 train_predictors = (data[predictors].iloc[train,:])
                 train_target = data[outcome].iloc[train]
                 model.fit(train_predictors, train_target)
                 error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

             print ("Cross-Validation Score : ",n.mean(error))

             #Fit the model again:
             model.fit(data[predictors],data[outcome])
```

Let's try the Logistic Regression model first. Since we're trying to predict Loan_Status, we'll set the Outcome variable to Loan_Status. Since this is the benchmark (Refer to Section II. Benchmark), our predictor variable will simply be Credit_History. (Which we intuitively decided that if we have good credit, we'll assume we'll get approved for a loan). Then we call our function with these parameters.

```
In [35]: outcome = 'Loan_Status' #What we're trying to predict
         model = LogisticRegression() #Our model
         predictors = ['Credit_History'] #What we're trying to predict on
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888650963597
         Cross-Validation Score :  0.8307244844
```

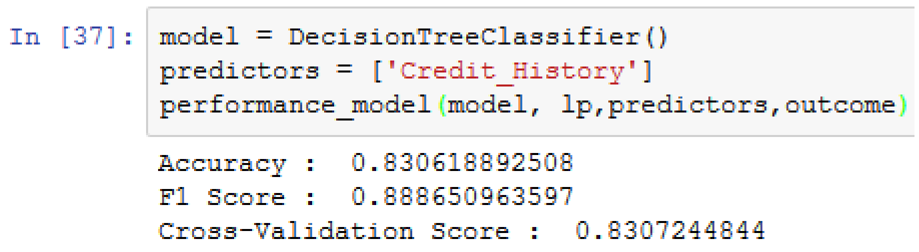
Let's try Regression model on a different set of predictor variables.

```
In [36]: model = LogisticRegression() #Our model
         predictors = ['Credit_History','Education','Married','Self_Employed','Property_Area']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888650963597
         Cross-Validation Score :  0.8307244844
```

We see here that Logistic Regression outputs the same scores across both tests. We would think that adding more features increases the scores, but instead we realize that Credit_History is the dominating factor.

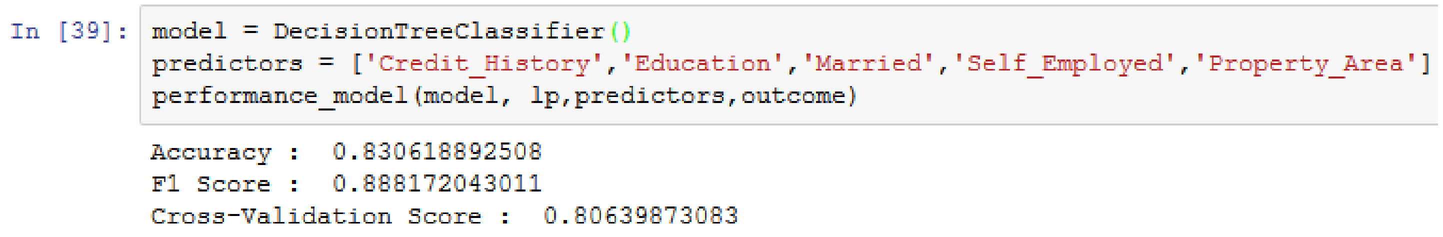
Let's see what the DecisionTreeClassifier can tell us.

Let's try DecisionTreeClassifier on the same initial set of variables.



```
In [37]: model = DecisionTreeClassifier()
         predictors = ['Credit_History']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888650963597
         Cross-Validation Score :  0.8307244844
```

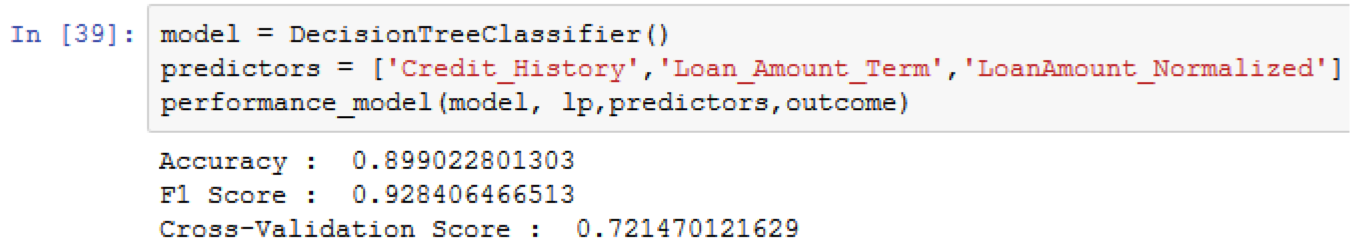
Now, let's try DecisionTreeClassifier again on the same (second) set of predictor variables.



```
In [39]: model = DecisionTreeClassifier()
         predictors = ['Credit_History','Education','Married','Self_Employed','Property_Area']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888172043011
         Cross-Validation Score :  0.80639873083
```

We see that DecisionTreeClassifier received the same score on accuracy, slightly lesser scores on F1, and even less or Cross-Validation. This goes to show us that a slightly more complicated model doesn’t necessarily equal better results!

Let's try DecisionTreeClassifier on a different set of variables. This time, on numerical variables.



```
In [39]: model = DecisionTreeClassifier()
         predictors = ['Credit_History','Loan_Amount_Term','LoanAmount_Normalized']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.899022801303
         F1 Score :  0.928406466513
         Cross-Validation Score :  0.721470121629
```

We see here that DecisionTreeClassifier outputs better scores. Which is what we expected.

In the first example, DecisionTreeClassifier outputs similar scores to Logistic Regression on the categorical variables and this is because Credit History is again dominating over them.

If we run the model on numerical values, we see the scores increase! Cross-validation score decreases a little - this is an early sign of overfitting and a sign that this model doesn’t generalize well.

Let's see what the RandomForestClassifier can tell us. We'll just use Credit_History to get a baseline. For now, we'll arbitrarily set N_estimators to 100.

```
In [40]: model = RandomForestClassifier(n_estimators=100)
         predictors = ['Credit_History']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888650963597
         Cross-Validation Score :  0.8307244844
```

We see here that the numbers are almost identical to the numbers before.

This time, we're going to use all the features since a RandomForestClassifier can return which features are most important - we can use this output to select the correct features.

```
In [41]: model = RandomForestClassifier(n_estimators=100)
         predictors = ['Gender', 'Married', 'Dependents', 'Education',
                 'Self_Employed', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
                  'LoanAmount_Normalized','TotalIncome_Normalized']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  1.0
         F1 Score :  1.0
         Cross-Validation Score :  0.81287678477
```

Immediately, we can see that accuracy and F1 score is 100% - we overfit the data. This makes sense considering we used all the variables available to us. Since we used a RandomForestClassifier, we can determine which features are most important.

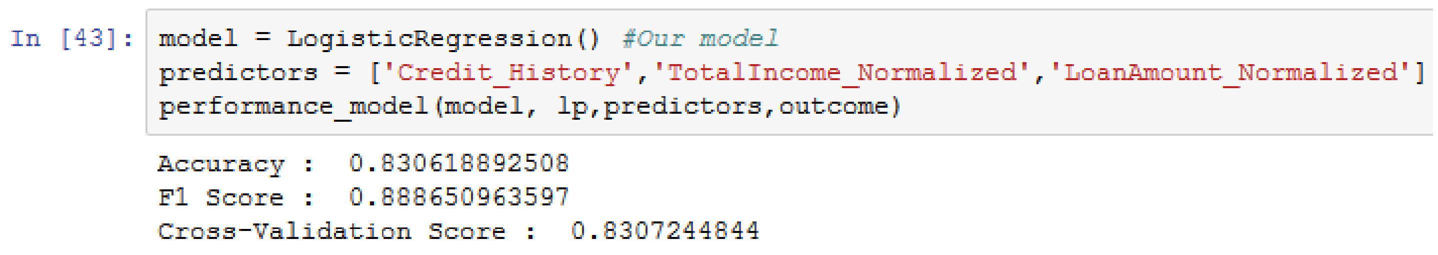
Let's print the important features.

```
In [40]: impFeatures = p.Series(model.feature_importances_, index=predictors).sort_values(ascending=False)
         print (impFeatures)

         Credit_History            0.326351
         TotalIncome_Normalized    0.244291
         LoanAmount_Normalized     0.210065
         Dependents                0.047365
         Loan_Amount_Term          0.044549
         Property_Area             0.044063
         Married                   0.023546
         Education                 0.021101
         Gender                    0.020730
         Self_Employed             0.017938
         dtype: float64
```

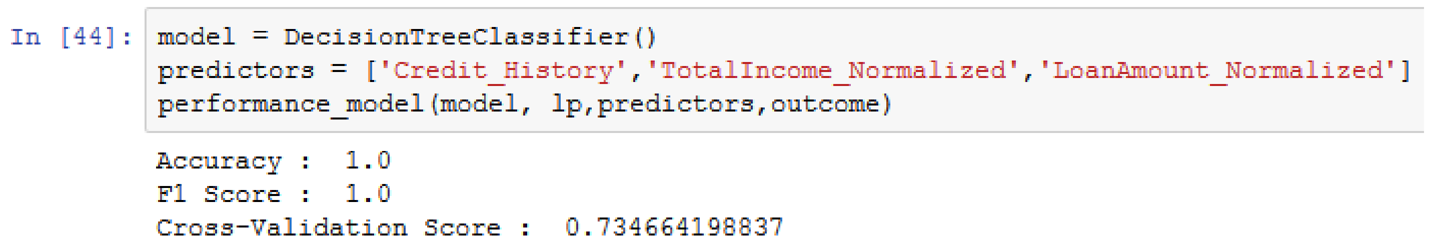
The table above shows us which features are the most important. We'll arbitrarily use the top 3 features and rerun our models with those features to see just how much has changed. We'll discuss the changes in the next section, Refinement.

Refinement

Since we established the important features, let's run our previous models with the those features and see our scores and arrive at a conclusion.

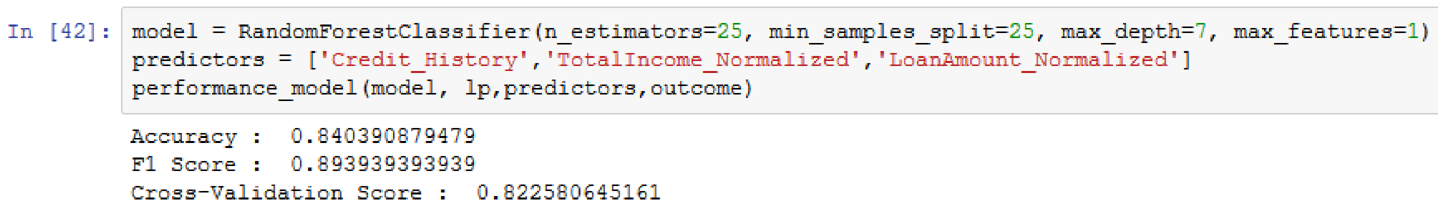


```
In [43]: model = LogisticRegression() #Our model
         predictors = ['Credit_History','TotalIncome_Normalized','LoanAmount_Normalized']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.830618892508
         F1 Score :  0.888650963597
         Cross-Validation Score :  0.8307244844
```



```
In [44]: model = DecisionTreeClassifier()
         predictors = ['Credit_History','TotalIncome_Normalized','LoanAmount_Normalized']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  1.0
         F1 Score :  1.0
         Cross-Validation Score :  0.734664198837
```

I modified a few parameters in RandomForestClassifier. I randomly selected n_estimators, min_samples_split, max_depth, max_features, as my sandbox and tweaked numbers as necessary to achieve high scores. This was the result



```
In [42]: model = RandomForestClassifier(n_estimators=25, min_samples_split=25, max_depth=7, max_features=1)
         predictors = ['Credit_History','TotalIncome_Normalized','LoanAmount_Normalized']
         performance_model(model, lp,predictors,outcome)

         Accuracy :  0.840390879479
         F1 Score :  0.893939393939
         Cross-Validation Score :  0.822580645161
```

Results

Model Evaluation and Validation

We can see above that running these models with the refined variables (the ones found most important by RandomForestClassifier) showed greatly improved scores for DecisionTreeClassifier and RandomForestClassifier. LogisticRegression showed identical scores and this makes sense considering that Credit_History is dominating over the data. Based off the results above and from our refinement, we can conclude that the best model would be the RandomForestClassifier. DecisionTreeClassifier overfits the data, whereas the accuracy and F1 score for the LogisticRegression isn't as good as our RandomForestClassifier. RandomForestClassifier seems to be the best-balanced approach to this problem.

In RandomForestClassifier, I modified the parameters n_estimators, min_samples_split, max_depth, max_features, and edited numbers as necessary to achieve high scores. For sake of playing with the model, the numbers I assigned to these variables were purely arbitrary. My goal was to increase the scores as best as I could. It's also interesting to note that all models were all equally insufficient when predicting on just one predictor variable, Credit_History.

I had initially suspected that LogisticRegression would provide the worst scores simply because it was the simplest model. It was definitely *one* of the worst models, but it turns out, that DecisionTreeClassifier proved to be worse if we added more dimension in its analysis. This observation lines with my initial hypothesis that using a more sophisticated model does not necessarily mean it will yield better results. The final parameters for the best model we've chosen makes sense, given that we output exactly which variables have the most impact (from the feature_importances method).

In order to test whether the model generalizes well with unseen data, we implemented the Cross-Validation score, which, again, is the score for assessing how the results of a statistical analysis will generalize to an independent data set. Our cross-validation score was slightly lower than some score in our other models, but if we consider holistically at accuracy and F1, we can see that we arrive at a fair conclusion.

Overall, it seems fair to say that given the amount of categories, the model we suggested is robust enough for the problem we're trying to solve and the results can be trusted.

Justification

Model	***Predictors***	***Accuracy***	***F1***	***Cross-Validation***	***Average Score***
LogisticRegression() **BENCHMARK**	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
LogisticRegression()	Credit_History','Education','Married'	0.830618893	0.888650964	0.830724484	0.849998114
LogisticRegression() **REFINED**	Credit_History','TotalIncome_Norma	0.830618893	0.888650964	0.830724484	0.849998114
DecisionTreeClassifier()	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
DecisionTreeClassifier()	Credit_History','Education','Married'	0.830618893	0.888172043	0.806398731	0.841729889
DecisionTreeClassifier()	Credit_History','Loan_Amount_Term	0.899022801	0.928406467	0.721470122	0.84963313
DecisionTreeClassifier() **REFINED**	Credit_History','TotalIncome_Norma	1	1	0.744500264	0.914833421
RandomForestClassifier()	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
RandomForestClassifier() **ALL VAR**	Gender', 'Married', 'Dependents', 'Ed	1	1	0.812876785	0.937625595
RandomForestClassifier() **REFINED**	Credit_History','TotalIncome_Norma	0.846905537	0.897603486	0.820967742	0.855158922

In analyzing all our scores across all models and all predictors, we come to realize that the best average score across all three metrics (Accuracy, F1, and Cross-Validation) would be RandomForestClassifier() on the refined set of predictors. That is, the set of variables that we have established play the biggest role in determining the outcome of loan approval. We ignore the refined DecisionTreeClassifier() with the refined variables because it shows clear signs of overfitting – it's accuracy and F1 scores are too high and it cannot generalize well because of its low cross-validation score. Likewise, we also ignore RandomForestClassifier() where we used all the variables for the same reason.

After calculating the average score across all metrics and excluding those we deem not suitable due to overfitting, we realize that RandomForestClassifier with the refined variables is the best model.

However, we should note that average score across all the metrics should not be the sole determining factor in determining which model fits best. If we approach our problem that way, then we would apply the best model to RandomForestClassifier() with all the predictor variables. We avoided this mistake because we looked at the individual scores and realized from both Accuracy and F1, that we cannot use RandomForestClassifier() with all the predictor variables because it overfits the data. We apply this same reasoning to why we cannot select the second-best average score, DecisionTreeClassifier() with the refined variables.

The results from the RandomForestClassifier() with the refined variables aren't significantly stronger than the benchmark, but since we've established that the model is better, it stands to reason that we will use the better model – even for a small percentage difference.

Conclusion

Free-Form Visualization

Model	***Predictors***	***Accuracy***	***F1***	***Cross-Validation***	***Average Score***
LogisticRegression() **BENCHMARK**	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
LogisticRegression()	Credit_History','Education','Married'	0.830618893	0.888650964	0.830724484	0.849998114
LogisticRegression() **REFINED**	Credit_History','TotalIncome_Norma	0.830618893	0.888650964	0.830724484	0.849998114
DecisionTreeClassifier()	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
DecisionTreeClassifier()	Credit_History','Education','Married'	0.830618893	0.888172043	0.806398731	0.841729889
DecisionTreeClassifier()	Credit_History','Loan_Amount_Term	0.899022801	0.928406467	0.721470122	0.84963313
DecisionTreeClassifier() **REFINED**	Credit_History','TotalIncome_Norma	1	1	0.744500264	0.914833421
RandomForestClassifier()	Credit_History	0.830618893	0.888650964	0.830724484	0.849998114
RandomForestClassifier() **ALL VAR**	Gender', 'Married', 'Dependents', 'Ed	1	1	0.812876785	0.937625595
RandomForestClassifier() **REFINED**	Credit_History','TotalIncome_Norma	0.846905537	0.897603486	0.820967742	0.855158922

It's important that we realize from the graph above that having more complicated models does not necessarily yield better results. That's an important fact I'd like to mention because we would think, intuitively, that using a more robust model that's capable of working with many pieces of data, (including many different types of data, or even missing or incomplete data), would result in better numbers. This is not true. This goes to show that each problem and data set requires a careful approach in selecting which model is more appropriate to use. Each model has its strengths and weaknesses and it's a matter of experimenting on those models to determine which would produce the outcomes we're looking for.

Reflection

The process analyzed data consisting of many numerical and categorical values. My goal was to try to find a model which can accurately predict whether a customer (one line item in the data set) will be approved for a loan. The customer will be either approved or non-approved, therefore this presents a binary classifier problem. I first analyzed and described the data to determine the mean values, the most frequent values, the maximum, the minimum, etc. Some of most valuable pieces of information I obtained in describing the data are the pieces of data which did not have any information and the pieces of data that were outliers – some data that just didn't make sense. I needed to determine what steps was needed to take to 'clean' this data. This process was difficult in that there were many approaches to this problem. A major guiding factor in how I approached my data set was ruled by <u>intuition</u>. I often asked myself, which variables make sense in having an impact on another variable? I've concluded that the variable that could most likely impact a loan approval decision would be applicant income, and I used this as my guiding principle. I also looked at the distribution analysis for a few variables to see if there were any outliers and if there were, what I could do to mitigate those extreme values. I've learned that an acceptable approach would be to normalize those outlier variables by getting the log of those values, thus normalizing the data – I stuck with this approach.

Once I analyzed the different categories, I needed to find a solution to the missing pieces of data in the data set. Again, using intuition, I implemented rules to fill these data sets which made sense to me. The rules I implemented were as follows:

- Gender - Arbitrarily assign the 13 missing data as Male since the majority of applicants are male.
- Married - Arbitrarily assign the 3 missing data as Married since the majority of applicants are Married.
- Dependents - Arbitrarily assign the 15 missing data as 0 since the majority of dependents are 0.
- Loan_Amount_Term - Since the mean is close to 50% of the data, we'll replace all missing values with the mean.
- Credit_History - I'll assume that if the Loan_Status was granted, they had good credit and if the Loan_Status was denied, they have bad credit. I.E. Loan_Status = N then Credit_History = 0 and if Loan_Status =

Y then Credit_History = 1. To do this, I'm going to encode Loan_Status to a numerical value and then fill Credit_History with the corresponding Loan_Status values.

Once we normalized the outlier values and filled the missing data, I implemented the three models of my choosing using a variety of categories to determine which categories were best at predicting the outcome of the problem. In order to define which was the best solution, I used a baseline model LogisticRegression to gather the accuracy, F1, and cross-validation score. Every attempt at creating a better model relied on increasing the score from the baseline. After playing with some features, I concluded that RandomForestClassifier, using the most important features, was the best model at predicting whether a customer will get approved for a loan.

The most interesting aspect of the project was from seeing the scores fluctuate based on which parameters we determined were the 'predictor' parameters. On the other hand, the most difficult aspect was trying to find a way to 'clean' the data properly – particularly, what to do with the blank data. I'm most certain there are more elegant solutions, but for the time being, I stuck with my <u>intuition</u>, I stuck with a reasoning that made sense.

Since I initially expected RandomForestClassifier to yield the best results, the final model and solution fit my expectation for the model. Now, since this problem relies on this particular set of categorical data, I don't think it would be fair for this particular model to be used in a general setting to solve these types of problems. That isn't to say that this model is incorrect, it just simply means that future data needs to be re-evaluated to even determine which model of the many out there seems to be the best to tackle the data. However, if I were to generalize the process itself, I'd like to think that my process can indeed be generalized to similar problems. (My process in a nutshell: Explore > Describe > Analyze > Hypothesize > Clean > Build > Repeat)

Improvement

One area from my implementation that could be greatly improved would be how I filled missing values. Since there are so many techniques to counter missing values, I'm almost certain some techniques would better fit the data I was working with. The techniques I used on the blank values consisted of simple replacement (replacing all values with some value, replacing blanks with the mean and replacing blanks with output from another category). In theory, I could have deleted empty entries, replaced values with the median or mode, create another prediction model to handle the missing data, or used KNN imputation[7].

Alternatively, I could have also improved how I handled outlier values. In this problem, I normalized the values by getting the log. Again, I had many options like deleting those values, treat those values separately, or find some other method of transforming those variables. Natural log of a value seemed to make the most sense to me.

All of the options for treating missing variables and handling outlier variables would have made for a better implementation and possibly have resulted in a better score. However, in the interest of time and ability, I decided to stick with techniques I was familiar with (though in the future, I'd love to explore these possibilities).

Another approach to improvement would be to consider using different types of models. I initially stuck with LogisticRegression, DecisionTreeClassifier, and RandomForestClassifier – of course, there are many more models that could be used that could, arguably, be a better fit for the data.

I feel confident in using the final solution I provided, but I have no doubt that a better solution exists.

Citations

[1]Sengupta, Satadru. “The Power of Machine Learning in Insurance.” Cloudera VISION, Cloudera, 11 May 2017, vision.cloudera.com/the-power-of-machine-learning-in-insurance/.

[2]Brownlee, Jason. “Machine Learning Algorithm Recipes in Scikit-Learn.” Machine Learning Mastery, Machine Learning Mastery, 21 Sept. 2016, machinelearningmastery.com/get-your-hands-dirty-with-scikit-learn-now/.

[3]“Practice Problem: Loan Prediction III | Knowledge and Learning.” DataHack : Biggest Data Hackathon Platform for Data Scientists, Analytics Vidhya, 25 May 2016, datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/.

[4] “F1 Score.” Wikipedia, Wikimedia Foundation, 28 Aug. 2017, en.wikipedia.org/wiki/F1_score.

[6]Team, Analytics Vidhya Content, et al. “A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python).” *Analytics Vidhya*, Analytics Vidhya, 1 May 2017, www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/.

[7] Ray, Sunil, et al. “A Complete Tutorial Which Teaches Data Exploration in Detail.” Analytics Vidhya, Analytics Vidhya, 1 May 2017, www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/.