Part 1

Bookmark this page

Exploratory analysis in Python using Pandas

In order to explore our data further, let me introduce you to another animal (as if Python was not enough!) - Pandas

Pandas is one of the most useful data analysis library in Python (I know these names sounds weird, but hang on!). They have been instrumental in increasing the use of Python in data science community. We will now use Pandas to read a data set from an Analytics Vidhya competition, perform exploratory analysis and build our first basic categorization algorithm for solving this problem.

Before loading the data, lets understand the 2 key data structures in Pandas - Series and DataFrames

Introduction to Series and Dataframes

Series can be understood as a 1 dimensional labelled / indexed array. You can access individual elements of this series through these labels.

A dataframe is similar to Excel workbook - you have column names referring to columns and you have rows, which can be accessed with use of row numbers. The essential difference being that column names and row numbers are known as column and row index, in case of dataframes.

Series and dataframes form the core data model for Pandas in Python. The data sets are first read into these dataframes and then various operations (e.g. group by, aggregation etc.) can be applied very easily to its columns.

More: 10 Minutes to Pandas

Practice data set - Loan Prediction Problem

You can download the dataset from here. Here is the description of variables:

```
VARIABLE DESCRIPTIONS:
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)
```

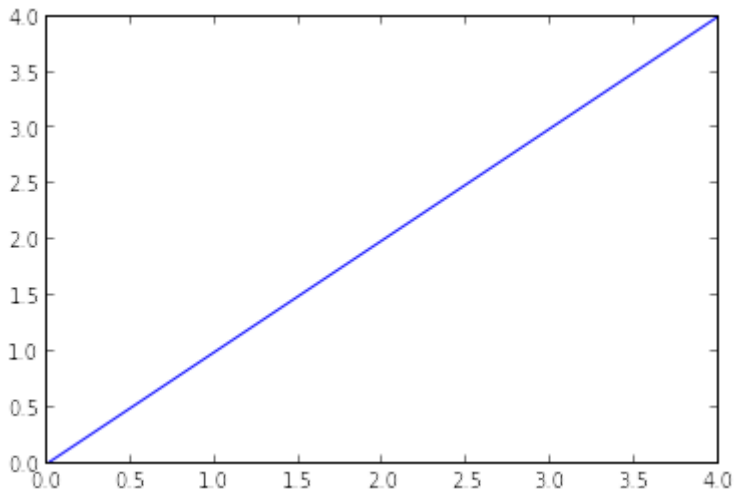
Let's begin with exploration

To begin, start iPython interface in Inline Pylab mode by typing following on your terminal / windows command prompt:

```
ipython notebook --pylab=inline
```

This opens up iPython notebook in pylab environment, which has a few useful libraries already imported. Also, you will be able to plot your data inline, which makes this a really good environment for interactive data analysis. You can check whether the environment has loaded correctly, by typing the following command (and getting the output as seen in the figure below):

```
plot(arange(5))
```



I am currently working in Linux, and have stored the dataset in the following location:

/home/kunal/Downloads/Loan_Prediction/train.csv

Importing libraries and the data set:

Following are the libraries we will use during this tutorial:

- numpy
- matplotlib
- pandas

Please note that you do not need to import matplotlib and numpy because of Pylab environment. I have still kept them in the code, in case you use the code in a different environment.

After importing the library, you read the dataset using function read_csv(). This is how the code looks like till this stage:

```
import pandas as pd
import numpy as np
import matplotlib as plt

df = pd.read_csv("/home/kunal/Downloads/Loan_Prediction/train.csv")
#Reading the dataset in a dataframe using Pandas
```

Quick Data Exploration

Once you have read the dataset, you can have a look at few top rows by using the function head()

```
df.head(10)
```

```
In [3]: df.head(10)  #Printing first 10 rows of dataset
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	Male	No	0	Graduate	No	5849	0	NaN	360	1
1	LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1
4	LP001008	Male	No	0	Graduate	No	6000	0	141	360	1
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0
8	LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1
9	LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1

This should print 10 rows. Alternately, you can also look at more rows by printing the dataset. Next, you can look at summary of numerical fields by using describe() function

```
df.describe()
```

```
In [4]: df.describe()  #Get summary of numerical variables
```

Out[4]:

	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

describe() function would provide count, mean, standard deviation (std), min, quartiles and max in its output (Read this article to refresh basic statistics to understand population distribution)

Here are a few inferences, you can draw by looking at the output of describe() function:

1. LoanAmount has (614 – 592) 22 missing values.
2. Loan_Amount_Term has (614 – 600) 14 missing values.
3. Credit_History has (614 – 564) 50 missing values.
4. We can also look that about 84% applicants have a credit_history. How? The mean of Credit_History field is 0.84 (Remember, Credit_History has value 1 for those who have a credit history and 0 otherwise)
5. The ApplicantIncome distribution seems to be in line with expectation. Same with CoapplicantIncome

Please note that we can get an idea of a possible skew in the data by comparing the mean to the median, i.e. the 50% figure.

For the non-numerical values (e.g. Property_Area, Credit_History etc.), we can look at frequency distribution to understand whether they make sense or not. The frequency table can be printed by following command:

```
df['Property_Area'].value_counts()
```

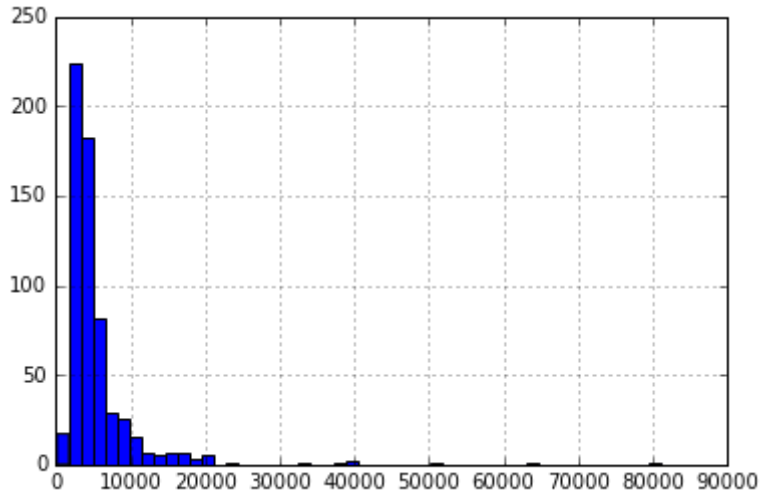
Similarly, we can look at unique values of port of credit history. Note that dfname['column_name'] is a basic indexing technique to acess a particular column of the dataframe. It can be a list of columns as well. For more information, refer to the "10 Minutes to Pandas" resource shared above.

Distribution analysis

Now that we are familiar with basic data characteristics, let us study distribution of various variables. Let us start with numeric variables - namely ApplicantIncome and LoanAmount

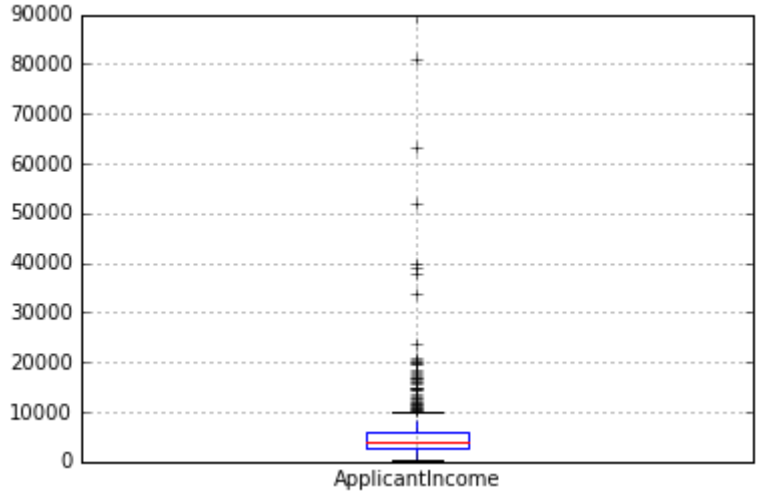
Lets start by plotting the histogram of ApplicantIncome using the following commands:

```
df['ApplicantIncome'].hist(bins=50)
```



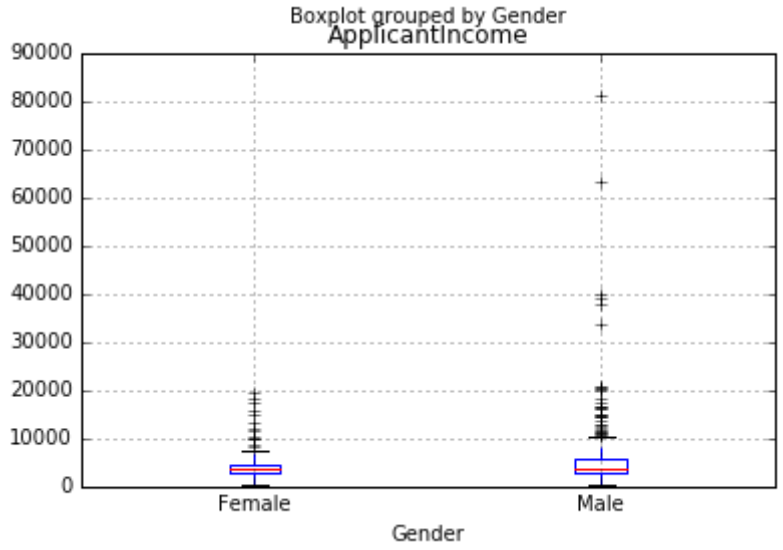
Here we observe that there are few extreme values. This is also the reason why 50 bins are required to depict the distribution clearly. Next, we look at box plots to understand the distributions. Box plot for fare can be plotted by:

```
df.boxplot(column='ApplicantIncome')
```



This confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society. Part of this can be driven by the fact that we are looking at people with different education levels. Let us segregate them by Education:

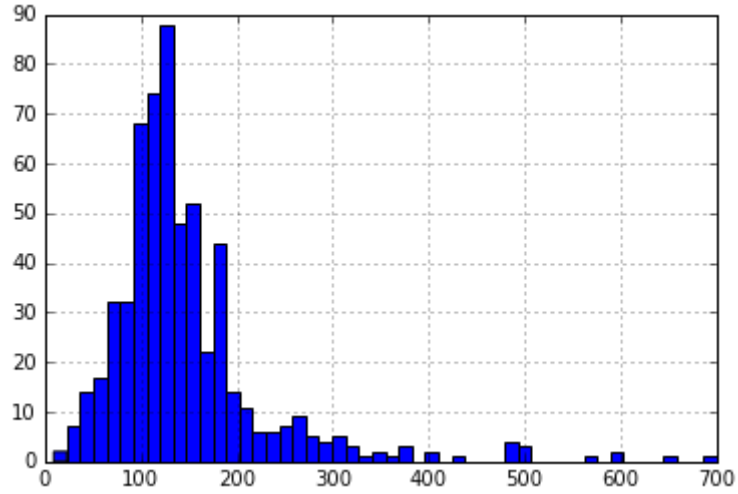
```
df.boxplot(column='ApplicantIncome', by = 'Education')
```



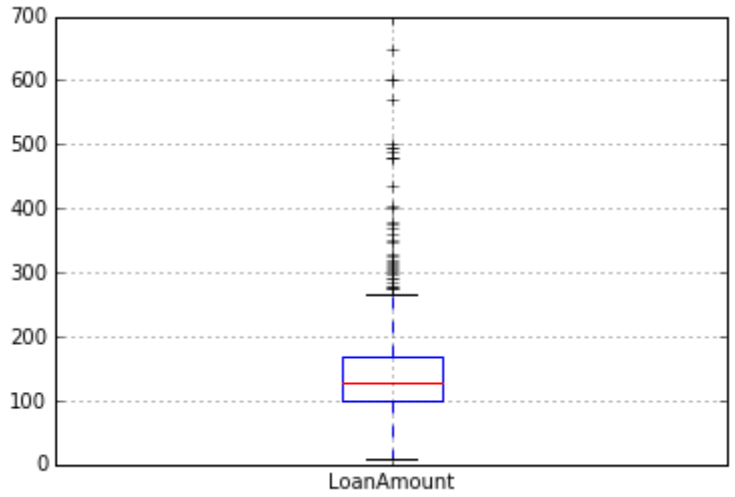
We can see that there is no substantial different between the mean income of graduate and non-graduates. But there are a higher number of graduates with very high incomes, which are appearing to be the outliers.

Now, Let's look at the histogram and boxplot of LoanAmount using the following command:

```
df['LoanAmount'].hist(bins=50)
```



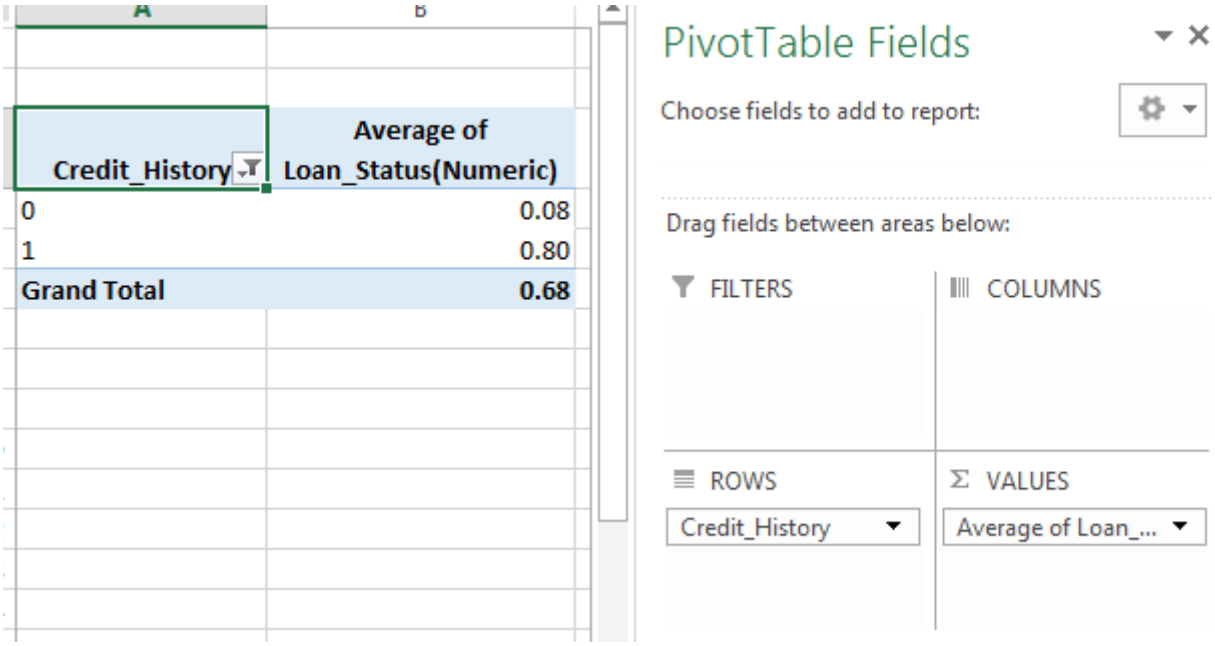

```
df.boxplot(column='LoanAmount')
```



Again, there are some extreme values. Clearly, both ApplicantIncome and LoanAmount require some amount of data munging. LoanAmount has missing and well as extreme values values, while ApplicantIncome has a few extreme values, which demand deeper understanding. We will take this up in coming sections.

Categorical variable analysis

Now that we understand distributions for ApplicantIncome and LoanIncome, let us understand categorical variables in more details. We will use Excel style pivot table and cross-tabulation. For instance, let us look at the chances of getting a loan based on credit history. This can be achieved in MS Excel using a pivot table as:



Credit_History	Average of Loan_Status(Numeric)
0	0.08
1	0.80
Grand Total	0.68

Note: here loan status has been coded as 1 for Yes and 0 for No. So the mean represents the probability of getting loan.

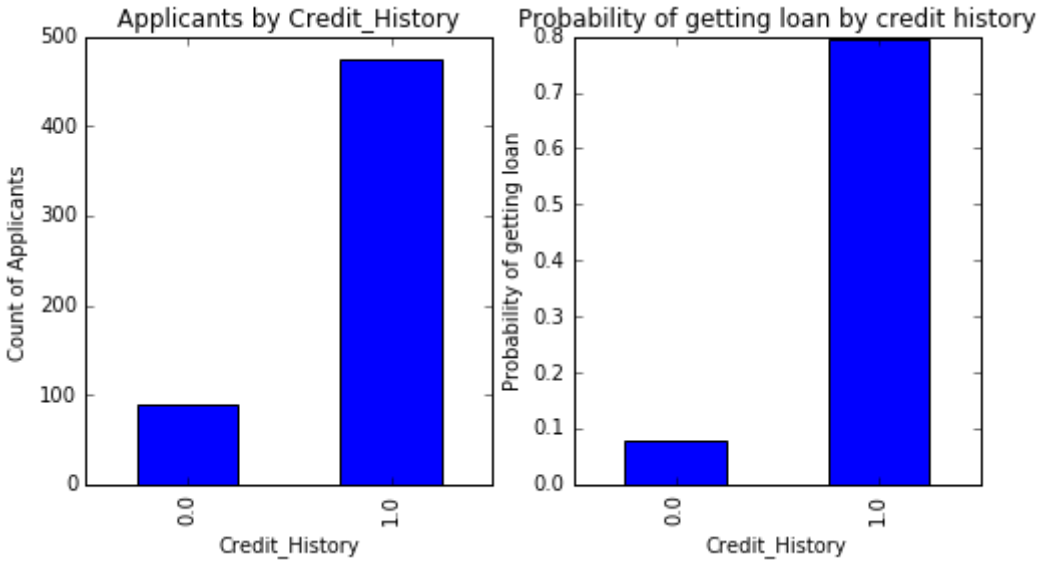
Now we will look at the steps required to generate a similar insight using Python. Please refer to this article for getting a hang of the different data manipulation techniques in Pandas.

```
temp1 = df['Credit_History'].value_counts(ascending=True)
temp2 = df.pivot_table(values='Loan_Status',index=
['Credit_History'],aggfunc=lambda x: x.map({'Y':1,'N':0}).mean())
print 'Frequency Table for Credit History:'
print temp1

print '\nProbility of getting loan for each Credit History class:'
print temp2
```

```
Frequency Table for Credit History:
0     89
1    475
Name: Credit_History, dtype: int64

Probility of getting loan for each Credit History class:
Credit_History
0    0.078652
1    0.795789
Name: Loan_Status, dtype: float64
```

Now we can observe that we get a similar pivot_table like the MS Excel one. This can be plotted as a bar chart using the "matplotlib" library with following code:

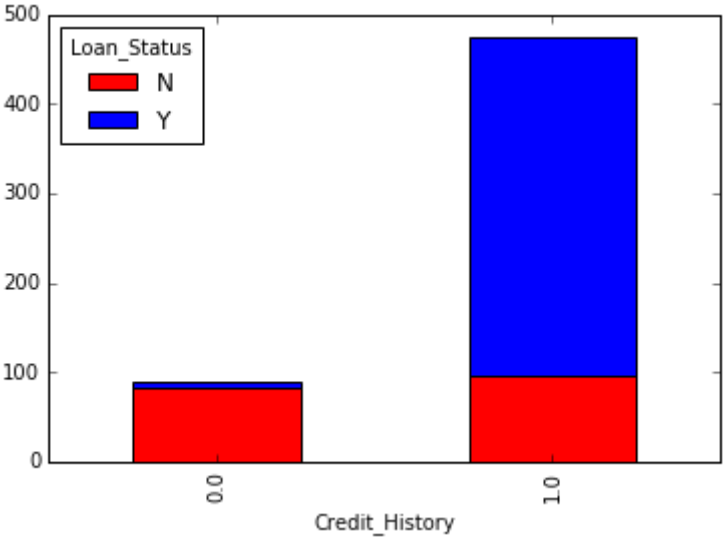
```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax1.set_xlabel('Credit_History')
ax1.set_ylabel('Count of Applicants')
ax1.set_title("Applicants by Credit_History")
temp1.plot(kind='bar')

ax2 = fig.add_subplot(122)
temp2.plot(kind = 'bar')
ax2.set_xlabel('Credit_History')
ax2.set_ylabel('Probability of getting loan')
ax2.set_title("Probability of getting loan by credit history")
```



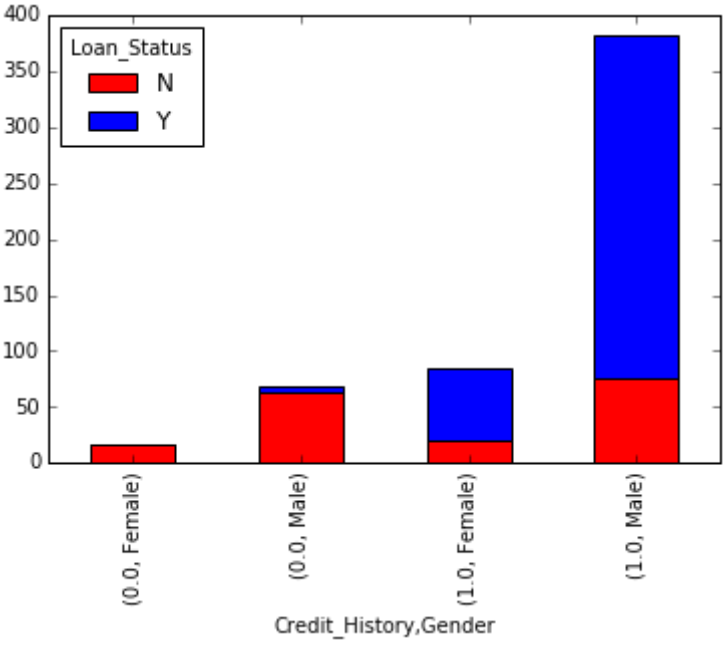
This shows that the chances of getting a loan are eight-fold if the applicant has a valid credit history. You can plot similar graphs by Married, Self-Employed, Property_Area, etc.

Alternately, these two plots can also be visualized by combining them in a stacked chart::

```
temp3 = pd.crosstab(df['Credit_History'], df['Loan_Status'])
temp3.plot(kind='bar', stacked=True, color=['red','blue'], grid=False)
```



You can also add gender into the mix (similar to the pivot table in Excel):



If you have not realized already, we have just created two basic classification algorithms here, one based on credit history, while other on 2 categorical variables (including gender). You can quickly code this to create your first submission on AV Datahacks.

We just saw how we can do exploratory analysis in Python using Pandas. I hope your love for pandas (the animal) would have increased by now - given the amount of help, the library can provide you in analyzing datasets.

Next let's explore ApplicantIncome and LoanStatus variables further, perform data munging and create a dataset for applying various modeling techniques. I would strongly urge that you take another dataset and problem and go through an independent example before reading further.

Data Munging in Python : Using Pandas

For those, who have been following, here are your must wear shoes to start running.

Data munging - recap of the need

While our exploration of the data, we found a few problems in the data set, which needs to be solved before the data is ready for a good model. This exercise is typically referred as "Data Munging". Here are the problems, we are already aware of:

1. There are missing values in some variables. We should estimate those values wisely depending on the amount of missing values and the expected importance of variables.
2. While looking at the distributions, we saw that ApplicantIncome and LoanAmount seemed to contain extreme values at either end. Though they might make intuitive sense, but should be treated appropriately.

In addition to these problems with numerical fields, we should also look at the non-numerical fields i.e. Gender, Property_Area, Married, Education and Dependents to see, if they contain any useful information.

If you are new to Pandas, I would recommend reading this article before moving on. It details some useful techniques of data manipulation.

Check missing values in the dataset

Let us look at missing values in all the variables because most of the models don't work with missing data and even if they do, imputing them helps more often than not. So, let us check the number of nulls / NaNs in the dataset

```
df.apply(lambda x: sum(x.isnull()),axis=0)
```

This command should tell us the number of missing values in each column as isnull() returns 1, if the value is null.

```
In [14]: df.apply(lambda x: sum(x.isnull()),axis=0)

Out[14]: 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
```

Though the missing values are not very high in number, but many variables have them and each one of these should be estimated and added in the data. Get a detailed view on different imputation techniques through this article.

Note: Remember that missing values may not always be NaNs. For instance, if the Loan_Amount_Term is 0, does it makes sense or would you consider that missing? I suppose your answer is missing and you're right. So we should check for values which are unpractical.

How to fill missing values in LoanAmount?

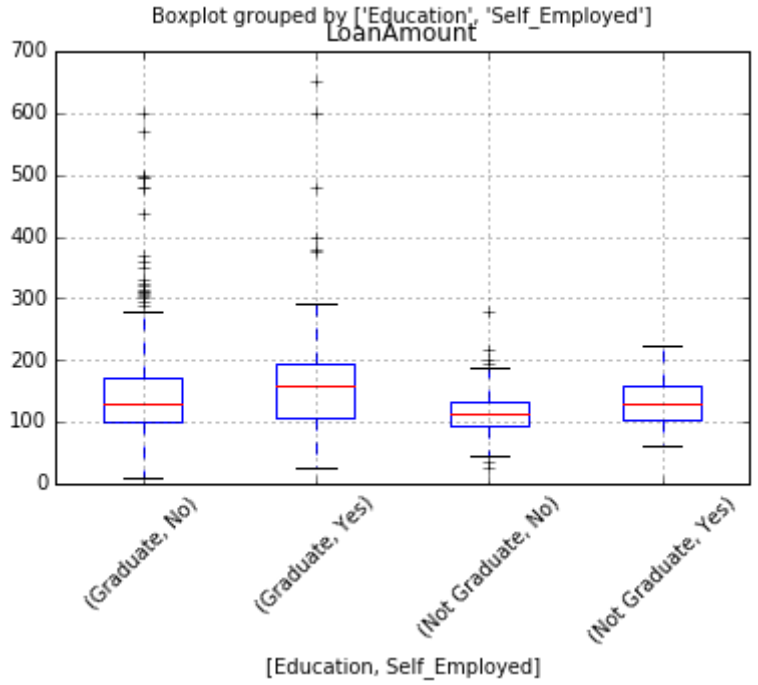
There are numerous ways to fill the missing values of loan amount - the simplest being replacement by mean, which can be done by following code:

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
```

The other extreme could be to build a supervised learning model to predict loan amount on the basis of other variables and then use age along with other variables to predict survival.

Since, the purpose now is to bring out the steps in data munging, I'll rather take an approach, which lies some where in between these 2 extremes. A key hypothesis is that the whether a person is educated or self-employed can combine to give a good estimate of loan amount.

First, let's look at the boxplot to see if a trend exists:



Thus we see some variations in the median of loan amount for each group and this can be used to impute the values. But first, we have to ensure that each of Self_Employed and Education variables should not have a missing values.

As we say earlier, Self_Employed has some missing values. Let's look at the frequency table:

```
In [40]: df['Self_Employed'].value_counts()

Out[40]: No     500
         Yes     82
         Name: Self_Employed, dtype: int64
```

Since ~86% values are "No", it is safe to impute the missing values as "No" as there is a high probability of success. This can be done using the following code:

```
df['Self_Employed'].fillna('No',inplace=True)
```

Now, we will create a Pivot table, which provides us median values for all the groups of unique values of Self_Employed and Education features. Next, we define a function, which returns the values of these cells and apply it to fill the missing values of loan amount:

```
table = df.pivot_table(values='LoanAmount', index='Self_Employed'
,columns='Education', aggfunc=np.median)
# Define function to return value of this pivot_table
def fage(x):
 return table.loc[x['Self_Employed'],x['Education']]
# Replace missing values
df['LoanAmount'].fillna(df[df['LoanAmount'].isnull()].apply(fage,
axis=1), inplace=True)
```

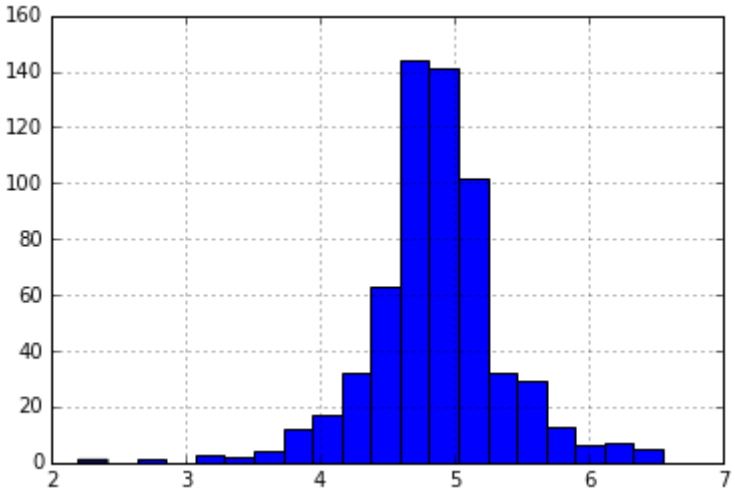
This should provide you a good way to impute missing values of loan amount.

How to treat for extreme values in distribution of LoanAmount and ApplicantIncome?

Let's analyze LoanAmount first. Since the extreme values are practically possible, i.e. some people might apply for high value loans due to specific needs. So instead of treating them as outliers, let's try a log transformation to nullify their effect:

```
 df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```

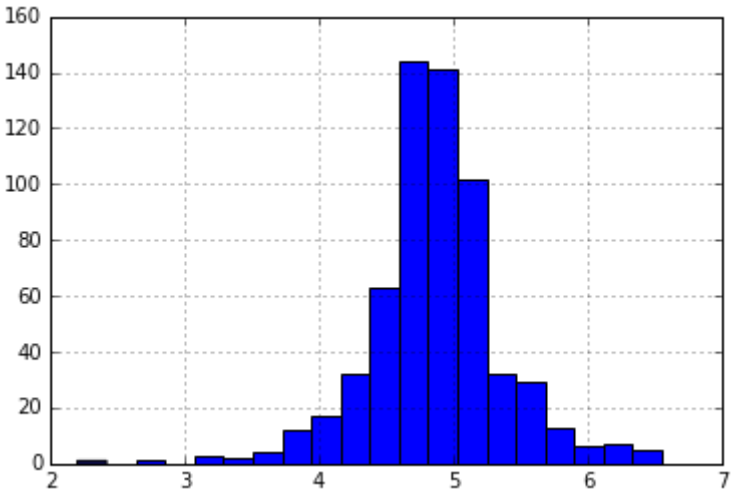
Looking at the histogram again:



Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

Coming to ApplicantIncome. One intuition can be that some applicants have lower income but strong support Co-applicants. So it might be a good idea to combine both incomes as total income and take a log transformation of the same.

```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log'] = np.log(df['TotalIncome'])
df['LoanAmount_log'].hist(bins=20)
```



Now we see that the distribution is much better than before. I will leave it upto you to impute the missing values for Gender, Married, Dependents, Loan_Amount_Term, Credit_History. Also, I encourage you to think about possible additional information which can be derived from the data. For example, creating a column for LoanAmount/TotalIncome might make sense as it gives an idea of how well the applicant is suited to pay back his loan.