Loan Prediction

1 Loan Prediction

1.1 Problem

- A Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers. Here they have provided a data set.

1.2 Data

- 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)

- Rows: 615
- Source: Datahack
- Jupyter Notebook

In [45]:
```
# Importing Library import pandas
as pd
import numpy as np from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder

# Reading the training dataset in a dataframe using Pandas df =
pd.read_csv("train.csv")

# Reading the test dataset in a dataframe using Pandas
```

```
test = pd.read_csv("test.csv")
```

In [48]: # First 10 Rows of training Dataset

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

```
Out[48]:     Loan_ID Gender Married Dependents     Education Self_Employed \
1     LP001002 Male     No     0     Graduate     No
2     LP001003 Male     Yes     1     Graduate     No
3     LP001005 Male Yes 0 Graduate Yes 3 LP001006 Male Yes 0 Not Graduate No
4     LP001008 Male     No     0     Graduate     No
5     LP001011 Male Yes 2 Graduate Yes 6 LP001013 Male Yes 0 Not Graduate No
      7  LP001014 Male  Yes  3+  Graduate  No
      8  LP001018 Male  Yes  2   Graduate  No
      9  LP001020 Male  Yes  1   Graduate  No
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term \
0	5849	0.0	NaN	360.0
1	4583	1508.0	128.0	360.0
2	3000	0.0	66.0	360.0
3	2583	2358.0	120.0	360.0
4	6000	0.0	141.0	360.0
5	5417	4196.0	267.0	360.0
6	2333	1516.0	95.0	360.0
7	3036	2504.0	158.0	360.0
8	4006	1526.0	168.0	360.0
9	12841	10968.0	349.0	360.0

```
  Credit_History Property_Area Loan_Status
0   1.0   Urban   Y 1   1.0   Rural   N 2
1.0  Urban   Y 3   1.0   Urban   Y 4   1.0
Urban     Y 5   1.0   Urban   Y 6   1.0
Urban     Y
7         0.0Semiurban     N
8         1.0Urban   Y
9         1.0Semiurban     N
```

In [206]: # Store total number of observation in training dataset df_length =len(df)

```
# Store total number of columns in testing data set test_col =
len(test.columns)
```

2 Understanding the various features (columns) of the dataset.

In [50]: # Summary of numerical variables for training data set

```
df.describe()
```

Out[50]:

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

	Credit_History
count	564.000000
mean	0.842199
std	0.364878
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

1. 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.

In [51]: # Get the unique values and their frequency of variable Property_Area

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

Out[51]:

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

2. Understanding Distribution of Numerical Variables

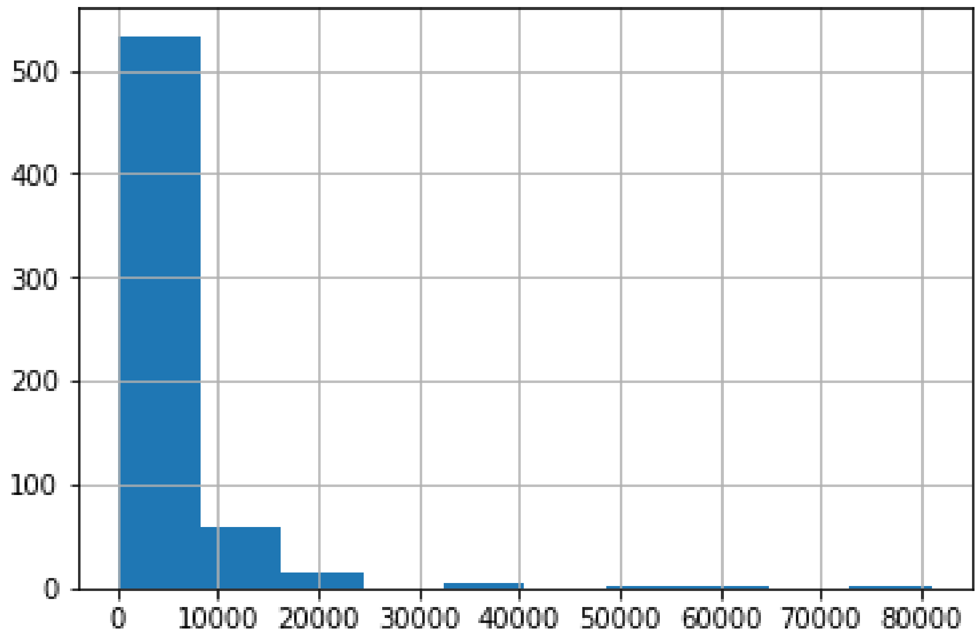
- ApplicantIncome
- LoanAmount

In [53]: # Box Plot for understanding the distributions and to observe the outliers.

```
%matplotlib inline

# Histogram of variable ApplicantIncome df['ApplicantIncome'].hist()
```

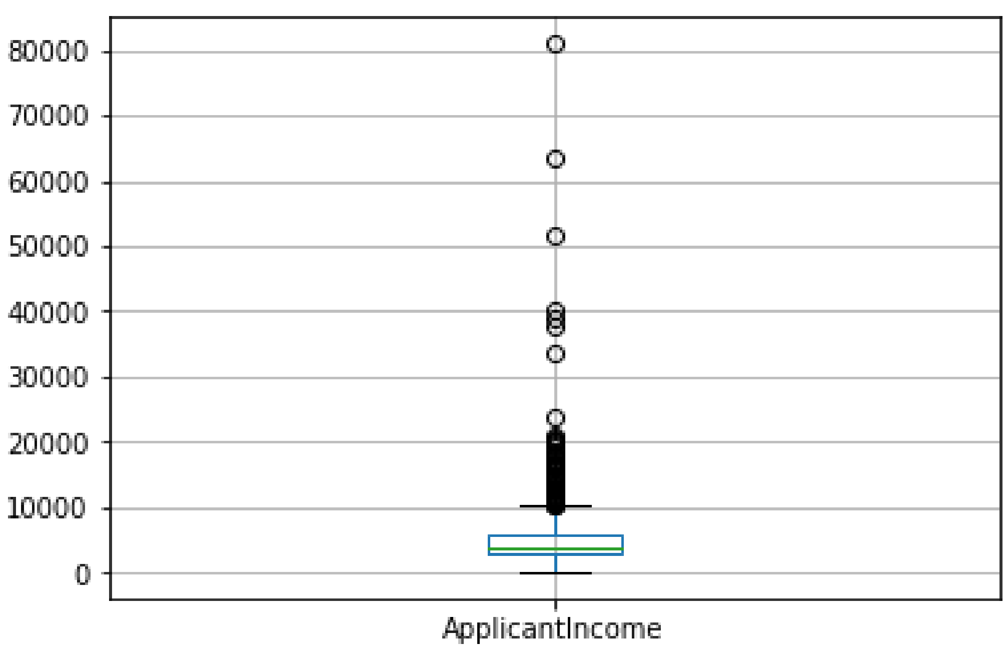
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc932780>



In [54]: # Box Plot for variable ApplicantIncome of training data set

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

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc85e278>

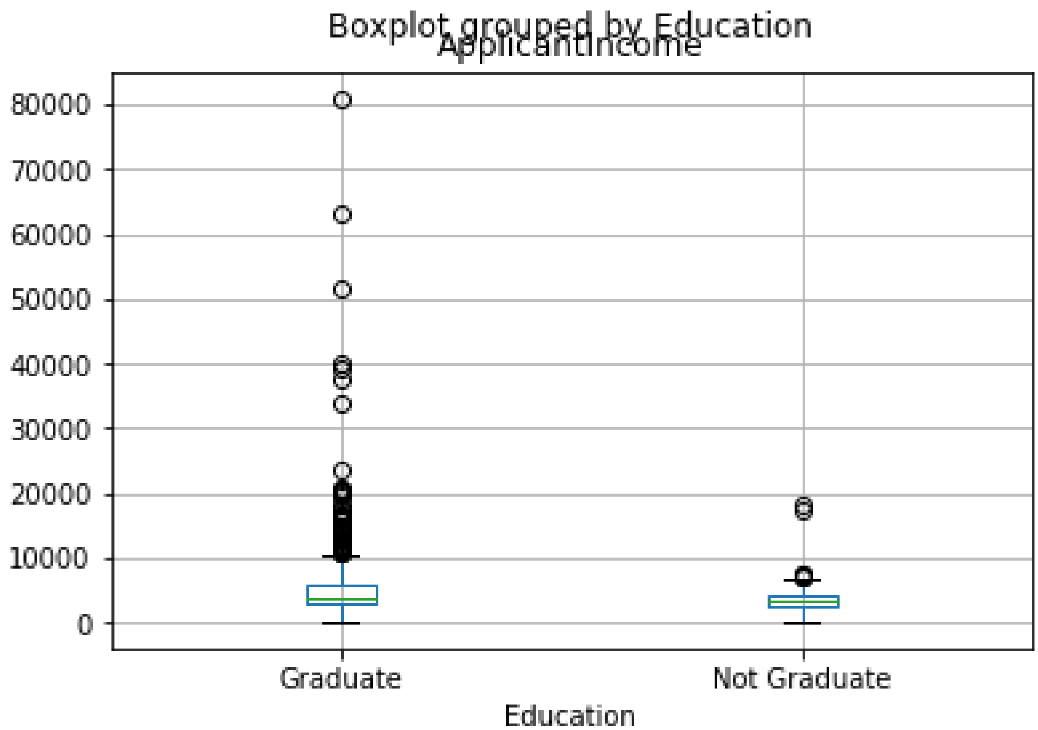


3. The above Box Plot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society.

In [55]: # Box Plot for variable ApplicantIncome by variable Education of training data set

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

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc82e588>

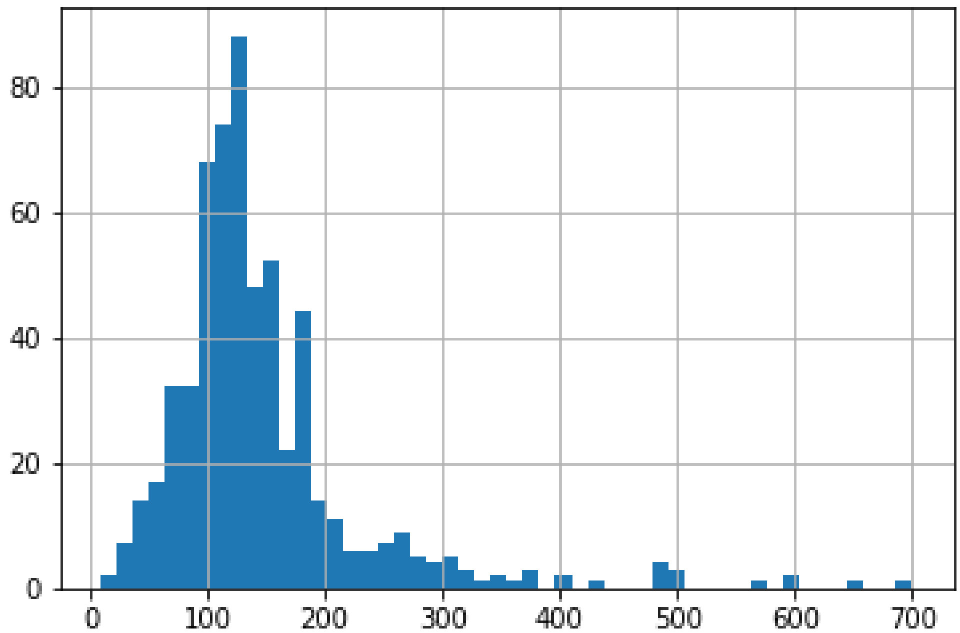


4. We can see that there is no substantial different between the mean income of graduate and non-graduates. But there are a higher number of graduates with very high incomes, which are appearing to be the outliers

In [56]: # Histogram of variable LoanAmount

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

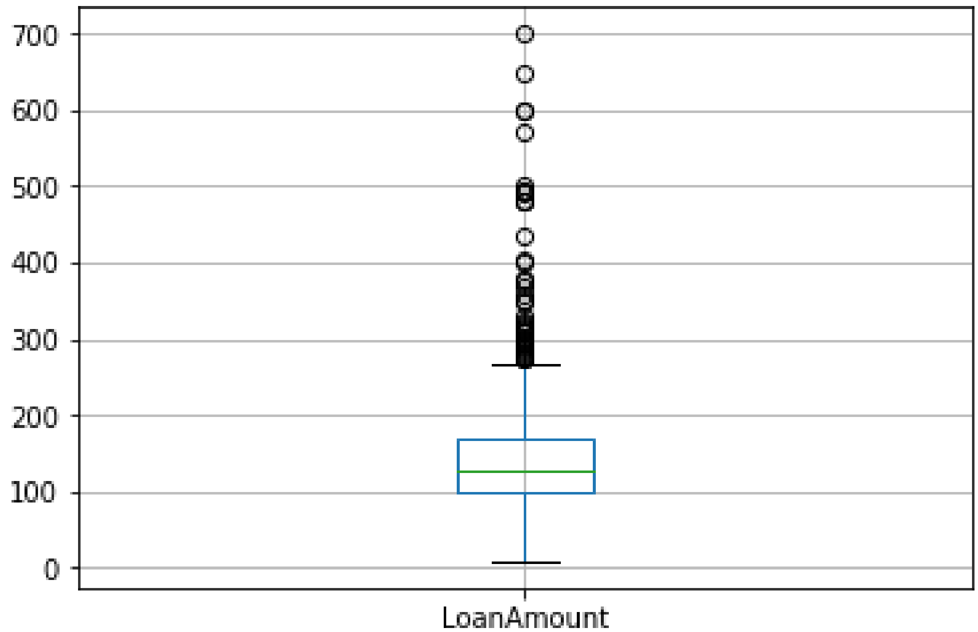
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc73e2e8>



In [57]: # Box Plot for variable LoanAmount of training data set

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

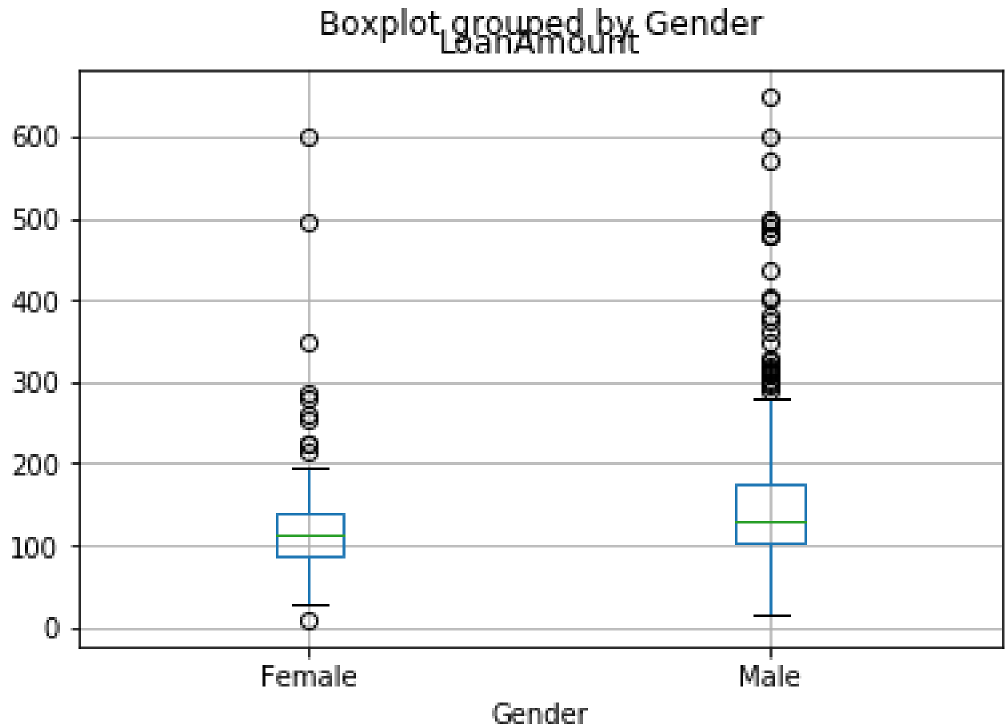
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc728be0>



In [58]: # Box Plot for variable LoanAmount by variable Gender of training data set

```
df.boxplot(column='LoanAmount', by = 'Gender')
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bc79acc0>



5. LoanAmount has missing as well as extreme values, while ApplicantIncome has a few extreme values.

3 Understanding Distribution of Categorical Variables

In [15]: # Loan approval rates in absolute numbers

```
loan_approval = df['Loan_Status'].value_counts()['Y'] print(loan_approval)
```

422

- 422 number of loans were approved.

In [37]: # Credit History and Loan Status pd.crosstab(df ['Credit_History'], df ['Loan_Status'], margins=True)

Out[37]: Loan_Status N Y All

Credit_History	N	Y	All
0.0	82	7	89
1.0	97	378	475
All	179	385	564

In [204]:
```
#Function to output percentage row wise in a cross table def
percentageConvert(ser):
    return ser/float(ser[-1])

# Loan approval rate for customers having Credit_History (1)
df=pd.crosstab(df ["Credit_History"], df ["Loan_Status"], margins=True).apply(percenta
loan_approval_with_Credit_1 = df['Y'][1] print(loan_approval_with_Credit_1*100)
```

79.04761904761905

- 79.58 % of the applicants whose loans were approved have Credit_History equals to 1.

In [39]:
```
df['Y']
```

Out[39]: Credit_History
0.0 0.078652 1.0
0.795789 All 0.682624
Name: Y, dtype: float64

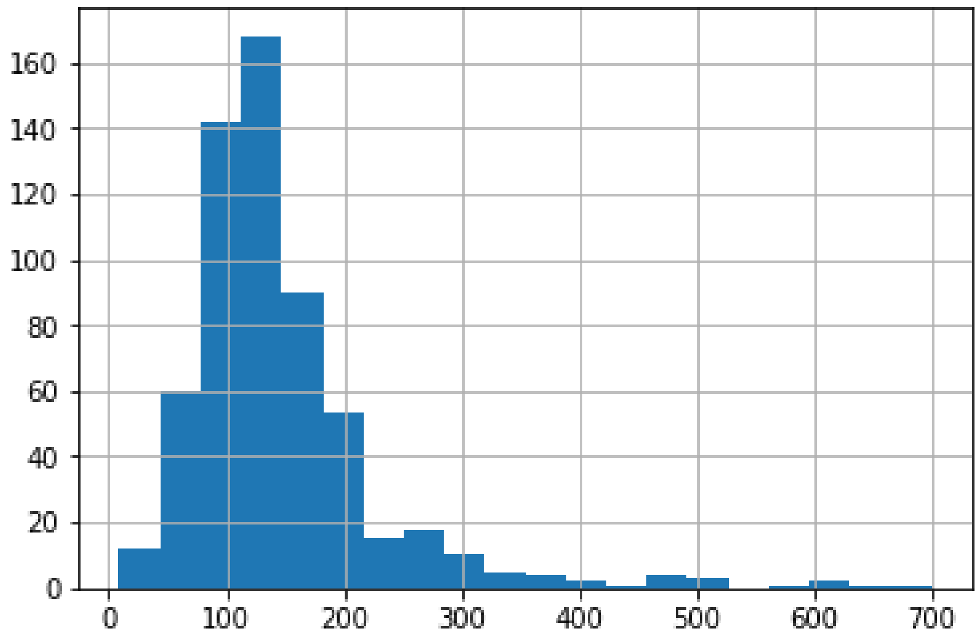
In [591]:
```
# Replace missing value of Self_Employed with more frequent category
df['Self_Employed'].fillna('No',inplace=True)
```

4 Outliers of LoanAmount and Applicant Income

In [588]:
```
# Add both ApplicantIncome and CoapplicantIncome to TotalIncome df['TotalIncome'] =
df['ApplicantIncome'] + df['CoapplicantIncome']

# Looking at the distribtion of TotalIncome df['LoanAmount'].hist(bins=20)
```

Out[588]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6fadc7ff98>

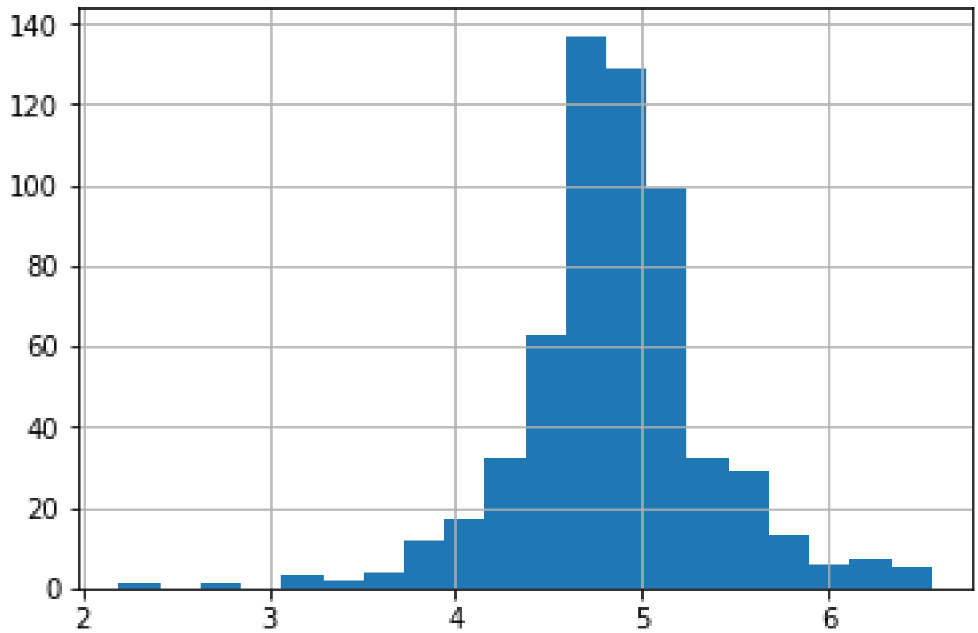


• The extreme values are practically possible, i.e. some people might apply for high value loans due to specific needs. So instead of treating them as outliers, let's try a log transformation to nullify their effect:

In [112]:
```
# Perform log transformation of TotalIncome to make it closer to normal df['LoanAmount_log'] =
np.log(df['LoanAmount'])

# Looking at the distribtion of TotalIncome_log df['LoanAmount_log'].hist(bins=20)
```

Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bbececc50>



5 Data Preparation for Model Building

- sklearn requires all inputs to be numeric, we should convert all our categorical variables into numeric by encoding the categories. Before that we will fill all the missing values in the dataset.

In [62]:
```
# Impute missing values for Gender
df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)

# Impute missing values for Married df['Married'].fillna(df['Married'].mode()[0],inplace=True)

# Impute missing values for Dependents
df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)

# Impute missing values for Credit_History
df['Credit_History'].fillna(df['Credit_History'].mode()[0],inplace=True)

# Convert all non-numeric values to number
cat=['Gender','Married','Dependents','Education','Self_Employed','Credit_History','Prop

for var in cat:
    le = preprocessing.LabelEncoder()
    df[var]=le.fit_transform(df[var].astype('str'))
```

```
df.dtypes
```

Out[62]:

```
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         int64
Property_Area          int64
Loan_Status dtype:    object
object
```

6 Generic Classification Function

In [208]:

```
#Import models from scikit learn module: from sklearn
import metrics
from sklearn.cross_validation import KFold

                #Generic function for making a classification model and accessing performance:

def classification_model(model, data, predictors, outcome):
    #Fit the model: model.fit(data[predictors],data[outcome])

    #Make predictions on training set: predictions =
    model.predict(data[predictors])

    #Print accuracy
    accuracy = metrics.accuracy_score(predictions,data[outcome]) print ("Accuracy :
    %s" % "{0:.3%}".format(accuracy))

    #Perform k-fold cross-validation with 5 folds kf =
    KFold(data.shape[0], n_folds=5)
    error = [] for train, test in kf: # Filter training data train_predictors =
    (data[predictors].iloc[train,:])

        # The target we re using to train the algorithm. train_target =
        data[outcome].iloc[train]

        # Training the algorithm using the predictors and target.
        model.fit(train_predictors, train_target)
        #Record error from each cross-validation run
```

In [186]: #Combining both train and test dataset

```
#Create a flag for Train and Test Data set
df['Type']='Train' test['Type']='Test'
fullData = pd.concat([df,test],axis=0, sort=True)

#Look at the available missing values in the dataset fullData.isnull().sum()
```

Out[186]:

```
ApplicantIncome          0
CoapplicantIncome        0
Credit_History          29
Dependents              10
Education                0
Gender                  11
LoanAmount              27
LoanAmount_log         389
Loan_Amount_Term        20
Loan_ID                  0
Loan_Status            367
Married                  0
Property_Area            0
Self_Employed           23
Type dtype:              0
int64
```

In [187]: #Identify categorical and continuous variables

```
ID_col = ['Loan_ID'] target_col =
["Loan_Status"]
        cat_cols = ['Credit_History','Dependents','Gender','Married','Education','Property_Are
```

In [200]: #Imputing Missing values with mean for continuous variable

```
fullData['LoanAmount'].fillna(fullData['LoanAmount'].mean(), inplace=True)
fullData['LoanAmount_log'].fillna(fullData['LoanAmount_log'].mean(), inplace=True)
fullData['Loan_Amount_Term'].fillna(fullData['Loan_Amount_Term'].mean(), inplace=True)
fullData['ApplicantIncome'].fillna(fullData['ApplicantIncome'].mean(), inplace=True)
fullData['CoapplicantIncome'].fillna(fullData['CoapplicantIncome'].mean(), inplace=Tru
        error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[tes print

    ("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))

    #Fit the model again so that it can be refered outside the function:
    model.fit(data[predictors],data[outcome])
```

7 Model Building

```
#Imputing Missing values with mode for categorical variables
fullData['Gender'].fillna(fullData['Gender'].mode()[0], inplace=True)
fullData['Married'].fillna(fullData['Married'].mode()[0], inplace=True)
fullData['Dependents'].fillna(fullData['Dependents'].mode()[0], inplace=True)
fullData['Loan_Amount_Term'].fillna(fullData['Loan_Amount_Term'].mode()[0], inplace=Tr
fullData['Credit_History'].fillna(fullData['Credit_History'].mode()[0], inplace=True)
```

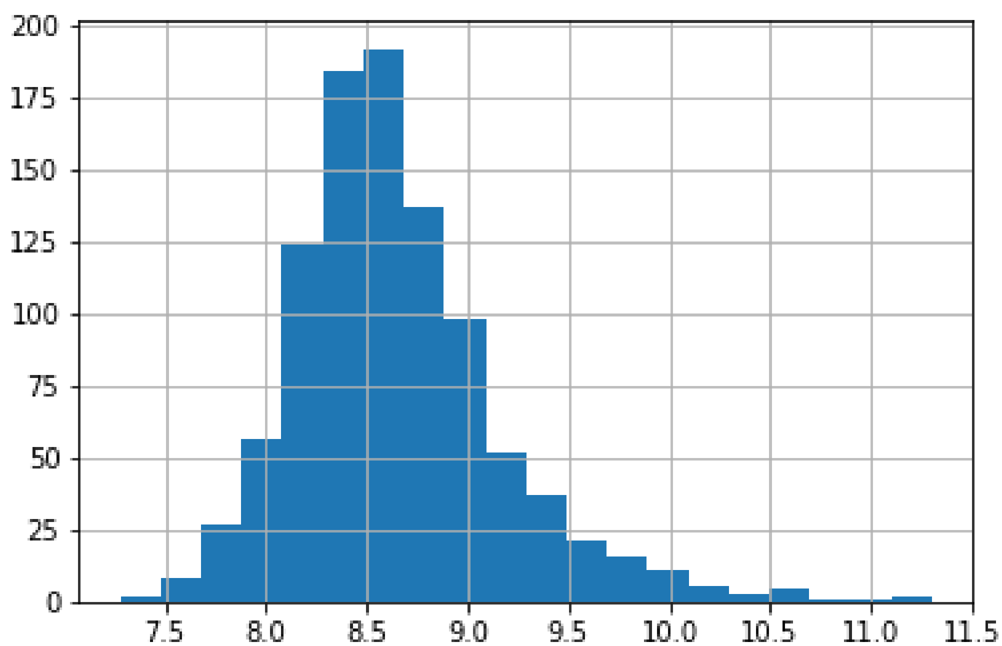
In [202]: #Create a new column as Total Income

```
fullData['TotalIncome']=fullData['ApplicantIncome'] + fullData['CoapplicantIncome']

fullData['TotalIncome_log'] = np.log(fullData['TotalIncome'])

#Histogram for Total Income fullData['TotalIncome_log'].hist(bins=20)
```

Out[202]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93bbd93a20>



In [197]: #create label encoders for categorical features for var in cat_cols:

```
    number = LabelEncoder()
    fullData[var] = number.fit_transform(fullData[var].astype('str'))

train_modified=fullData[fullData['Type']=='Train'] test_modified=fullData[fullData['Type']=='Test']
```

```
train_modified["Loan_Status"] = number.fit_transform(train_modified["Loan_Status"].ast
/home/parths007/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: SettingWithCopyWa A
value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer,col_indexer] =
value instead
```

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#

7.1 Logistic Regression Model

1. The chances of getting a loan will be higher for:

- Applicants having a credit history (we observed this in exploration.)
- Applicants with higher applicant and co-applicant incomes
- Applicants with higher education level
- Properties in urban areas with high growth perspectives

So let's make our model with 'Credit_History', 'Education' & 'Gender'

In [198]:
```
from sklearn.linear_model import LogisticRegression

predictors_Logistic=['Credit_History','Education','Gender']

x_train = train_modified[list(predictors_Logistic)].values y_train =

train_modified["Loan_Status"].values

x_test=test_modified[list(predictors_Logistic)].values
```

In [203]:
```
# Create logistic regression object model =
LogisticRegression()

# Train the model using the training sets model.fit(x_train, y_train)

#Predict Output predicted=
model.predict(x_test)

#Reverse encoding for predicted outcome predicted =
number.inverse_transform(predicted)

#Store it to test dataset
```

test_modified['Loan_Status']=predicted outcome_var = 'Loan_Status' classification_model(model,

df,predictors_Logistic,outcome_var)

test_modified.to_csv("Logistic_Prediction.csv",columns=['Loan_ID','Loan_Status']) Accuracy : 80.945%

Cross-Validation Score : 80.946%

/home/parths007/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/label.py:151: Deprec if diff: /home/parths007/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:14: SettingWithCopyW A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#