# loanpredict

October 19, 2019

## 1 Loan Prediction Dataset

This dataset is a simplified version of the one available on Kaggle. Source: (https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/)

## 2 Problem:

Predict if a loan will get approved or not

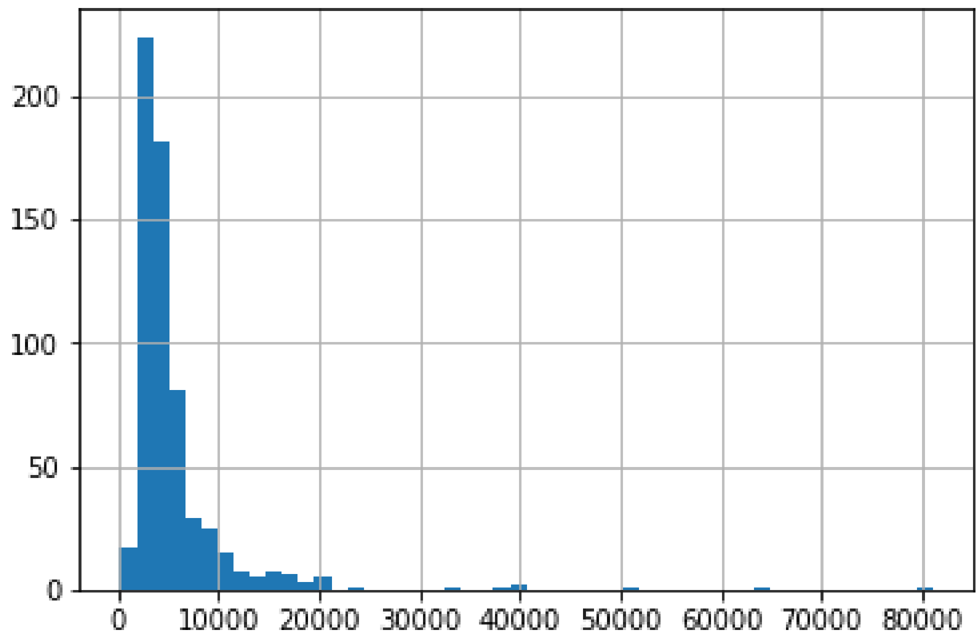
```
In [72]: import pandas as pd
         import matplotlib as plt
         import numpy as np


         #Load the data
         data_train = pd.read_csv("/Users/munirmalik/loan_predict/train.csv")
         data_test = pd.read_csv("/Users/munirmalik/loan_predict/test.csv")

         #data_train.head(20)

In [73]: data_train.describe()
```

```
Out[73]:        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
```

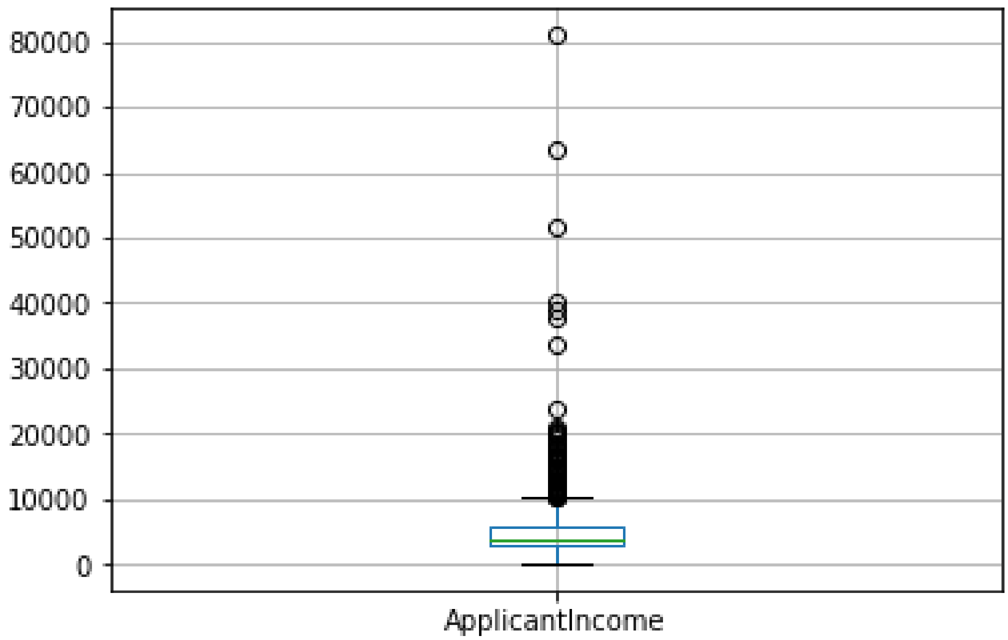
In [74]:
```
data_train['ApplicantIncome'].hist(bins=50)
```

Out[74]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16c45a90>
```

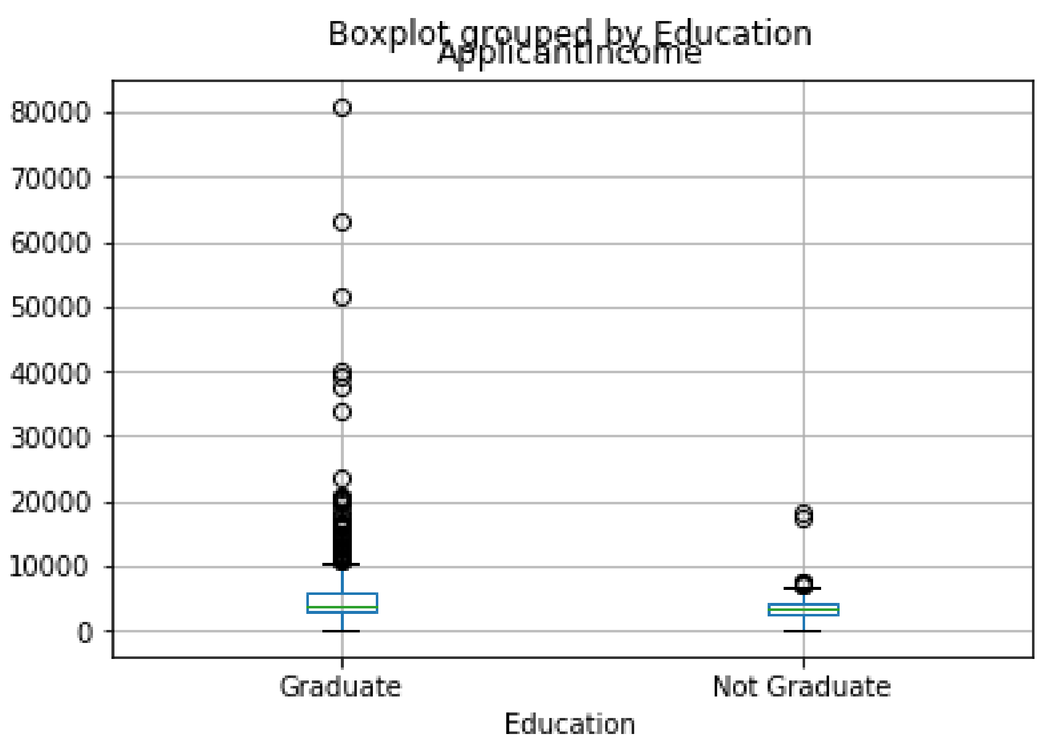



In [75]:
```
data_train.boxplot(column='ApplicantIncome')
```

Out[75]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a21991630>
```

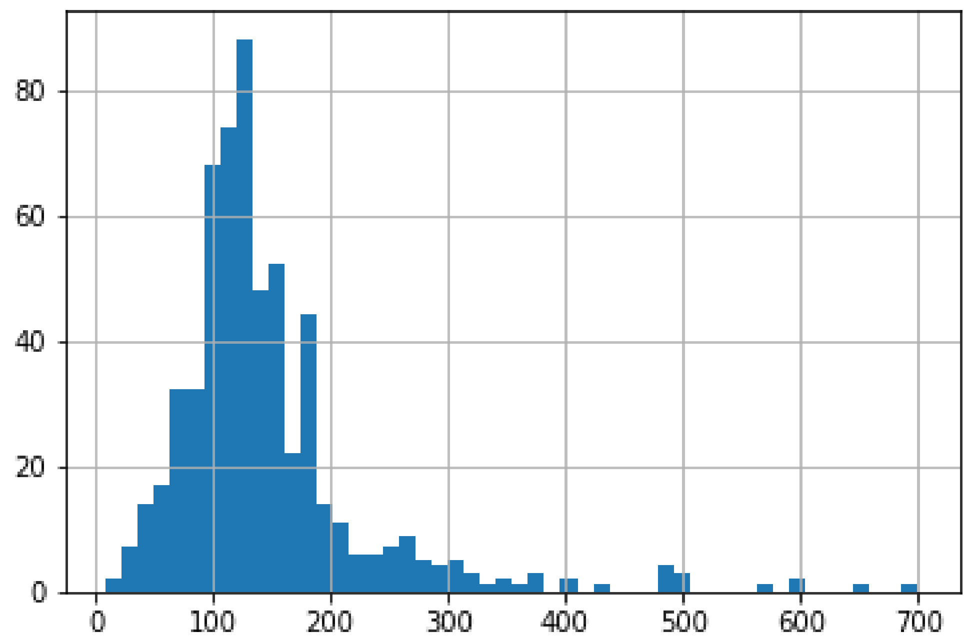



```
In [76]: data_train.boxplot(column='ApplicantIncome', by = 'Education')

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x1a21b23fd0>
```



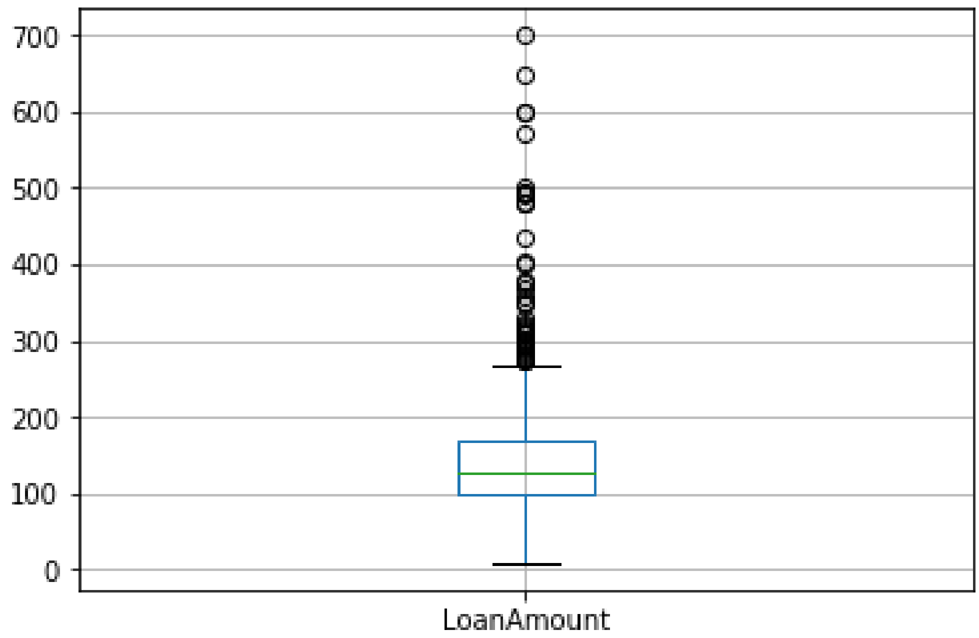
```
In [77]: data_train['LoanAmount'].hist(bins=50)
```

```
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x1a21befe80>
```




```
In [78]: data_train.boxplot(column="LoanAmount")
```

```
Out[78]: <matplotlib.axes._subplots.AxesSubplot at 0x1a21caec18>
```



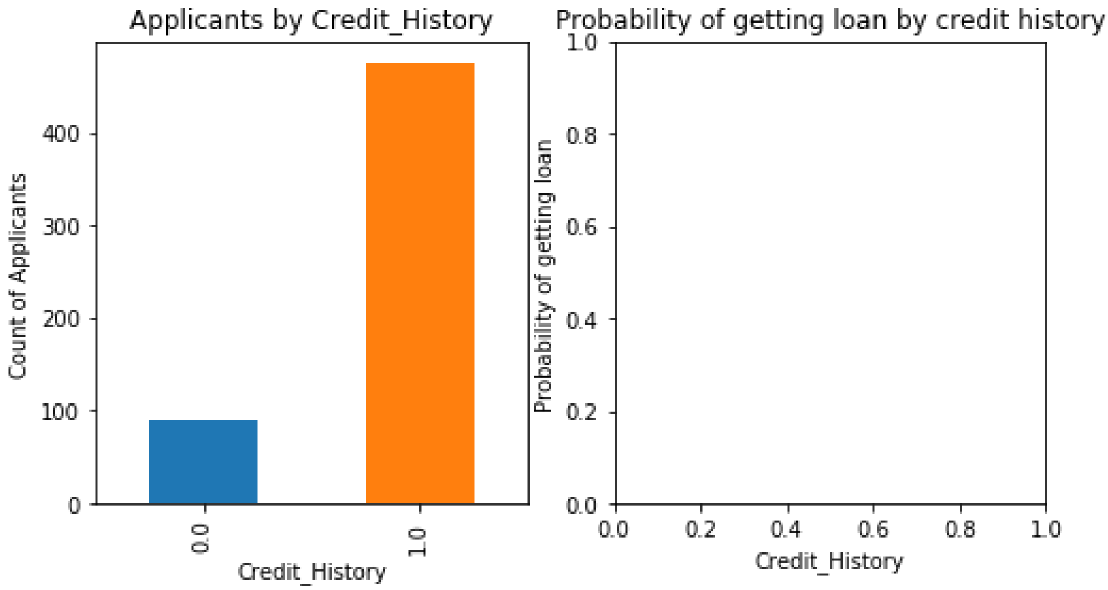

In [79]:
```
temp1 = data_train['Credit_History'].value_counts(ascending=True)
temp2 = data_train.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=l
print('Frequency Table for Credit History:')
print(temp1)

print('\nProbability of getting loan for each Credit History class')
print(temp2)
```

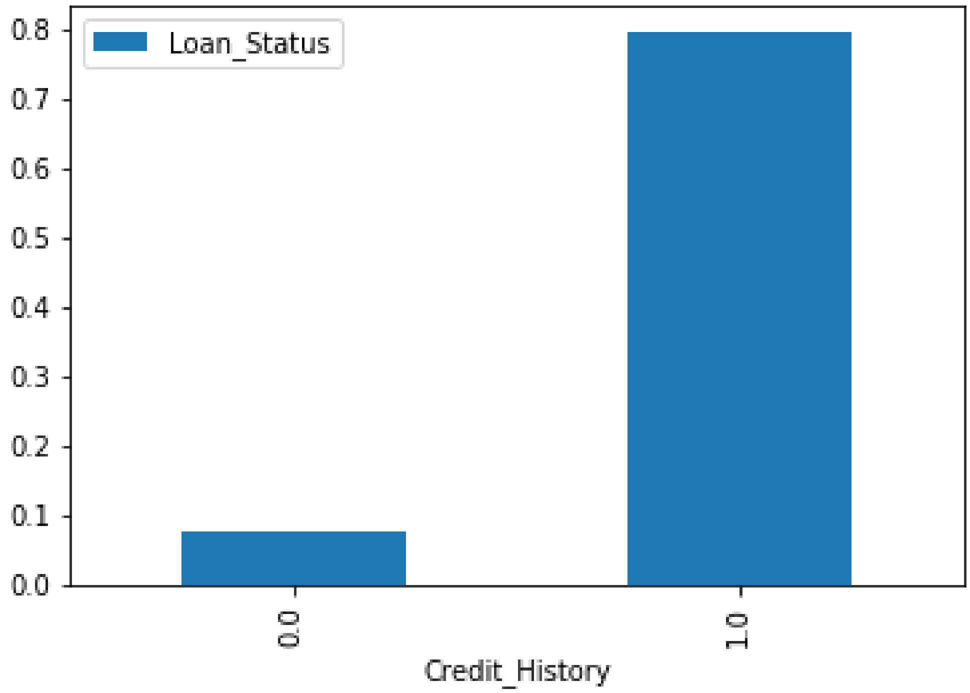
```
Frequency Table for Credit History:
0.0     89
1.0    475
Name: Credit_History, dtype: int64

Probability of getting loan for each Credit History class
                Loan_Status
Credit_History
0.0                0.078652
1.0                0.795789
```

In [80]:
```
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)
ax2.set_xlabel('Credit_History')
ax2.set_ylabel('Probability of getting loan')
ax2.set_title("Probability of getting loan by credit history")
temp2.plot(kind = 'bar')
```

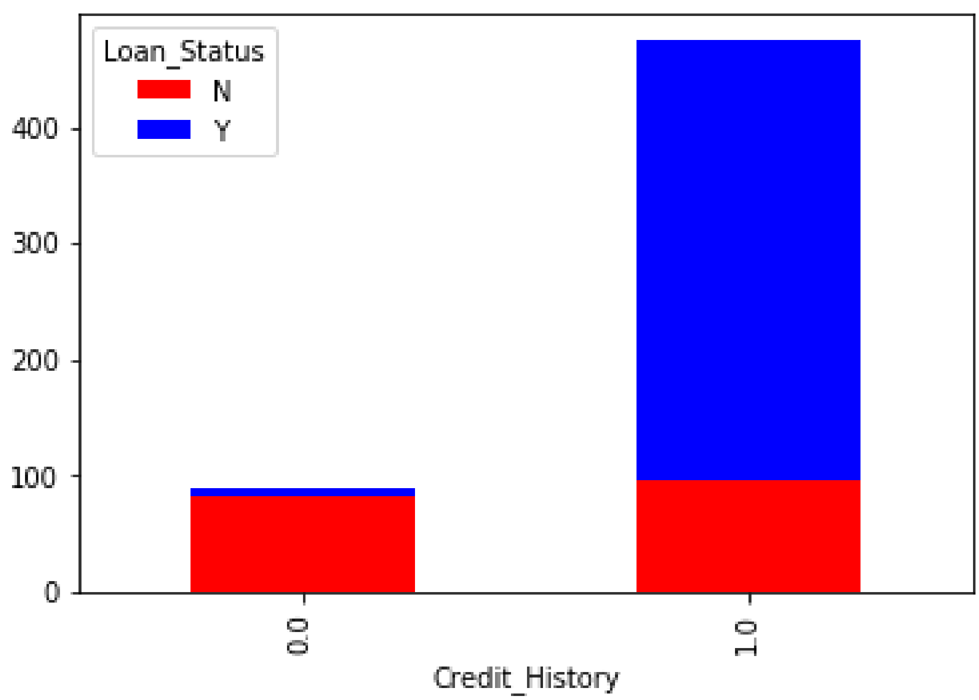
Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1a220457f0>






```
In [81]: temp3 = pd.crosstab(data_train['Credit_History'], data_train['Loan_Status'])
         temp3.plot(kind='bar', stacked=True, color=['red','blue'], grid=False)
```

```
Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2204c550>
```



In [82]:
```
#include code here that separates the above data by gender
```

In [83]:
```
data_train.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[83]:
```
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
```

In [84]:
```
data_train['Self_Employed'].value_counts()
```

Out[84]:
```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [85]:
```
# Filling in the missing values for Self_Employed column
# Since ~86% values are "No", it is safe to impute the missing values as "No"
data_train['Self_Employed'].fillna('No',inplace=True)
```

In [86]:
```
table = data_train.pivot_table(values='LoanAmount', index='Self_Employed', columns='Ed
# Define function to return value of this pivot_table
def fage(x):
    return table.loc[x['Self_Employed'],x['Education']]
# Replace missing values
data_train['LoanAmount'].fillna(data_train[data_train['LoanAmount'].isnull()].apply(fa
```

In [87]:
```
# Filling in the missing values for number of Dependents
# If this is left blank, it is safe to assume that the number of Dependents is 0
data_train['Dependents'].fillna(0,inplace=True)
```

In [88]:
```
# Filling in the missing values for Marital Status
# It is safe to assume that if it was left blank, then the individual is single
#data_train['Married'].fillna('No',inplace=True)
```

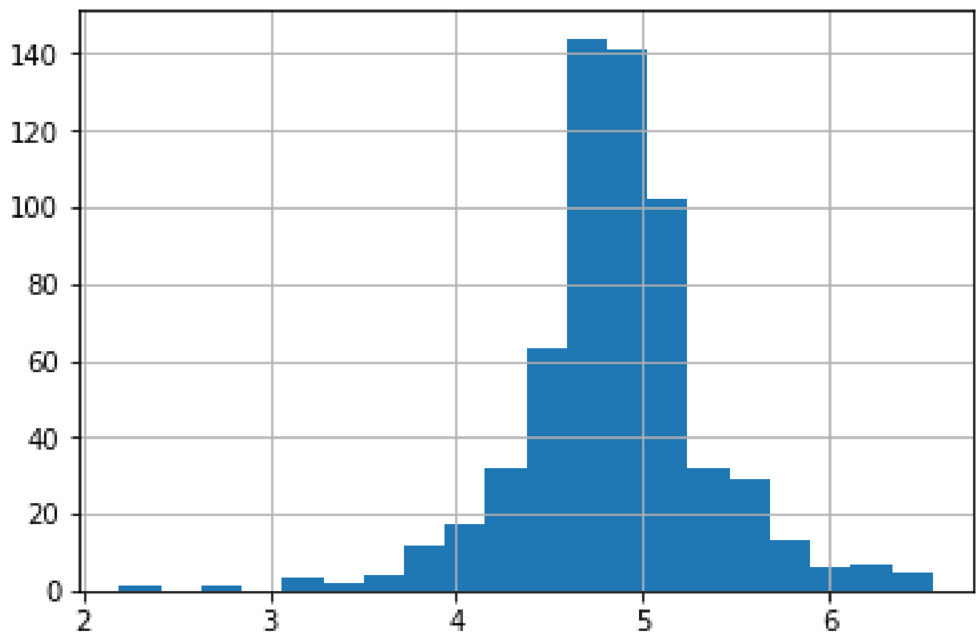
In [89]:
```
# Filling in the missing values for Credit_History
#
data_train['Credit_History'].fillna(0,inplace=True)
```

```
In [90]: data_train.apply(lambda x: sum(x.isnull()),axis=0)
```

```
Out[90]: Loan_ID               0
         Gender               13
         Married               3
         Dependents            0
         Education             0
         Self_Employed         0
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount            0
         Loan_Amount_Term     14
         Credit_History        0
         Property_Area         0
         Loan_Status           0
         dtype: int64
```

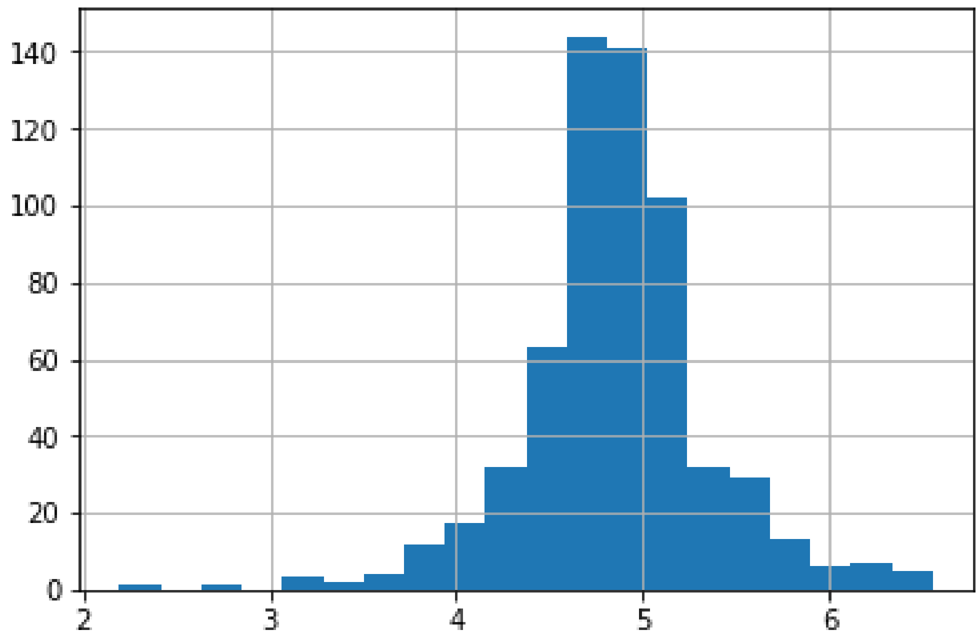
```
In [91]: data_train['LoanAmount_log'] = np.log(data_train['LoanAmount'])
         data_train['LoanAmount_log'].hist(bins=20)
```

```
Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20a139b0>
```




```
In [92]: data_train['TotalIncome'] = data_train['ApplicantIncome'] + data_train['CoapplicantInc
         data_train['TotalIncome_log'] = np.log(data_train['TotalIncome'])
         data_train['LoanAmount_log'].hist(bins=20)
```

```
Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2217e710>
```




## 3 Building a Predictive Model

Before we build the model, we have to first fill all the null values in the dataset, and then convert the categorical variables into numerical ones (since sklearn requires all inputs to be numeric)

In [93]:
```
# This bit essentially fills all the missing values with zero
# It's what I did before lol

data_train['Gender'].fillna(data_train['Gender'].mode()[0], inplace=True)
data_train['Married'].fillna(data_train['Married'].mode()[0], inplace=True)
data_train['Dependents'].fillna(data_train['Dependents'].mode()[0], inplace=True)
data_train['Loan_Amount_Term'].fillna(data_train['Loan_Amount_Term'].mode()[0], inpla
data_train['Credit_History'].fillna(data_train['Credit_History'].mode()[0], inplace=T

# This part is rather messy. I did this because some of the values were strings and s
data_train['Dependents'].replace(to_replace='3+', value=3,inplace=True)
data_train['Dependents'].replace(to_replace='0', value=0,inplace=True)
data_train['Dependents'].replace(to_replace='2', value=2,inplace=True)
data_train['Dependents'].replace(to_replace='1', value=1,inplace=True)
```

In [94]:
```
data_train['Dependents'].value_counts()
```

Out[94]:
```
0    360
1    102
2    101
```

```
3     51
Name: Dependents, dtype: int64
```

In [95]:
```python
from sklearn.preprocessing import LabelEncoder
var_mod = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_
le = LabelEncoder()
for i in var_mod:
    print(i)
    data_train[i].map(type).value_counts()
    data_train[i] = le.fit_transform(data_train[i])
data_train.dtypes
```

```
Gender
Married
Dependents
Education
Self_Employed
Property_Area
Loan_Status
```

Out[95]:
```
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_log       float64
TotalIncome          float64
TotalIncome_log      float64
dtype: object
```

In [96]:
```python
# Importing models from the scikit-learn module
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics

#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
      error.append(model.score(data[predictors].iloc[test,:], data[outcome].iloc[test]))

    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])
```

# 4 Logistic Regression

```
In [97]: outcome_var = 'Loan_Status'
         model = LogisticRegression()
         predictor_var = ['Credit_History']
         classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 77.036%
Cross-Validation Score : 77.041%
```

```
In [98]: # A different combination of variables:
         predictor_var = ['Credit_History','Education','Married','Self_Employed','Property_Area
         classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 77.036%
Cross-Validation Score : 77.041%
```

## 5 Decision Tree Classifier

In [99]:
```
model = DecisionTreeClassifier()
predictor_var = ['Credit_History','Gender','Married','Education']
classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 77.199%
Cross-Validation Score : 76.553%
```

In [100]:
```
# The credit history is so dominating, the categorical variables barely have an impa
# Therefore, we'll try a few numerical variables:
model = DecisionTreeClassifier()
predictor_var = ['Credit_History','Loan_Amount_Term','LoanAmount_log']
classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 88.599%
Cross-Validation Score : 64.653%
```

## 6 Random Forest

In [101]:
```
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender','Married','Dependents','Education','Self_Employed','Loan_Am
classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 75.083%
```

In [102]:
```
# The accuracy above is too high due to overfitting
# We shall create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(asce
print(featimp)
```

```
TotalIncome_log      0.300684
LoanAmount_log       0.246227
Credit_History       0.178686
Dependents           0.063144
Loan_Amount_Term     0.054235
Property_Area        0.051206
Education            0.028983
Married              0.026864
Gender               0.026130
Self_Employed        0.023841
dtype: float64
```

In [103]:
```
# We'll create a model using the top 5 variables
model = RandomForestClassifier(n_estimators=25, min_samples_split=25, max_depth=7,ma
predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents','P
classification_model(model,data_train,predictor_var,outcome_var)
```

```
Accuracy : 79.805%
Cross-Validation Score : 75.902%
```