In [1]:

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

In [2]:

```
df = pd.read_csv('train.csv')
df.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [3]:

```
df.info
```

Out[3]:

```
<bound method DataFrame.info of      Loan_ID  Gender Married Dependents
Education Self_Employed  \
0    LP001002    Male      No          0      Graduate           No
1    LP001003    Male     Yes          1      Graduate           No
2    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
10   LP001024    Male     Yes          2      Graduate           No
11   LP001027    Male     Yes          2      Graduate          NaN
12   LP001028    Male     Yes          2      Graduate           No
13   LP001029    Male      No          0      Graduate           No
14   LP001030    Male     Yes          2      Graduate           No
15   LP001032    Male      No          0      Graduate           No
16   LP001034    Male      No          1  Not Graduate           No
17   LP001036  Female      No          0      Graduate           No
```

Typesetting math: 0%

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	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

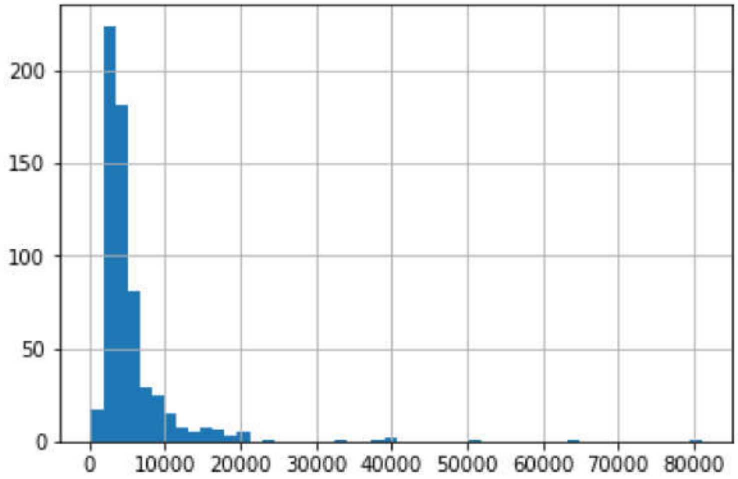
In [6]:
```
df['Property_Area'].value_counts()
```

Out[6]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

Typesetting math: 0%

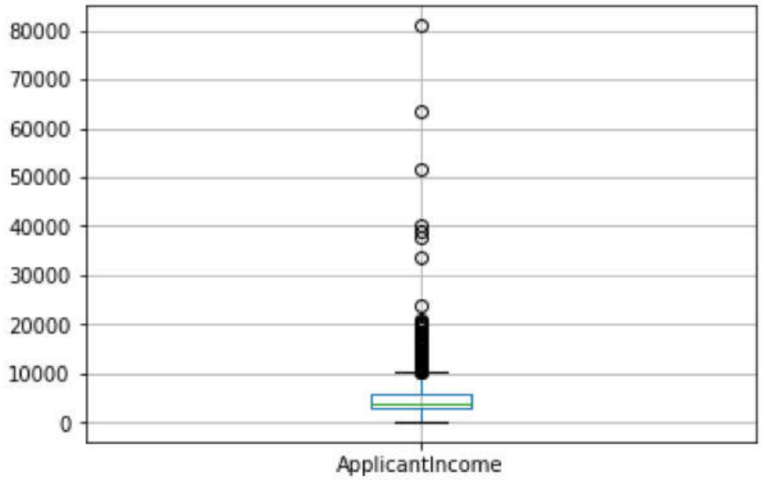
In [7]:
```
df['ApplicantIncome'].hist(bins=50)
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21b65cc76d8>
```

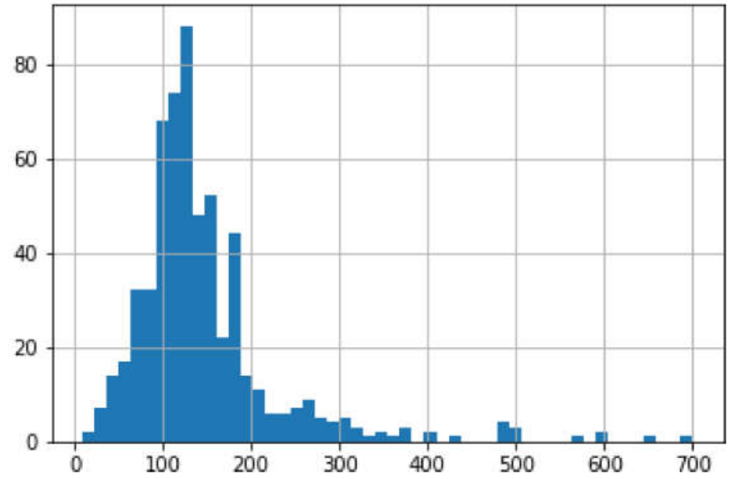


In [8]:
```
df.boxplot(column='ApplicantIncome')
```

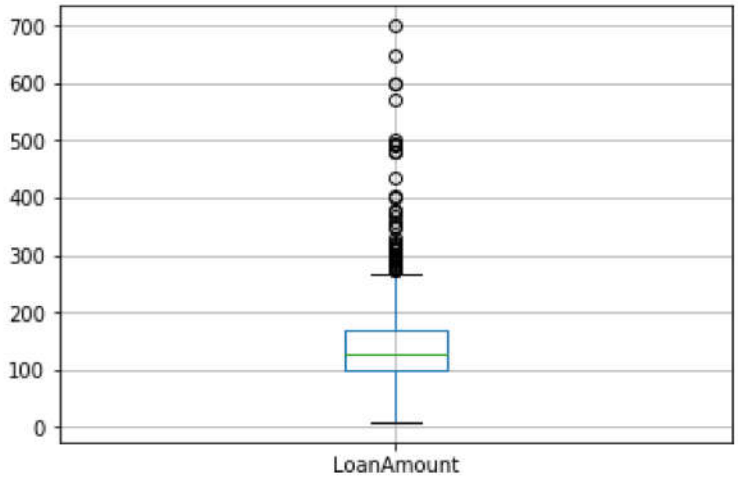
Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21b65ddca90>
```



```
In [9]: df['LoanAmount'].hist(bins=50)

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x21b660b5e80>
```



```
In [10]: df.boxplot(column='LoanAmount')

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x21b66175128>
```



In [11]:

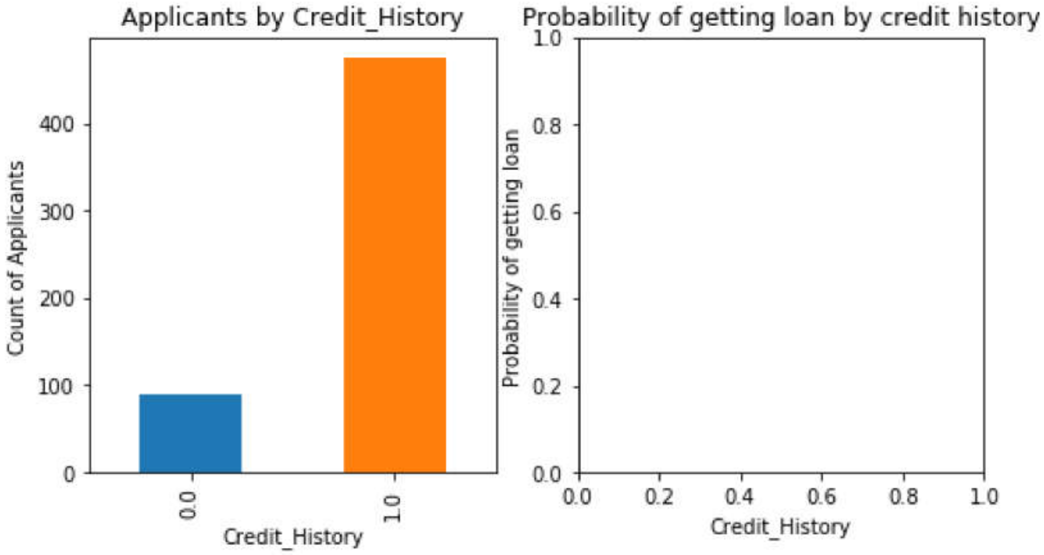
```
temp1 = df['Credit_History'].value_counts(ascending=True)
temp2 = df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lamb
print ('Frequency Table for Credit History:')
print (temp1)

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

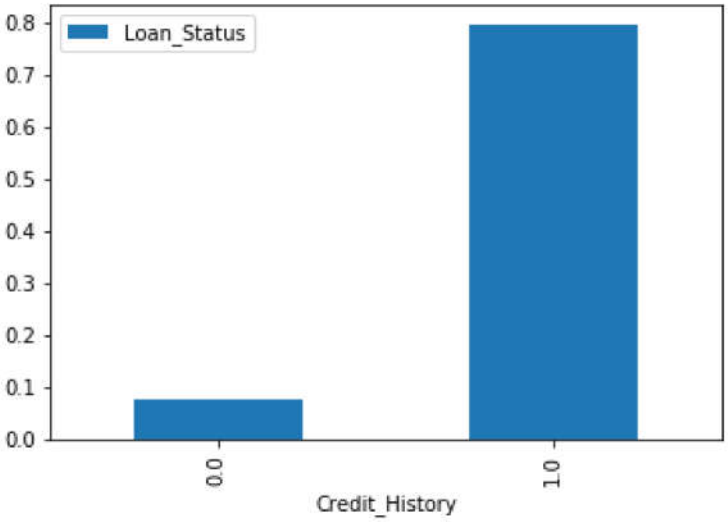
```
Frequency Table for Credit History:
0.0     89
1.0    475
Name: Credit_History, dtype: int64

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

Typesetting math: 0%

In [13]:
```
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")
```

Out[13]:
```
Text(0.5,1,'Probability of getting loan by credit history')
```





Typesetting math: 0%

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
```

In [15]:
```
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
```

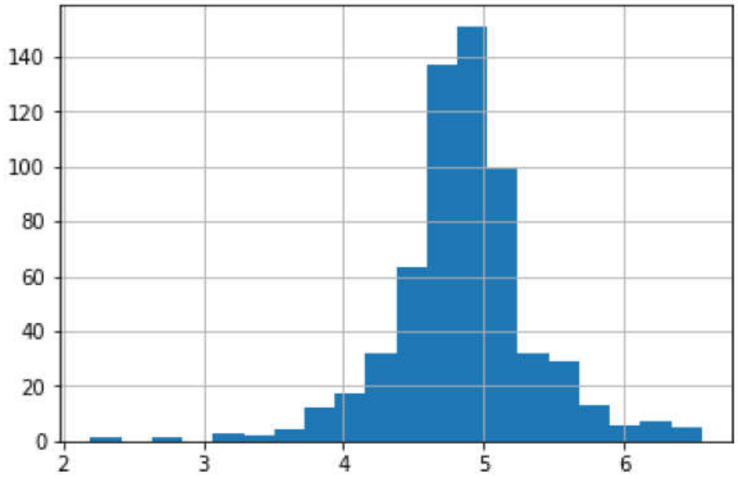
In [16]:
```
df['Self_Employed'].fillna('No',inplace=True)
```

In [17]:
```
table = df.pivot_table(values='LoanAmount', index='Self_Employed' ,columns='Educa
# 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), inplac
```

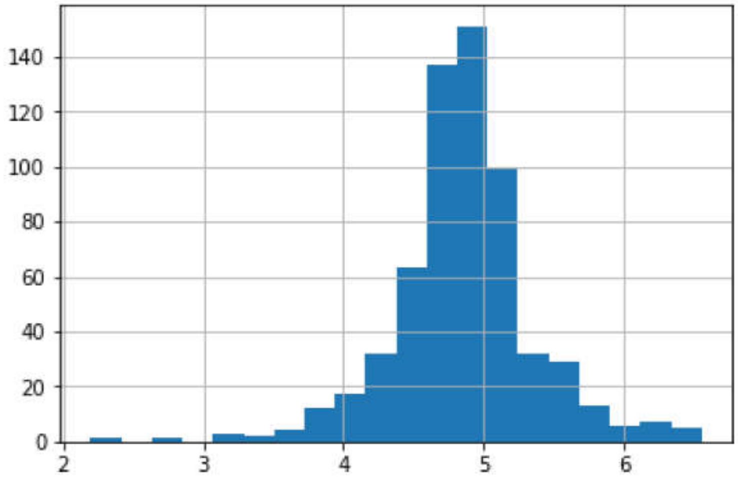
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-7b68a5f9512f> in <module>()
      4  return table.loc[x['Self_Employed'],x['Education']]
      5 # Replace missing values
----> 6 df['LoanAmount'].fillna(df[df['LoanAmount'].isnull()].apply(fage, axis=
1), inplace=True)

C:\anaconda\lib\site-packages\pandas\core\series.py in fillna(self, value, meth
od, axis, inplace, limit, downcast, **kwargs)
   3420                                           axis=axis, inplace=inplace,
   3421                                           limit=limit, downcast=downcas
t,
-> 3422                                           **kwargs)
   3423 
   3424     @Appender(generic._shared_docs['replace'] % _shared_doc_kwargs)

C:\anaconda\lib\site-packages\pandas\core\generic.py in fillna(self, value, met
hod, axis, inplace, limit, downcast)
   5398                     raise TypeError('"value" parameter must be a scala
r, dict '
   5399                                     'or Series, but you passed a '
-> 5400                                     '"{0}"'.format(type(value).__name_
_))
   5401 
   5402                 new_data = self._data.fillna(value=value, limit=limit,

TypeError: "value" parameter must be a scalar, dict or Series, but you passed a
 "DataFrame"
```

Typesetting math: 0%

```
In [18]: df['LoanAmount_log'] = np.log(df['LoanAmount'])
         df['LoanAmount_log'].hist(bins=20)

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x21b67628940>
```



```
In [19]: df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
         df['TotalIncome_log'] = np.log(df['TotalIncome'])
         df['LoanAmount_log'].hist(bins=20)

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x21b676f77f0>
```



```
In [20]: df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
         df['Married'].fillna(df['Married'].mode()[0], inplace=True)
         df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
         df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
         df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

Typesetting math: 0%

In [21]:

```
from sklearn.preprocessing import LabelEncoder
var_mod = ['Gender','Married','Dependents','Education','Self_Employed','Property_
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i])
df.dtypes
```

Out[21]:

```
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
```

Typesetting math: 0%

In [22]:

```
#Import models from scikit learn module:
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold   #For K-fold cross validation
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[te

  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])
```

```
C:\anaconda\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarnin
g: This module was deprecated in version 0.18 in favor of the model_selection m
odule into which all the refactored classes and functions are moved. Also note
that the interface of the new CV iterators are different from that of this modu
le. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Logestic Regression

Typesetting math: 0%

In [24]:

```
outcome_var = 'Loan_Status'
model = LogisticRegression()
predictor_var = ['Credit_History']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [25]:

```
#We can try different combination of variables:
predictor_var = ['Credit_History','Education','Married','Self_Employed','Property
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

Decision Tree

In [26]:

```
model = DecisionTreeClassifier()
predictor_var = ['Credit_History','Gender','Married','Education']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [27]:

```
#We can try different combination of variables:
predictor_var = ['Credit_History','Loan_Amount_Term','LoanAmount_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 89.414%
Cross-Validation Score : 68.559%
```

Random Forest

In [28]:

```
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
        'LoanAmount_log','TotalIncome_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 77.689%
```

Typesetting math: 0%

In [29]:

```
#Create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(
print (featimp)
```

```
Credit_History      0.279387
TotalIncome_log     0.255215
LoanAmount_log      0.228393
Dependents          0.054111
Property_Area       0.048545
Loan_Amount_Term    0.041235
Married             0.026748
Education           0.024773
Self_Employed       0.020965
Gender              0.020629
dtype: float64
```

In [30]:

```
model = RandomForestClassifier(n_estimators=25, min_samples_split=25, max_depth=7
predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents'
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 83.062%
Cross-Validation Score : 81.109%
```

Typesetting math: 0%