**Name - Anjali.N

**Project name - Loan Status Prediction

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df=pd.read_csv("loan.csv")
df.head()
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

In [2]:
```
df.tail()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

In [3]:
```
df.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

In [4]:
```
df['Property_Area'].value_counts()
```

Out[4]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [5]:
```
df['Loan_Status'].value_counts()
```

Out[5]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

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

Out[6]:
```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [7]:
```
df['Gender'].value_counts()
```

Out[7]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [8]:
```
df['Married'].value_counts()
```

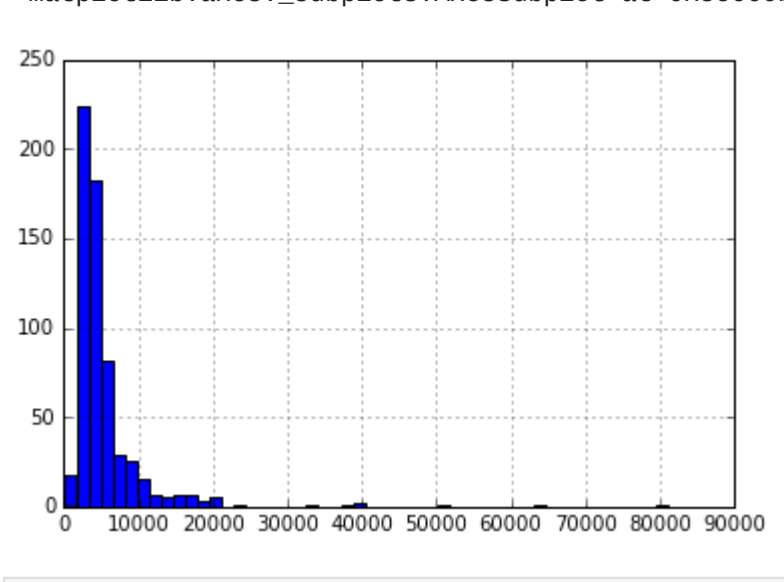
Out[8]:
```
Yes    398
No     213
Name: Married, dtype: int64
```

In [9]:
```
df['Education'].value_counts()
```

Out[9]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

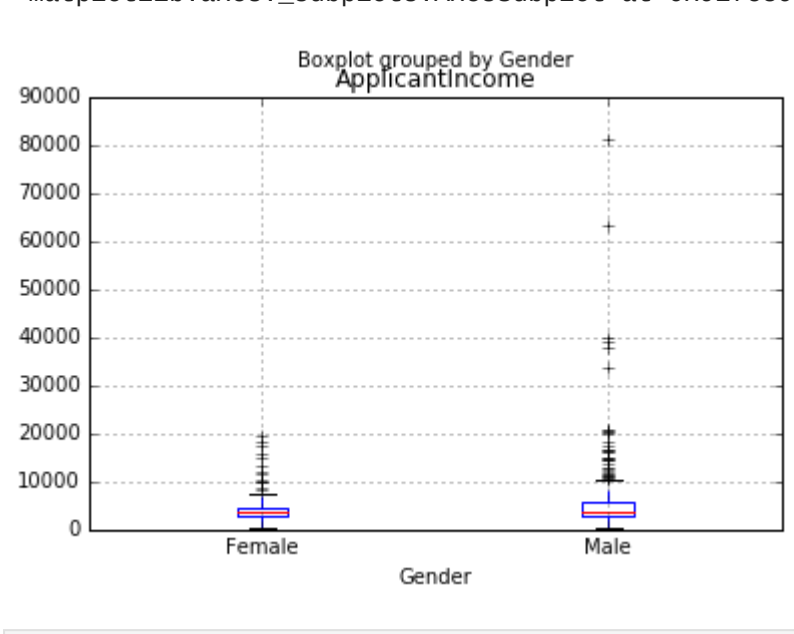
In [10]:
```
df['ApplicantIncome'].hist(bins=50)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x8e660b8>



In [11]:
```
df.boxplot(column='ApplicantIncome', by = 'Gender')
```

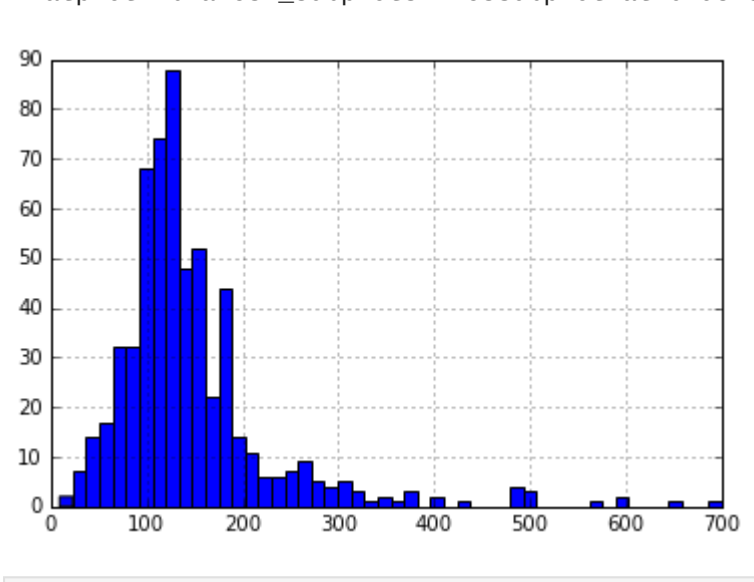
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x917e898>



In [12]:
```
#acc to above output, there are more men with more income
```

In [13]:
```
df['LoanAmount'].hist(bins=50)  #X-axis: the values of loan amount #Y-axis: the frequencies of different values of loan amount
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x9378358>



In [14]:
```
df['Credit_History'].value_counts()
```

Out[14]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [15]:
```
#creation of pivot tables using python
#the probability of getting loan as per the credit history
pt = df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lambda x: x.map({'Y':1,'N':0}).mean())
pt
```

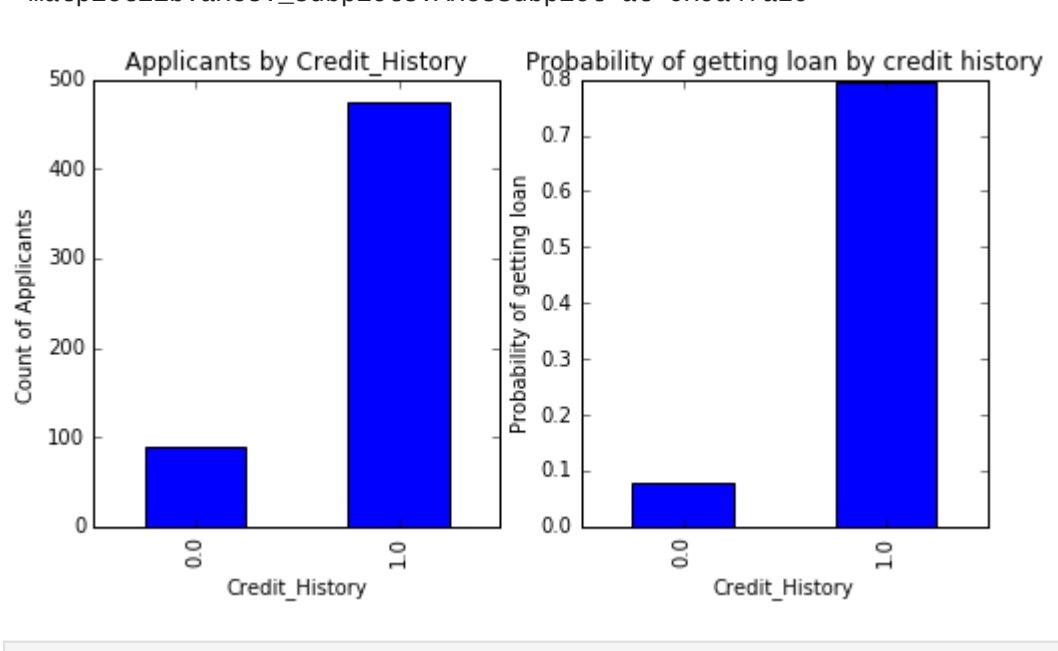
Out[15]:
```
Credit_History
0.0    0.078652
1.0    0.795789
Name: Loan_Status, dtype: float64
```

In [17]:
```
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")
ch.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")
pt.plot(kind = 'bar')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x9a4fa20>

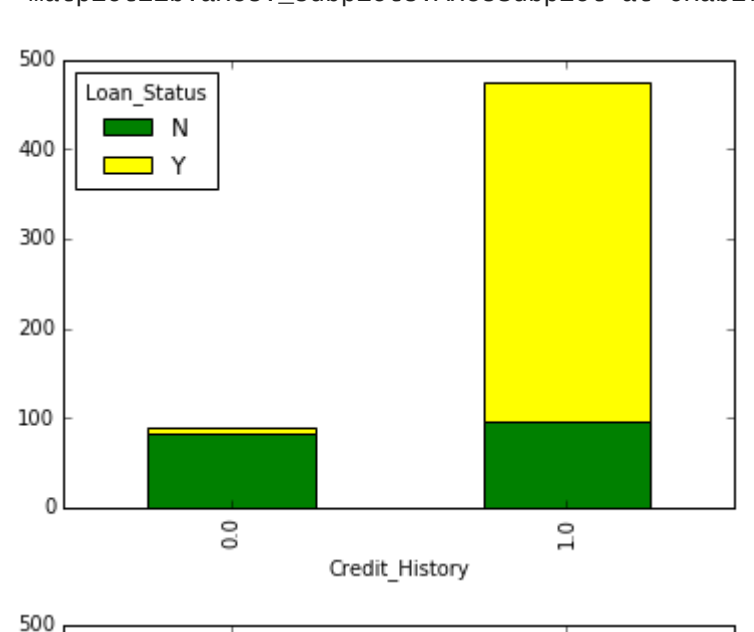


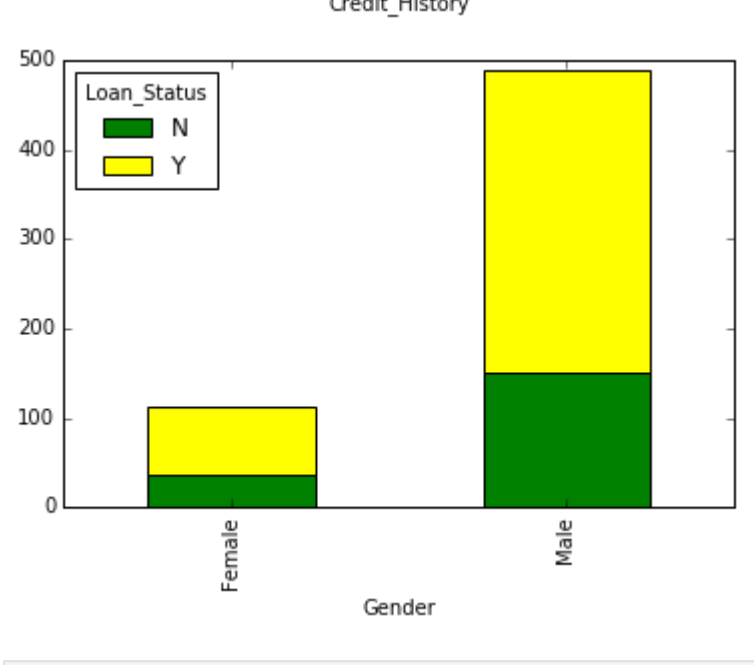
In [18]:
```
pv = df.pivot_table(values='Gender',index=['Credit_History'],aggfunc=lambda x: x.map({'Female':1,'Male':0}).mean())
pv
```

Out[18]:
```
Credit_History
0.0    0.197674
1.0    0.180258
Name: Gender, dtype: float64
```

In [19]:
```
#combining the above 2 plots in a stacked chart
stack_chart = pd.crosstab(df['Credit_History'], df['Loan_Status'])
stack_chart.plot(kind='bar', stacked=True, color=['green','yellow'], grid=False)
stack_chart = pd.crosstab(df['Gender'], df['Loan_Status'])
stack_chart.plot(kind='bar', stacked=True, color=['green','yellow'], grid=False)
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0xab17dd8>





In [20]:
```
df.apply(lambda x: sum(x.isnull()),axis=0) #to tell the missing values in each column
```

Out[20]:
```
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 [21]:
```
#to fill the missing values by mean
df.LoanAmount = df.LoanAmount.fillna(df.LoanAmount.mean())
df.describe()
```

Out[21]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	84.037468	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.250000	360.00000	1.000000
50%	3812.500000	1188.500000	129.000000	360.00000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [22]:
```
se = df['Self_Employed'].value_counts(ascending=True)
se.head()
```

Out[22]:
```
Yes     82
No     500
Name: Self_Employed, dtype: int64
```

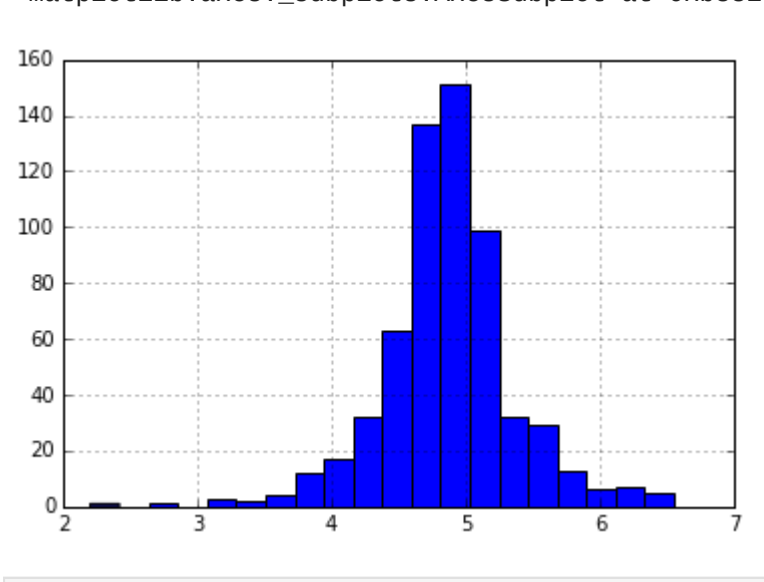
In [23]:
```
#since *no* has ~82% so, we will fill the missing values of this column with "no"
df.Self_Employed = df.Self_Employed.fillna('No')
df.describe()
```

Out[23]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	84.037468	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.250000	360.00000	1.000000
50%	3812.500000	1188.500000	129.000000	360.00000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

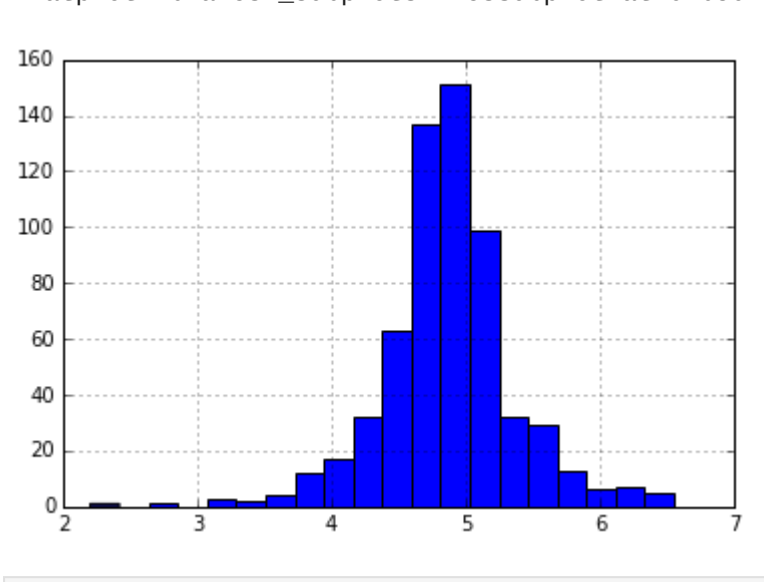
In [28]:
```
df['LoanAmount_log'] = np.log(df['LoanAmount']) #log function to reduce the extreme effect in Loan amount (which we saw in the previous barplot)
df['LoanAmount_log'].hist(bins=20)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xb3318d0>



In [29]:
```
#to decrease the extreme values of ApplicantIncome (which can be ssen from the previous barplot), we can add CoApplicantIncome to compensate the value
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log'] = np.log(df['TotalIncome'])
df['LoanAmount_log'].hist(bins=20)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0xb5049b0>



In [30]:
```
df['Capacity%'] = ((df['LoanAmount'] / df['TotalIncome']) *100).astype(float) #capacity of each applicant of how well he/she is suited to pay back his loan.
df.head(20)
```

Out[30]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
0	LP001002	Male	No	0	Graduate	No	5849	0.0	146.412162	360.0	1.0	Urban	Y	4.9
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	360.0	1.0	Rural	N	4.8
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	360.0	1.0	Urban	Y	4.1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	360.0	1.0	Urban	Y	4.7
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.000000	360.0	1.0	Urban	Y	4.9
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.000000	360.0	1.0	Urban	Y	5.5
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.000000	360.0	1.0	Urban	Y	4.5
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.000000	360.0	0.0	Semiurban	N	5.0
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.000000	360.0	1.0	Urban	Y	5.1
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.000000	360.0	1.0	Semiurban	N	5.8
10	LP001024	Male	Yes	2	Graduate	No	3200	700.0	70.000000	360.0	1.0	Urban	Y	4.2
11	LP001027	Male	Yes	2	Graduate	No	2500	1840.0	109.000000	360.0	1.0	Urban	Y	4.6
12	LP001028	Male	Yes	2	Graduate	No	3073	8106.0	200.000000	360.0	1.0	Urban	Y	5.2
13	LP001029	Male	No	0	Graduate	No	1853	2840.0	114.000000	360.0	1.0	Rural	N	4.7
14	LP001030	Male	Yes	2	Graduate	No	1299	1086.0	17.000000	120.0	1.0	Urban	Y	2.8
15	LP001032	Male	No	0	Graduate	No	4950	0.0	125.000000	360.0	1.0	Urban	Y	4.8
16	LP001034	Male	No	1	Not Graduate	No	3596	0.0	100.000000	240.0	NaN	Urban	Y	4.6
17	LP001036	Female	No	0	Graduate	No	3510	0.0	76.000000	360.0	0.0	Urban	N	4.3
18	LP001038	Male	Yes	0	Not Graduate	No	4887	0.0	133.000000	360.0	1.0	Rural	N	4.8
19	LP001041	Male	Yes	0	Graduate	No	2600	3500.0	115.000000	NaN	1.0	Urban	Y	4.7

In [31]:
```
#males are around ~81%, so we'll fill the missing values with "male"
df.Gender = df.Gender.fillna('Male')
```

In [32]:
```
gen = df['Gender'].value_counts(ascending=True)
gen.head()
```

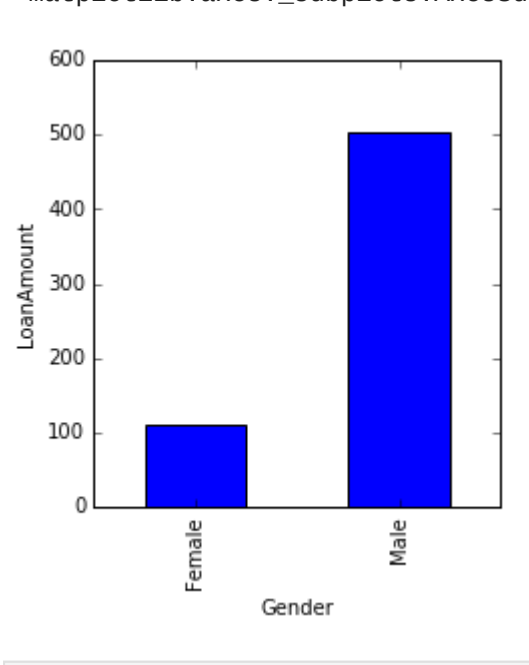
Out[32]:
```
Female    112
Male      502
Name: Gender, dtype: int64
```

In [33]:
```
#which gender has more loan amount
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)

ax1.set_xlabel('Gender')
ax1.set_ylabel('LoanAmount')

gen.plot(kind='bar')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0xb3a1978>



In [34]:
```
md = df['Married'].value_counts(ascending=True)
md.head()
```

Out[34]:
```
No     213
Yes    398
Name: Married, dtype: int64
```

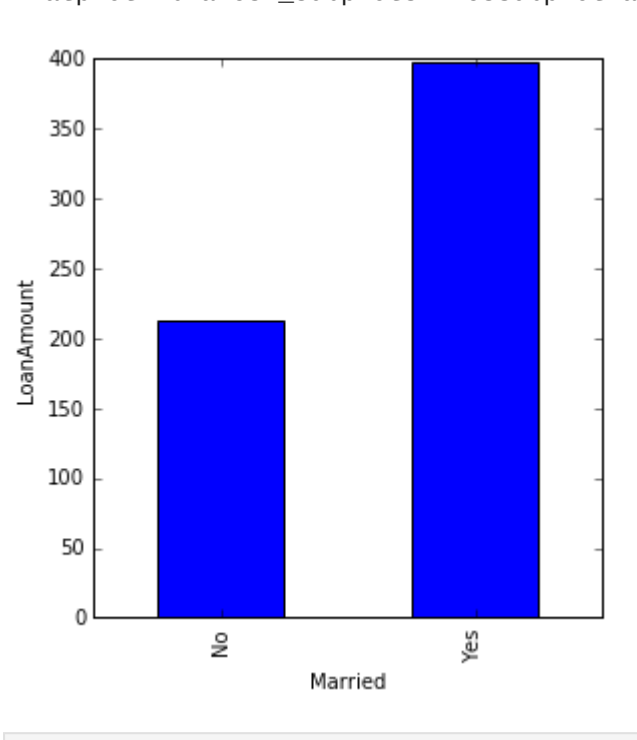
In [35]:
```
#yes has ~65%
#filling the missing values with yes
df.Married = df.Married.fillna('Yes')
```

In [36]:
```
fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(121)

ax1.set_xlabel('Married')
ax1.set_ylabel('LoanAmount')

md.plot(kind='bar')
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0xab68668>



In [41]:
```
loan = df['Loan_Status'].value_counts(ascending=True)
loan.head()
```

Out[41]:
```
0    192
1    422
Name: Loan_Status, dtype: int64
```

In [43]:
```
df.Loan_Status = df.Loan_Status.fillna('1')
df.describe()
```

Out[43]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmou
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000	614.000000	614.0
mean	0.817590	0.653094	1.719870	0.218241	0.133550	5403.459283	1621.245798	146.412162	342.00000	0.842199	1.037459	0.687296	4.8
std	0.386497	0.476373	1.039056	0.413389	0.340446	6109.041673	2926.248369	84.037468	65.12041	0.364878	0.787482	0.463973	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	9.000000	12.00000	0.000000	0.000000	0.000000	2.1
25%	1.000000	0.000000	1.000000	0.000000	0.000000	2877.500000	0.000000	100.250000	360.00000	1.000000	0.000000	0.000000	4.6
50%	1.000000	1.000000	1.000000	0.000000	0.000000	3812.500000	1188.500000	129.000000	360.00000	1.000000	1.000000	1.000000	4.8
75%	1.000000	1.000000	2.000000	0.000000	0.000000	5795.000000	2297.250000	164.750000	360.00000	1.000000	2.000000	1.000000	5.1
max	1.000000	1.000000	4.000000	1.000000	1.000000	81000.000000	41667.000000	700.000000	480.00000	1.000000	2.000000	1.000000	6.5

In [59]:
```
df.isnull().any()
```

Out[59]:
```
Loan_ID              False
Gender               False
Married              False
Dependents           False
Education            False
Self_Employed        False
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount           False
Loan_Amount_Term      True
Credit_History        True
Property_Area        False
Loan_Status          False
LoanAmount_log       False
TotalIncome          False
TotalIncome_log      False
Capacity%            False
dtype: bool
```

In [60]:
```
ch = df['Credit_History'].value_counts(ascending=True)
ch.head()
```

Out[60]:
```
0.0     89
1.0    475
Name: Credit_History, dtype: int64
```

In [66]:
```
df.Credit_History = df.Credit_History.fillna('1.0')
df.head()
```

Out[66]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmoun
0	LP001002	1	0	1	0	0	5849	0.0	146.412162	360.0	1	2	1	4.98
1	LP001003	1	1	2	0	0	4583	1508.0	128.000000	360.0	1	0	0	4.85
2	LP001005	1	1	1	0	1	3000	0.0	66.000000	360.0	1	2	1	4.18
3	LP001006	1	1	1	1	0	2583	2358.0	120.000000	360.0	1	2	1	4.78
4	LP001008	1	0	1	0	0	6000	0.0	141.000000	360.0	1	2	1	4.94

In [67]:
```
#building predictive model
#we will use scikit-learn (sklearn), for which we need to convert the categorical values into numerical values
from sklearn.preprocessing import LabelEncoder
var_col = ['Gender','Married','Dependents','Education','Self_Employed','Property_Area','Loan_Status']
le = LabelEncoder()  #Encode labels with value between 0 and n_classes-1.
for i in var_col:
    df[i] = le.fit_transform(df[i]) #Fit label encoder and return encoded labels
df.dtypes
```

Out[67]:
```
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        object
Property_Area          int64
Loan_Status            int64
LoanAmount_log       float64
TotalIncome          float64
TotalIncome_log      float64
Capacity%            float64
dtype: object
```

In [68]:
```
#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
```

In [69]:
```
#Generic function for making a classification model and accessing performance:
def classification_model(model, data, predictors, outcome):
    model.fit(data[predictors],data[outcome])
    predictions = model.predict(data[predictors])
    accuracy = metrics.accuracy_score(predictions,data[outcome])
    print "Accuracy : %s" % "{0:.3%}".format(accuracy)

    kf = KFold(data.shape[0], n_folds=5)
    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 : %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])
```

In [70]:
```
#using Logistic reasoning
outcome_var = ['Loan_Status']
model = LogisticRegression()
predictor_var = ['Credit_History']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\sklearn\utils\validation.py:526: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Accuracy : 80.945%
Cross-Validation Score : 80.946%
```

In [72]:
```
#using Logistic reasoning
outcome_var = ['Loan_Status']
model = LogisticRegression()
predictor_var = ['Credit_History','Gender','Married','Education']
classification_model(model, df,predictor_var,outcome_var)
```

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

In [75]:
```
#using Decision Tree
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 [76]:
```
model = DecisionTreeClassifier()
predictor_var = ['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Credit_History', 'Property_Area',
        'LoanAmount_log','TotalIncome_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 69.383%
```

In [74]:
```
#using Random Forest
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Credit_History', 'Property_Area',
        'LoanAmount_log','TotalIncome_log']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
Accuracy : 100.000%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Cross-Validation Score : 77.857%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:19: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

In [77]:
```
#using Random Forest
model = RandomForestClassifier(n_estimators=100)
predictor_var = ['Gender', 'Married', 'Dependents', 'Education']
classification_model(model, df,predictor_var,outcome_var)
```

```
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
Accuracy : 69.544%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
Cross-Validation Score : 66.939%
C:\Users\payel\Anaconda2\lib\site-packages\ipykernel\__main__.py:19: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```