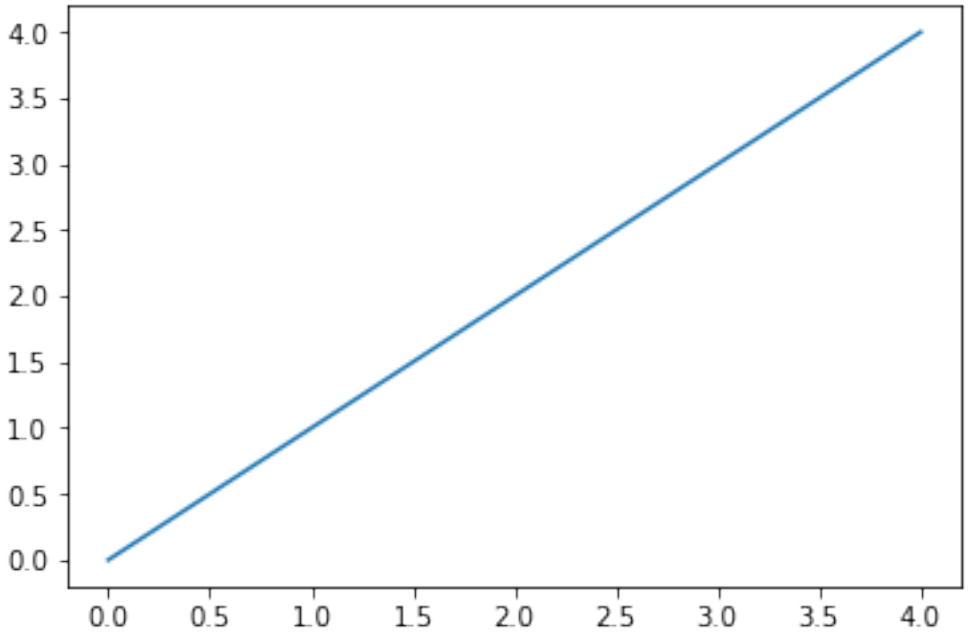
6014085_Chawalit_PandaProject

December 14, 2017

In [1]:
```
                                                    ### Part I ###
                                        ### Importing Data and Basic Function ###
```

In [89]:
```
    ## First of all, import the dataset in 3 step. ##

# Step1, import these 4 libaries first.
import pandas as pd
import numpy as np
import matplotlib as plt
import pylab as py
%matplotlib inline

# Step2, check whether the environment has loaded correctly by plot your data inline.
py.plot(range(5))

# Step3, define your source of data.
df=pd.read_csv("C:/Users/Poom/Downloads/Term_Project_Data.csv")
```



In [90]:
```
## After you have your dataset in Python, These are some basic fucntions that pyt

#1) Have a look at few top rows by using the function head()

df.head(10)
```

Out[90]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	\
0	LP001002	Male	No	0.0	Graduate	No	
1	LP001003	Male	Yes	1.0	Graduate	No	
2	LP001005	Male	Yes	0.0	Graduate	Yes	
3	LP001006	Male	Yes	0.0	Not Graduate	No	
4	LP001008	Male	No	0.0	Graduate	No	
5	LP001011	Male	Yes	2.0	Graduate	Yes	
6	LP001013	Male	Yes	0.0	Not Graduate	No	
7	LP001014	Male	Yes	3.0	Graduate	No	
8	LP001018	Male	Yes	2.0	Graduate	No	
9	LP001020	Male	Yes	1.0	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	0.0	Semiurban	N
8	1.0	Urban	Y
9	1.0	Semiurban	N

In [91]:
```
#2) look at summary of numerical fields by using describe() function

df.describe()
```

Out[91]:
```
       Dependents  ApplicantIncome  CoapplicantIncome  LoanAmount  \
count  599.000000       614.000000         614.000000  592.000000
mean     0.762938      5403.459283        1621.245798  146.412162
std      1.015216      6109.041673        2926.248369   85.587325
min      0.000000       150.000000           0.000000    9.000000
25%      0.000000      2877.500000           0.000000  100.000000
50%      0.000000      3812.500000        1188.500000  128.000000
75%      2.000000      5795.000000        2297.250000  168.000000
max      3.000000     81000.000000       41667.000000  700.000000

       Loan_Amount_Term  Credit_History
count         600.00000      564.000000
mean          342.00000        0.842199
std            65.12041        0.364878
min            12.00000        0.000000
25%           360.00000        1.000000
50%           360.00000        1.000000
75%           360.00000        1.000000
max           480.00000        1.000000
```

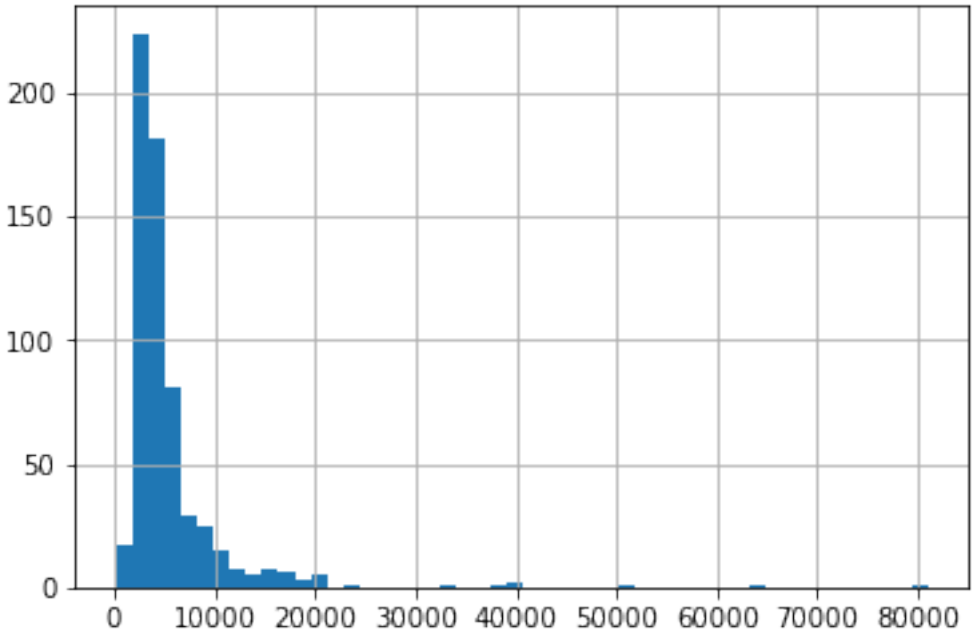
In [92]:
```
#2.1) Choose to look at specefic information of non-numerical values Ex:'Property_Are
df['Property_Area'].value_counts()
```

Out[92]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [93]:
```
                                                ### Part II ###
                                        ### Numeric variables analysis ###
```

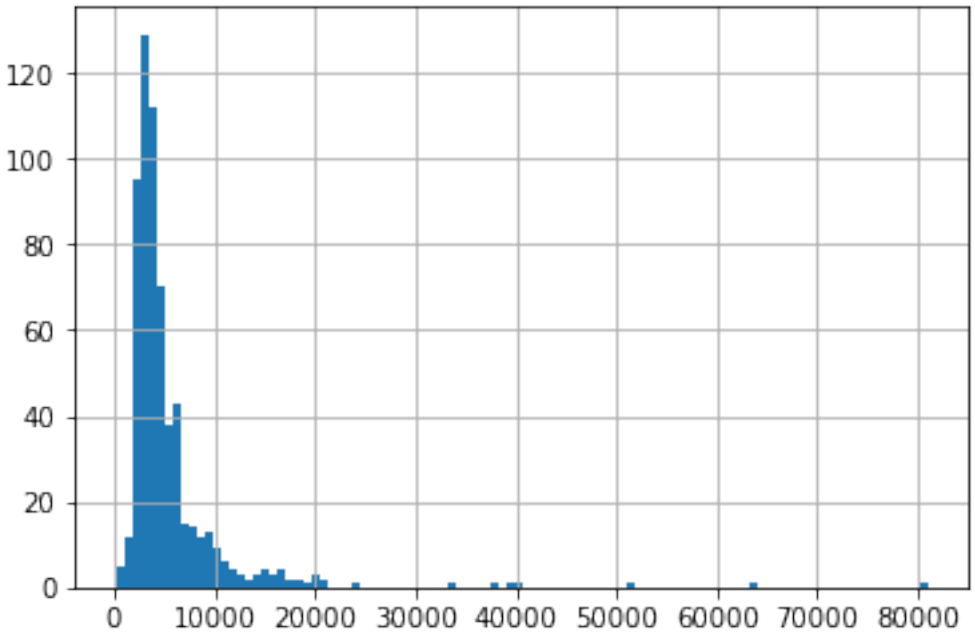
In [94]:
```
    ## 1)"Histogram" ##

# Plotting the "Histogram". This show most applicant income is lower than 20,000.
df['ApplicantIncome'].hist(bins=50)
```

Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22ddf410c50>
```



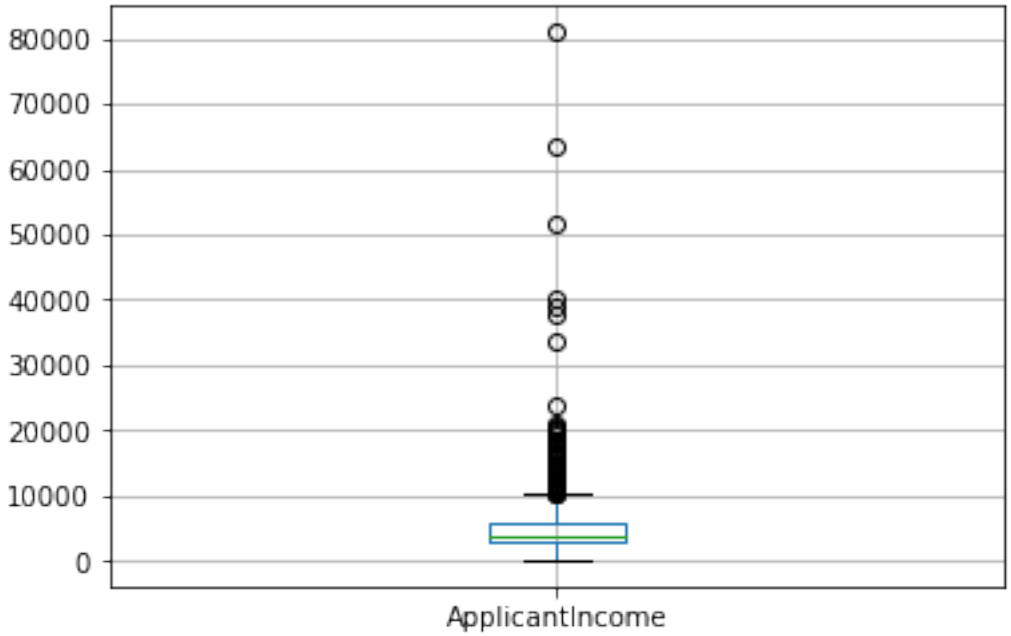
In [95]:
```
#You can choose for deepper detail in number
df['ApplicantIncome'].hist(bins=100)
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddf4849b0>



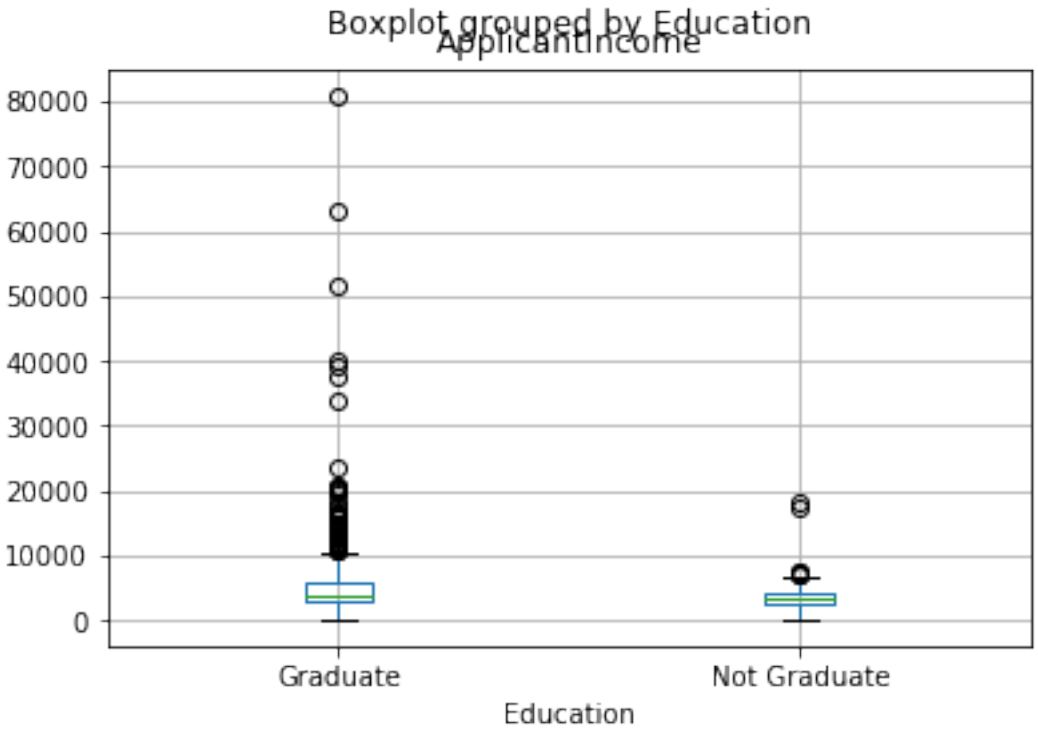
In [96]:
```
## 2)"Box Plots" ##

# Plotting the "Box Plots"
df.boxplot(column='ApplicantIncome')
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddf64d7b8>



In [97]:
```
#Use "Box Plots" to show both education and income in same graph for compare.
df.boxplot(column='ApplicantIncome', by = 'Education')
```

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddf7a1358>



In [98]:
```
### Part III ###
### Categorical variable analysis ###
```

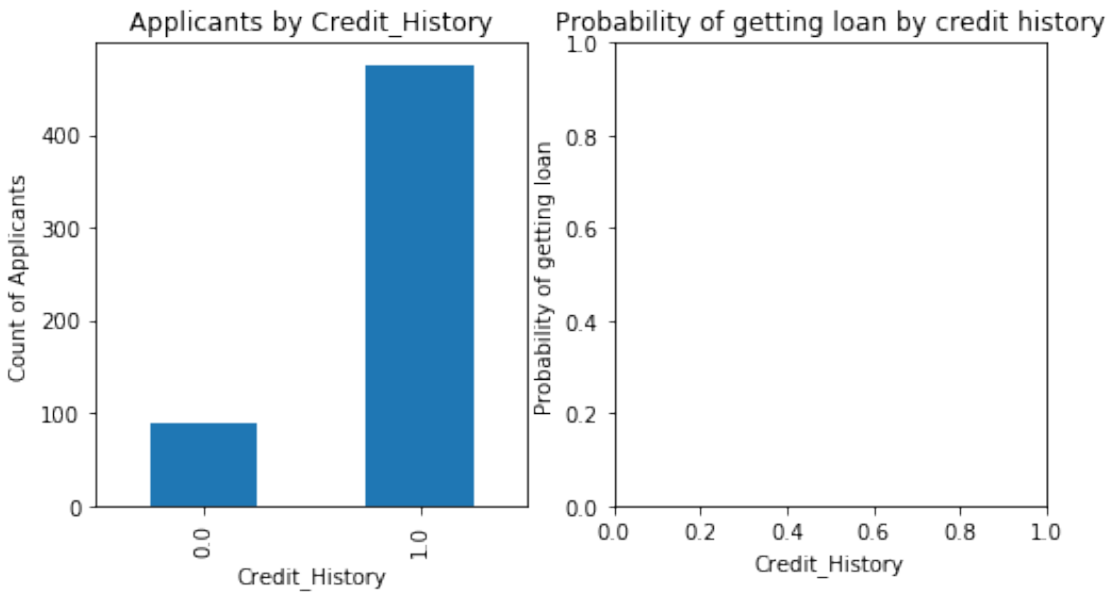
In [99]:
```
## First Generate a similar insight using Python. ##

temp1 = df['Credit_History'].value_counts(ascending=True)
temp2 = df.pivot_table(values='Loan_Status',index=['Credit_History'],aggfunc=lambda x

print('Frequency Table for Credit History:')
print (temp1)

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

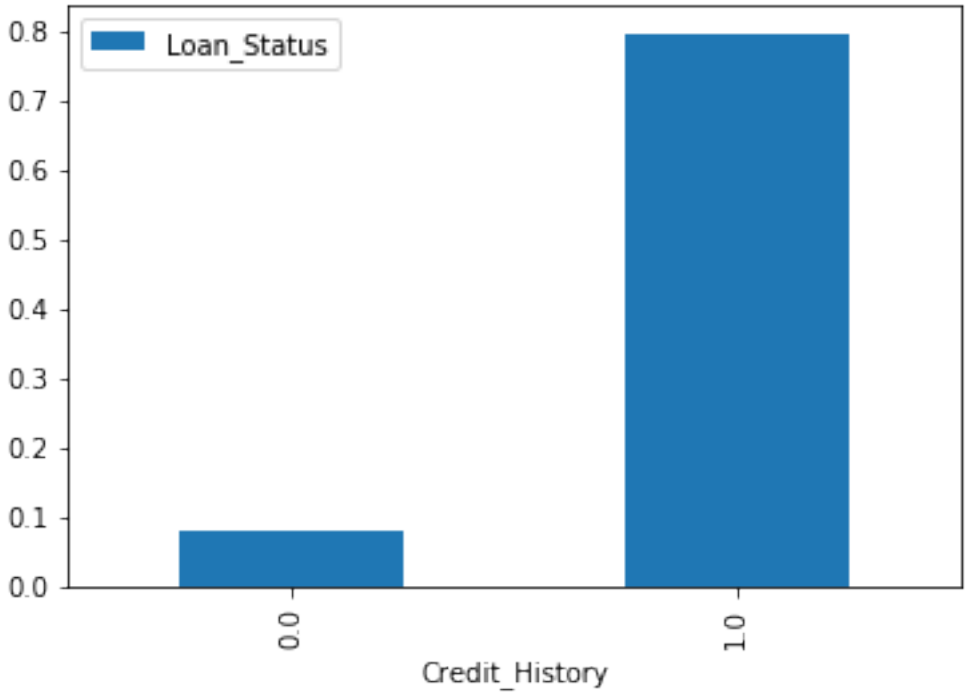
```
Frequency Table for Credit History:
0.0     89
1.0    475
Name: Credit_History, dtype: int64

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

```
1.0                 0.795789
```

In [100]:
```
## 1)Bar Chart ##

# This can be plotted as a bar chart using the matplotlib library with following cod

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")
```

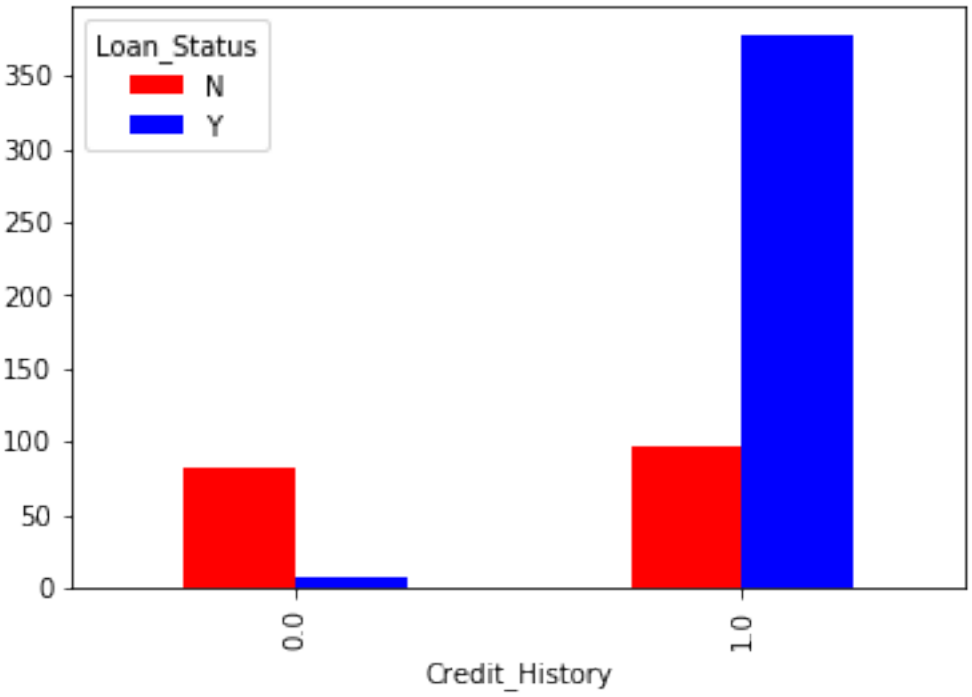
Out[100]:
```
<matplotlib.text.Text at 0x22ddf9519b0>
```





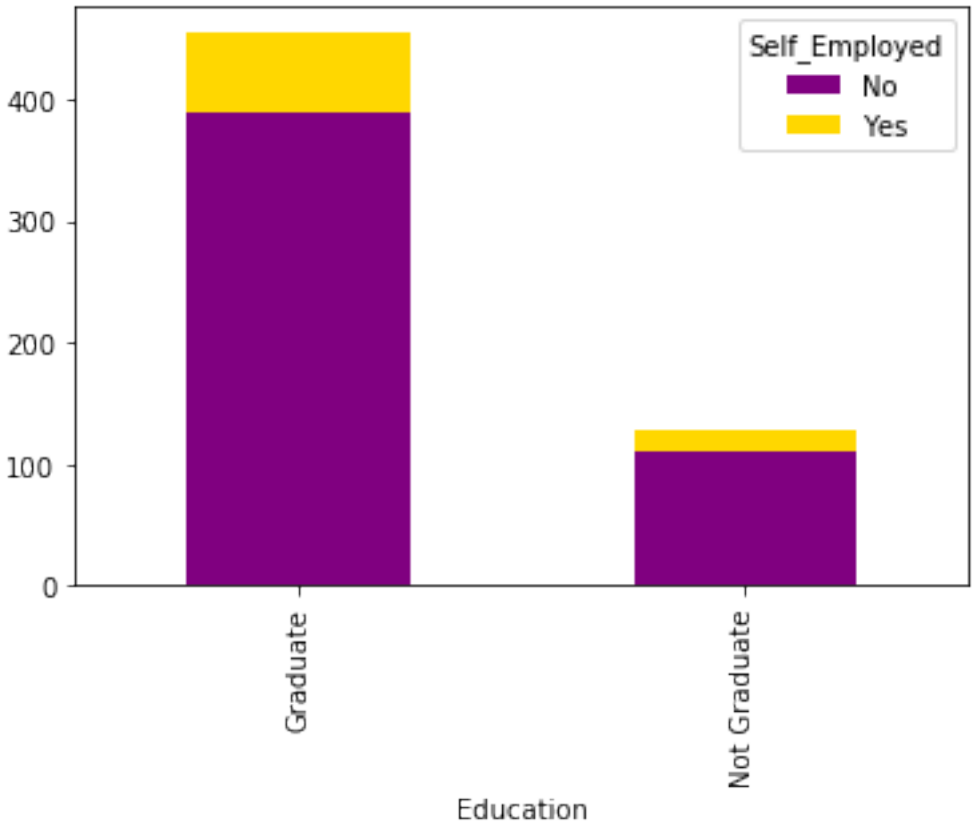

```
In [101]: temp3 = pd.crosstab(df['Credit_History'], df['Loan_Status'])
          temp3.plot(kind='bar', stacked=False, color=['red','blue'], grid=False)
```

```
Out[101]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddf9dae48>
```



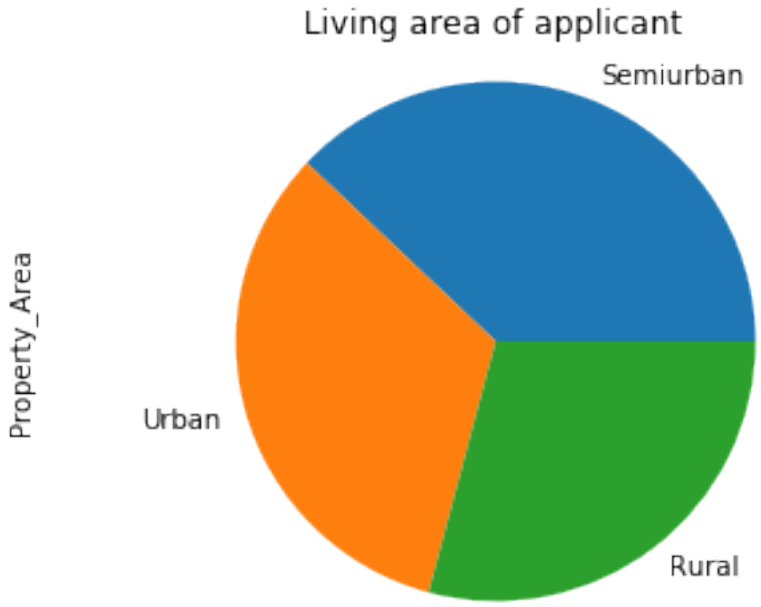
In [102]:
```
## 2)stacked chart ##

#Both graduate and nongaduate have more self emploed.
temp3 = pd.crosstab(df['Education'], df['Self_Employed'])
temp3.plot(kind='bar', stacked=True, color=['purple','gold'], grid=False)
```

Out[102]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfb237b8>



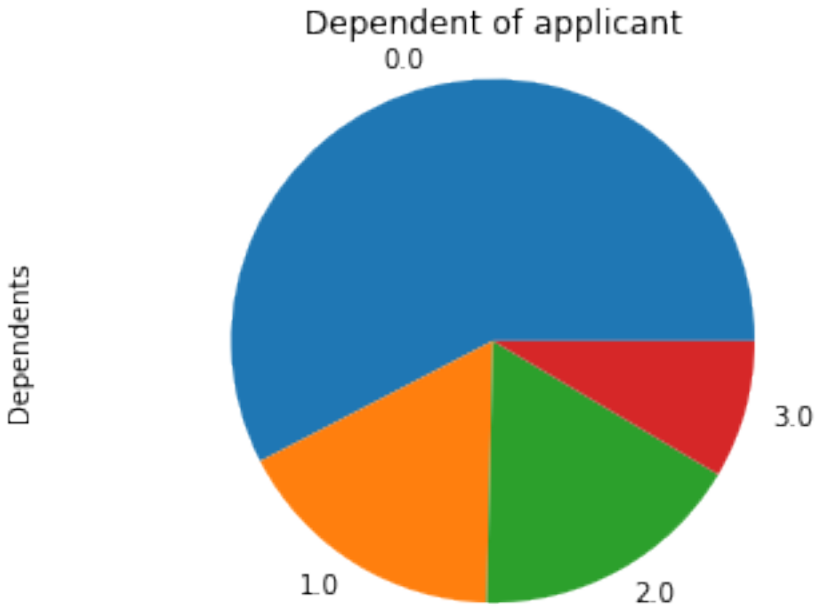
In [103]:
```
## 3) Pie Chart ##

df.Property_Area.value_counts().plot(kind='pie')
plt.axis('equal')
plt.title('Living area of applicant')
```

Out[103]: <matplotlib.text.Text at 0x22ddfc387f0>



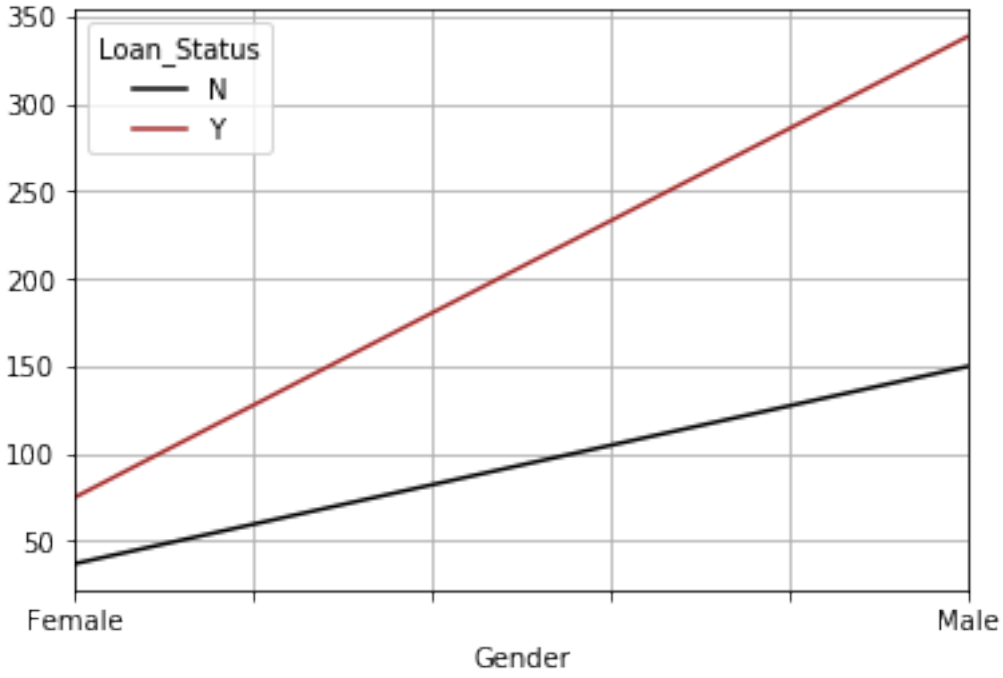

```
In [104]: df.Dependents.value_counts().plot(kind='pie')
          plt.axis('equal')
          plt.title('Dependent of applicant')
```

```
Out[104]: <matplotlib.text.Text at 0x22ddfcbf9b0>
```



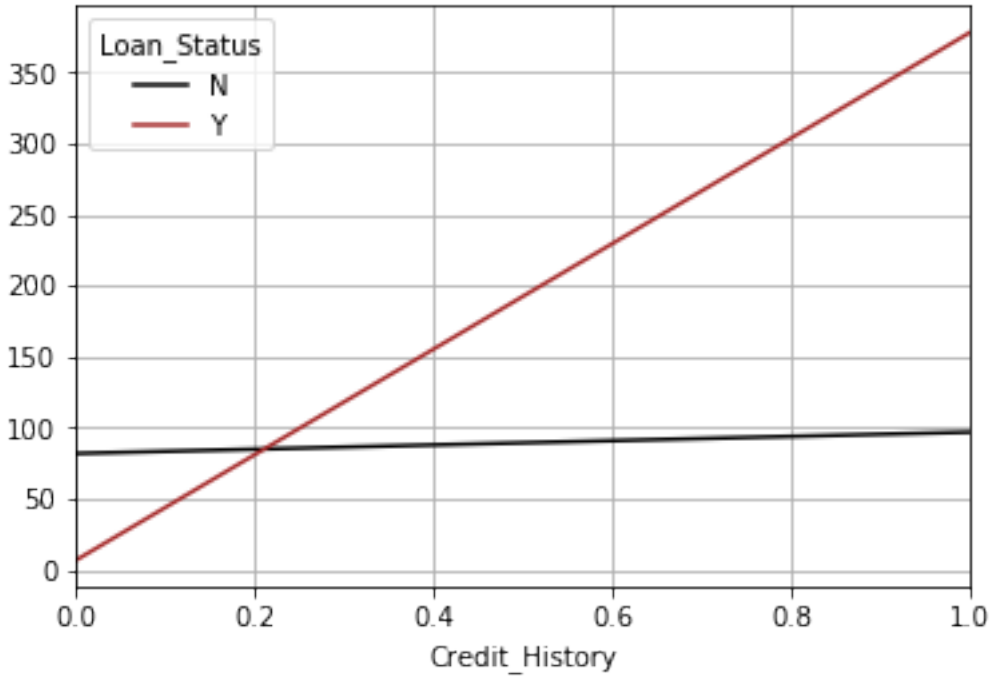
In [105]:
```
## 4)Line graph ##

temp3 = pd.crosstab(df['Gender'], df['Loan_Status'],)
temp3.plot(kind='line', stacked=False, color=['black','brown'], grid=True)
```

Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfc91320>



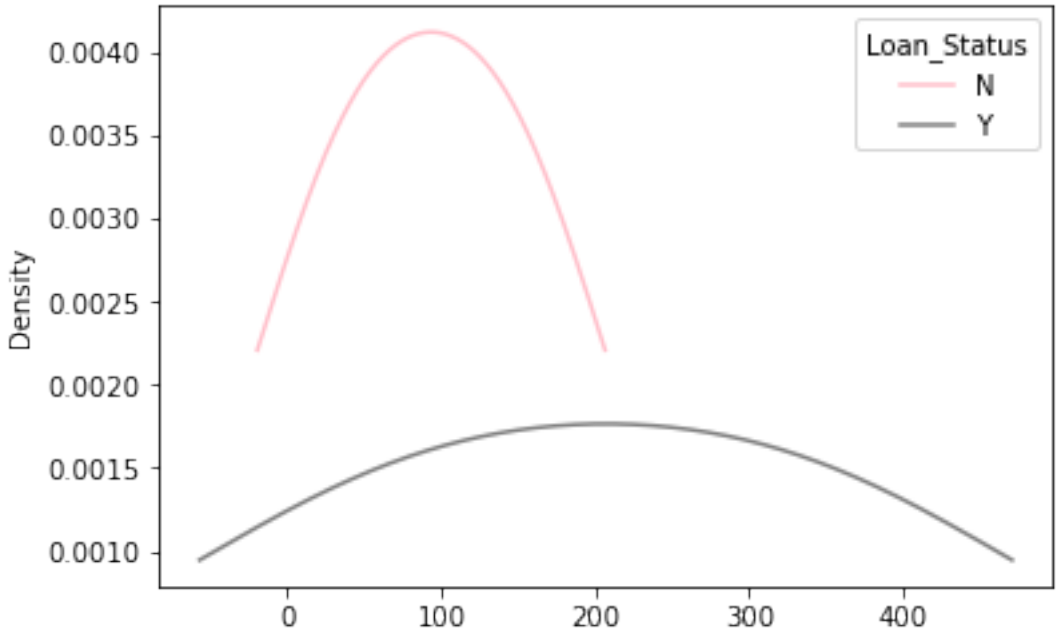
In [106]:
```
temp3 = pd.crosstab(df['Credit_History'], df['Loan_Status'],)
temp3.plot(kind='line', stacked=False, color=['black','brown'], grid=True)
```

Out[106]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfdd26d8>



In [107]:
```
## 5) Kernel Density Estimation plot ##

temp3 = pd.crosstab(df['Gender'], df['Loan_Status'])
temp3.plot(kind='kde', stacked=False, color=['pink','gray'], grid=False)
```

Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfdcb550>



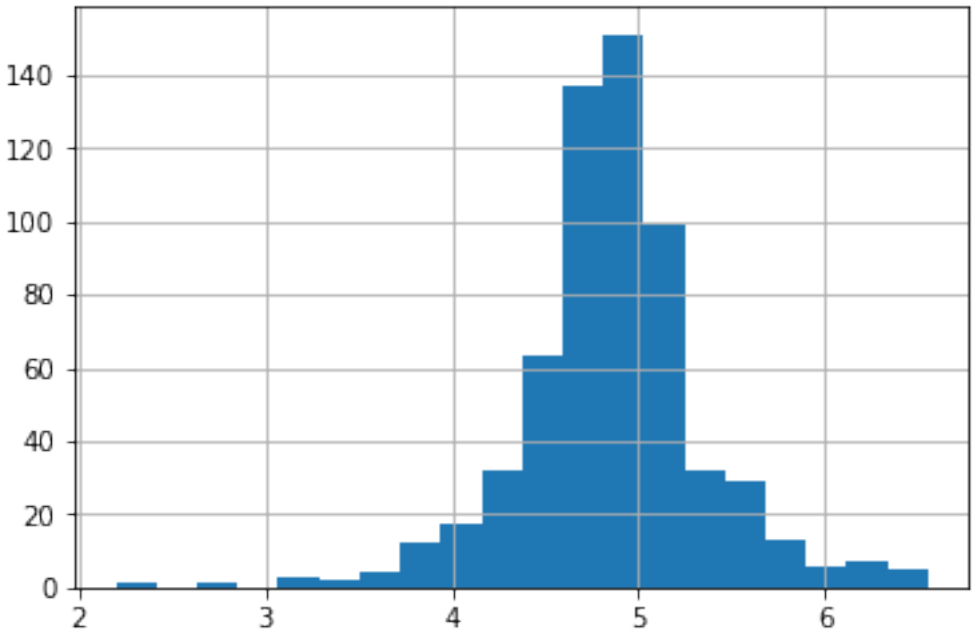
```
In [108]:                                                    ### Part IV ###
                                                    ### Data Munging in Python ###

In [109]: ##1) Check missing values in the dataset
          df.apply(lambda x: sum(x.isnull()),axis=0)

Out[109]: 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 [110]: ##2) Check Majority Of the Missing Data.
          df['Gender'].value_counts()

Out[110]: Male      489
          Female    112
          Name: Gender, dtype: int64

In [111]: df['Married'].value_counts()

Out[111]: Yes    398
          No     213
          Name: Married, dtype: int64

In [112]: df['Dependents'].value_counts()

Out[112]: 0.0    345
          1.0    102
          2.0    101
          3.0     51
          Name: Dependents, dtype: int64

In [113]: df['Self_Employed'].value_counts()

Out[113]: No     500
          Yes     82
          Name: Self_Employed, dtype: int64
```

```
In [114]: df['Loan_Amount_Term'].value_counts()

Out[114]: 360.0    512
          180.0     44
          480.0     15
          300.0     13
          84.0       4
          240.0      4
          120.0      3
          36.0       2
          60.0       2
          12.0       1
          Name: Loan_Amount_Term, dtype: int64

In [115]: df['Credit_History'].value_counts()

Out[115]: 1.0    475
          0.0     89
          Name: Credit_History, dtype: int64

In [116]: ##3) Fill the missing data with majority value.
          df['Gender'].fillna('M',inplace=True)
          df['Married'].fillna('Yes',inplace = True)
          df['Dependents'].fillna(0,inplace=True)
          df['Self_Employed'].fillna('No',inplace = True)
          df['LoanAmount'].fillna(df['LoanAmount'].mean(),inplace = True)
          df['Loan_Amount_Term'].fillna(360,inplace = True)
          df['Credit_History'].fillna(1,inplace = True)

In [117]: ##4) Check again to make sure no missing data left.
          df.apply(lambda x: sum(x.isnull()),axis=0)

Out[117]: Loan_ID              0
          Gender               0
          Married              0
          Dependents           0
          Education            0
          Self_Employed        0
          ApplicantIncome      0
          CoapplicantIncome    0
          LoanAmount           0
          Loan_Amount_Term     0
          Credit_History       0
          Property_Area        0
          Loan_Status          0
          dtype: int64

In [118]: ##5) Check type of data.
          df.dtypes
```

Out[118]:
```
Loan_ID               object
Gender                object
Married               object
Dependents           float64
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

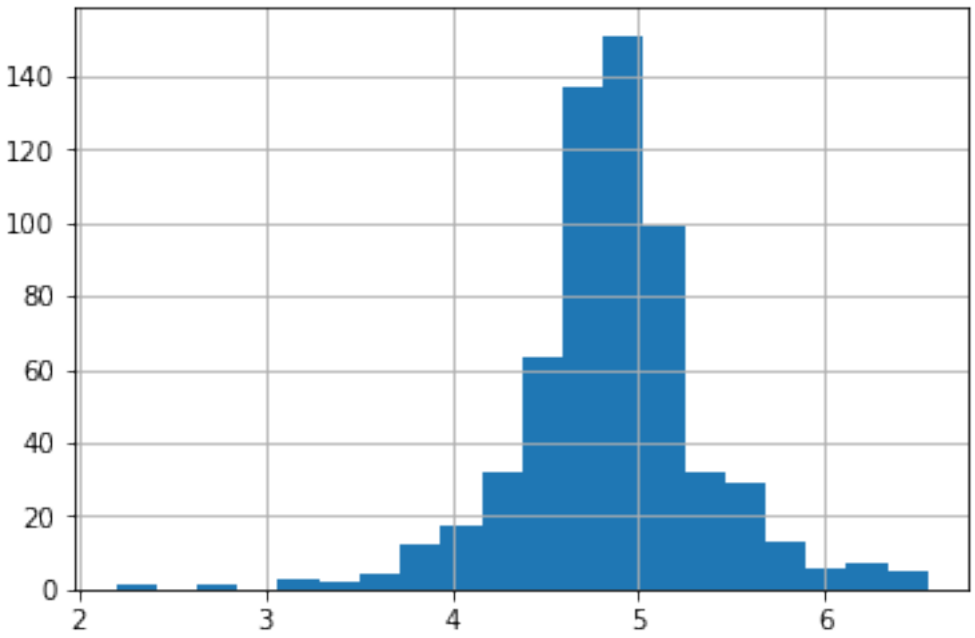
In [119]:
```
    ## Treat for extreme values in distribution of LoanAmount and ApplicantIncome ##
df['LoanAmount_log'] = np.log(df['LoanAmount'])
df['LoanAmount_log'].hist(bins=20)
```

Out[119]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfa4bf60>



In [120]:
```
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['TotalIncome_log'] = np.log(df['TotalIncome'])
df['LoanAmount_log'].hist(bins=20)
```

Out[120]: <matplotlib.axes._subplots.AxesSubplot at 0x22ddfec9080>



In [121]:
```
                                                    ### Part V ###
                                        ### Building a Predictive Model in Python ###
```

In [122]:
```
#Sklearn requires all inputs to be numeric, so code below convert all categorical va

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

Out[122]:
```
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 [123]:
```
#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[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 [124]:
```
#There are 3 model of predictation method.
```

In [125]:
```
    ## 1) Logistic Regression ##
outcome_var = 'Loan_Status'
```

```
model = LogisticRegression() #Defince Model1

#Combination of several variables
predictor_var = ['Credit_History']
classification_model(model, df,predictor_var,outcome_var)
```

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

In [126]:
```
#We can try different combination of variables:
predictor_var = ['Credit_History','Married','Property_Area']
classification_model(model, df,predictor_var,outcome_var)
```

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

In [127]:
```
## 2) Decision Tree
model = DecisionTreeClassifier() #Define Model2

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

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

In [128]:
```
#try different combination of variables:
predictor_var = ['Dependents','Loan_Amount_Term','LoanAmount_log','Education','Marri
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 58.778%
```

In [129]:
```
# Best Result I Can Get #
predictor_var = ['Credit_History','Loan_Amount_Term','LoanAmount_log','Education','A
classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 71.993%
```

In [130]:
```
## 3) Random Forest
model = RandomForestClassifier(n_estimators=100) #Define model 3

#Use all avariables
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 : 78.178%
```

```
In [131]: #Reducing the number of predictors and Tuning the model parameters.
          #Create a series with feature importances:

          featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(asc
          print(featimp)
```

```
Credit_History      0.267072
TotalIncome_log     0.257915
LoanAmount_log      0.224428
Dependents          0.053638
Property_Area       0.051948
Loan_Amount_Term    0.046452
Married             0.027358
Gender              0.026240
Education           0.023955
Self_Employed       0.020994
dtype: float64
```

```
In [132]: # use the top 5 variables for creating a model
          model = RandomForestClassifier(n_estimators=25, min_samples_split=25, max_depth=7, ma
          predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents','Pr
          classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 82.410%
Cross-Validation Score : 80.458%
```

```
In [133]: ### Test More ###
```

```
In [134]: # Try use Top 5 variables in Decision Tree  >> Out come is mine one is better.
          model = DecisionTreeClassifier()
          predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents','Pr
          classification_model(model, df,predictor_var,outcome_var)
```

```
Accuracy : 100.000%
Cross-Validation Score : 69.868%
```

```
In [135]: # Try use Top 5 variables in Logistic Regression.
          model = LogisticRegression() #Defince Model1

          #Combination of several variables
          predictor_var = ['TotalIncome_log','LoanAmount_log','Credit_History','Dependents','Pr
          classification_model(model, df,predictor_var,outcome_var)
```

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

```
In [1]: ### Thank you ###
```