In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
test = pd.read_csv('test.csv')
test.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

In [3]:
```
train = pd.read_csv('train.csv')
train.head()
```

Out[3]:

	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 [4]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
train_original = train.copy()
test_original = test.copy()
```

In [6]:
```
train.columns
```

Out[6]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [7]:
```
test.columns
```

Out[7]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

In [8]:
```
train.dtypes
```

Out[8]:
```
Loan_ID               object
Gender                object
Married               object
Dependents            object
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 [9]:
```
print('Training data shape: ', train.shape)
train.head()
```

```
Training data shape:  (614, 13)
```

Out[9]:

	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 [10]:
```
print('Test data shape: ', test.shape)
test.head()
```

```
Test data shape:  (367, 12)
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

In [11]:
```
#univariate analysis
```

In [12]:
```
train["Loan_Status"].count()
```

Out[12]:
```
614
```

In [13]:
```
train["Loan_Status"].value_counts()
```

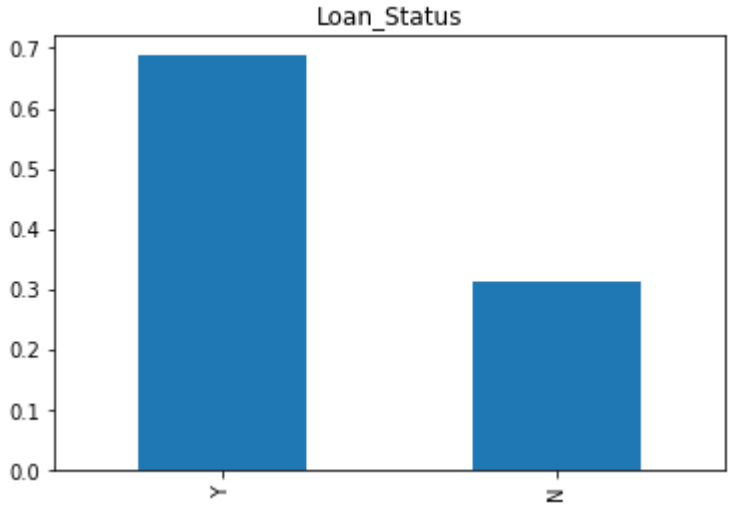
Out[13]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [14]:
```
# Normalize can be set to True to print proportions instead of number
train["Loan_Status"].value_counts(normalize=True)*100
```

Out[14]:
```
Y    68.729642
N    31.270358
Name: Loan_Status, dtype: float64
```

In [15]:
```
train["Loan_Status"].value_counts(normalize=True).plot.bar(title = 'Loan_Status'
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd4f2d52b0>
```



In [16]:
```
#Analysis on "Gender" variable
```

In [17]:
```
train["Gender"].count()
```

Out[17]:
```
601
```

In [18]:
```
train["Gender"].value_counts()
```

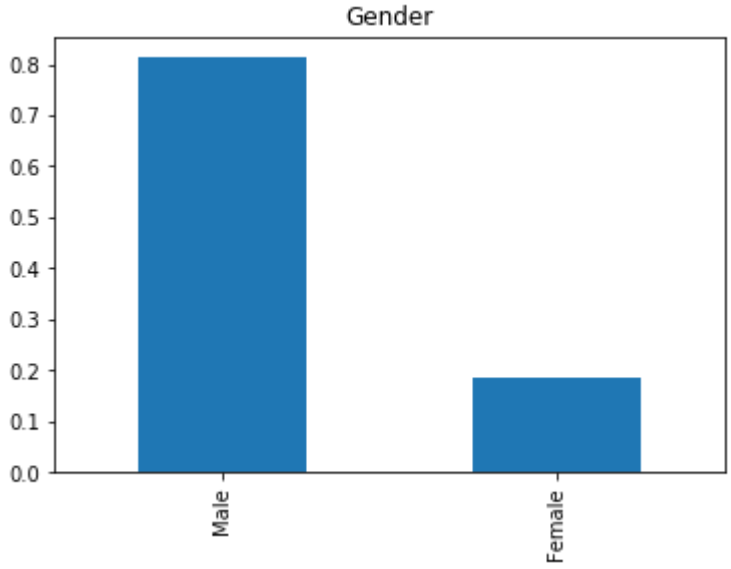
Out[18]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [19]:
```
train['Gender'].value_counts(normalize=True)*100
```

Out[19]:
```
Male      81.364393
Female    18.635607
Name: Gender, dtype: float64
```

In [20]:
```
train['Gender'].value_counts(normalize=True).plot.bar(title= 'Gender')
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd4f5d0d68>
```



In [21]:
```
#Analysis on "Married" variable
```

In [22]:
```
train["Married"].count()
```

Out[22]:
```
611
```

In [23]:
```
train["Married"].value_counts()
```

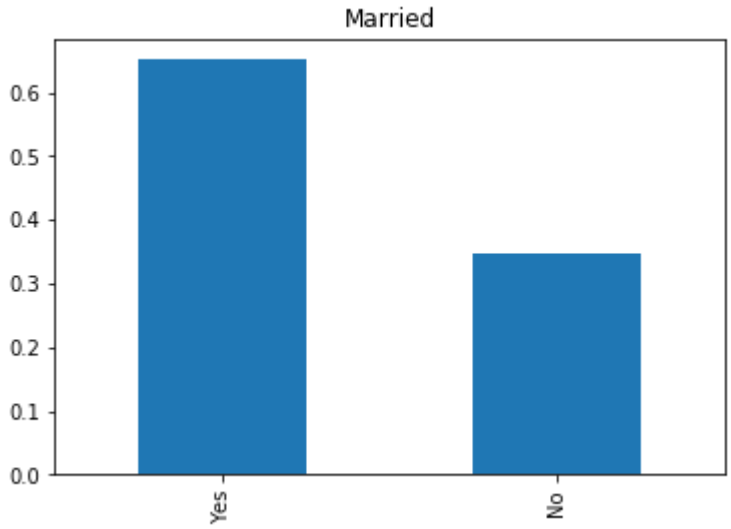
Out[23]:
```
Yes    398
No     213
Name: Married, dtype: int64
```

In [24]:
```
train['Married'].value_counts(normalize=True)*100
```

Out[24]:
```
Yes    65.139116
No     34.860884
Name: Married, dtype: float64
```

In [25]:
```
train['Married'].value_counts(normalize=True).plot.bar(title= 'Married')
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd5064f2b0>
```



In [26]:
```
#Analysis on "Self_Employed" variable
```

In [27]:
```
train["Self_Employed"].count()
```

Out[27]:
```
582
```

In [28]:
```
train["Self_Employed"].value_counts()
```

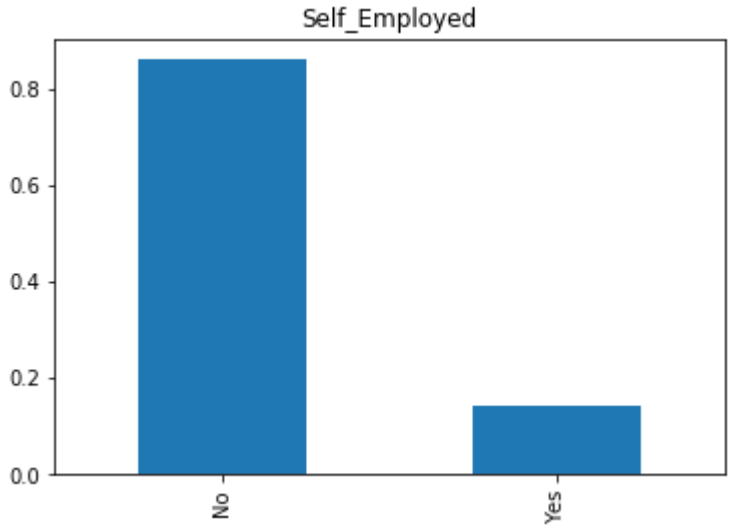
Out[28]:
```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [29]:
```
train['Self_Employed'].value_counts(normalize=True)*100
```

Out[29]:
```
No     85.910653
Yes    14.089347
Name: Self_Employed, dtype: float64
```

In [30]:
```
train['Self_Employed'].value_counts(normalize=True).plot.bar(title='Self_Employe
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd5068dd68>



In [31]:
```
#Analysis on "Credit_History" variable
```

In [32]:
```
train["Credit_History"].count()
```

Out[32]: 564

In [33]:
```
train["Credit_History"].value_counts()
```

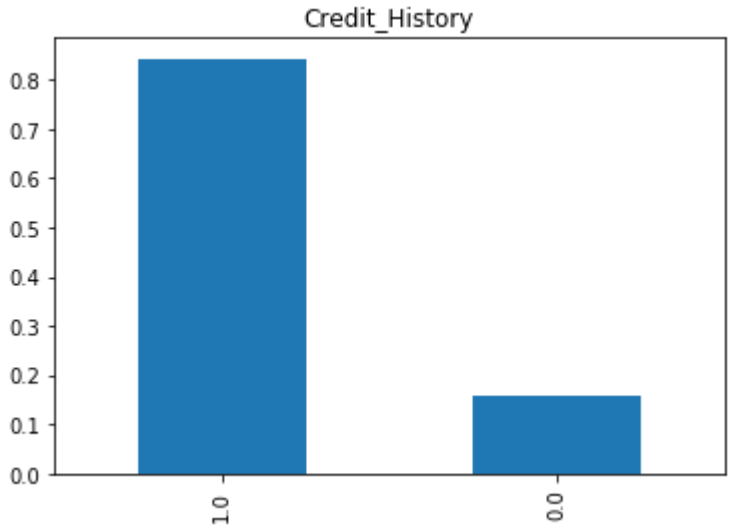
Out[33]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [34]:
```
train['Credit_History'].value_counts(normalize=True)*100
```

Out[34]:
```
1.0    84.219858
0.0    15.780142
Name: Credit_History, dtype: float64
```

In [35]:
```
train['Credit_History'].value_counts(normalize=True).plot.bar(title='Credit_Hist
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd506ffe48>
```



In [36]:
```
#Analysis on "Dependents" variable
```

In [37]:
```
train['Dependents'].count()
```

Out[37]:
```
599
```

In [38]:
```
train["Dependents"].value_counts()
```

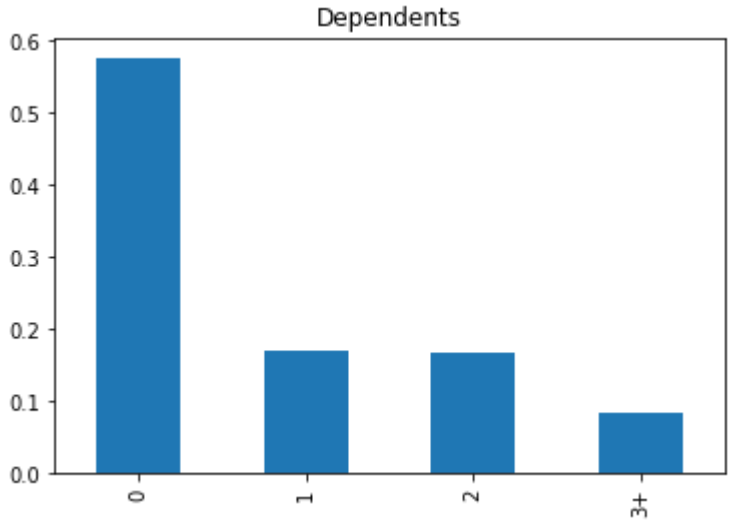
Out[38]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [39]:
```
train['Dependents'].value_counts(normalize=True)*100
```

Out[39]:
```
0     57.595993
1     17.028381
2     16.861436
3+     8.514190
Name: Dependents, dtype: float64
```

In [40]:
```
train['Dependents'].value_counts(normalize=True).plot.bar(title="Dependents")
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd5063de48>
```



In [41]:
```
#Analysis on "Education" variable
```

In [42]:
```
train["Education"].count()
```

Out[42]:
```
614
```

In [43]:
```
train["Education"].value_counts()
```

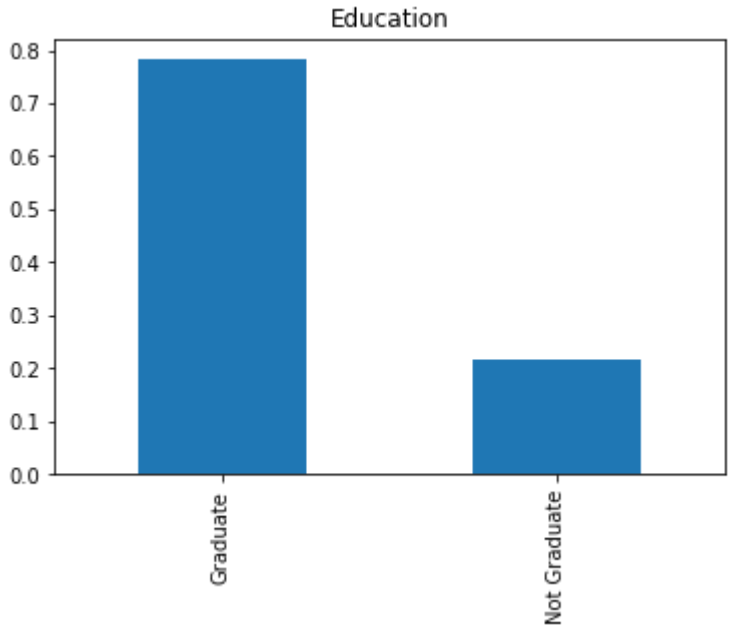
Out[43]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

In [44]:
```
train["Education"].value_counts(normalize=True)*100
```

Out[44]:
```
Graduate        78.175896
Not Graduate    21.824104
Name: Education, dtype: float64
```

In [45]:
```
train["Education"].value_counts(normalize=True).plot.bar(title = "Education")
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd50757898>
```



In [46]:
```
#Analysis on "Property_Area" variable
```

In [47]:
```
train["Property_Area"].count()
```

Out[47]:
```
614
```

In [48]:
```
train["Property_Area"].value_counts()
```

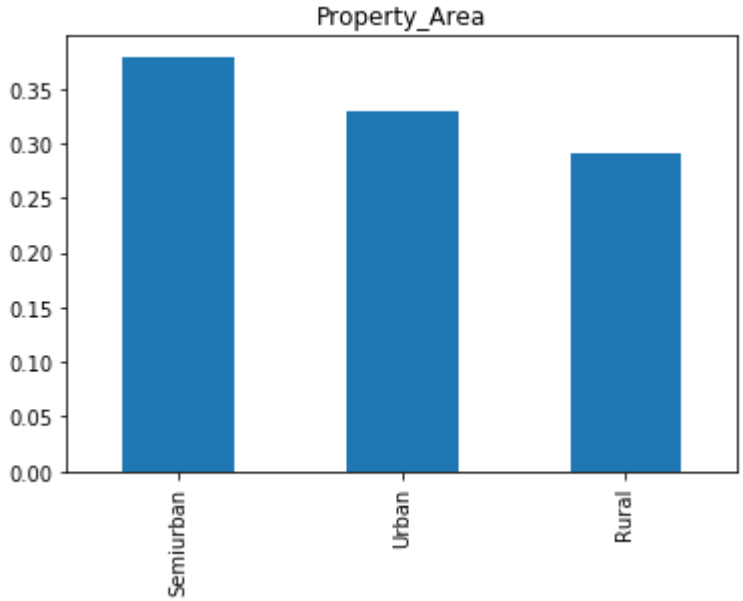
Out[48]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [49]:
```
train["Property_Area"].value_counts(normalize=True)*100
```

Out[49]:
```
Semiurban    37.947883
Urban        32.899023
Rural        29.153094
Name: Property_Area, dtype: float64
```

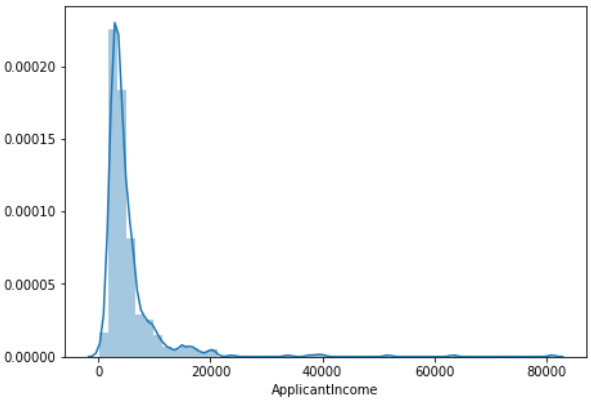
In [50]:
```
train["Property_Area"].value_counts(normalize=True).plot.bar(title="Property_Area
```

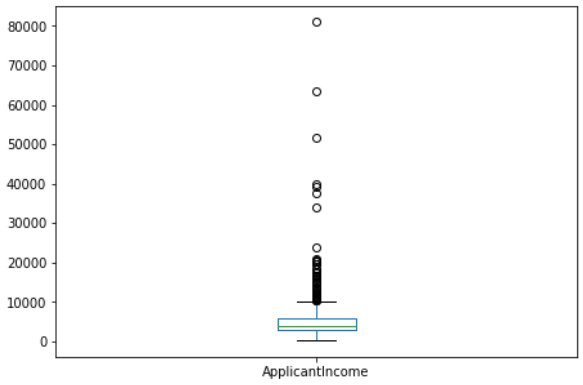
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd50835940>



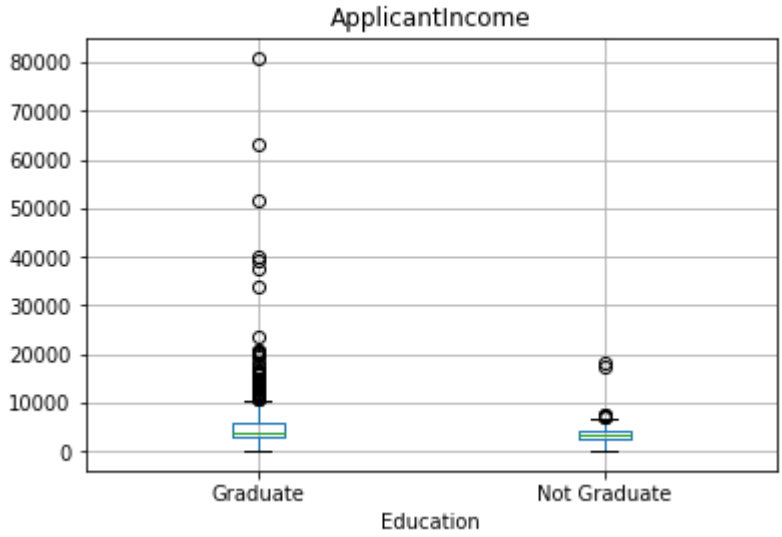
In [51]:
```
#"ApplicantIncome" distribution
```

In [52]:
```
plt.figure(1)
plt.subplot(121)
sns.distplot(train["ApplicantIncome"]);

plt.subplot(122)
train["ApplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```



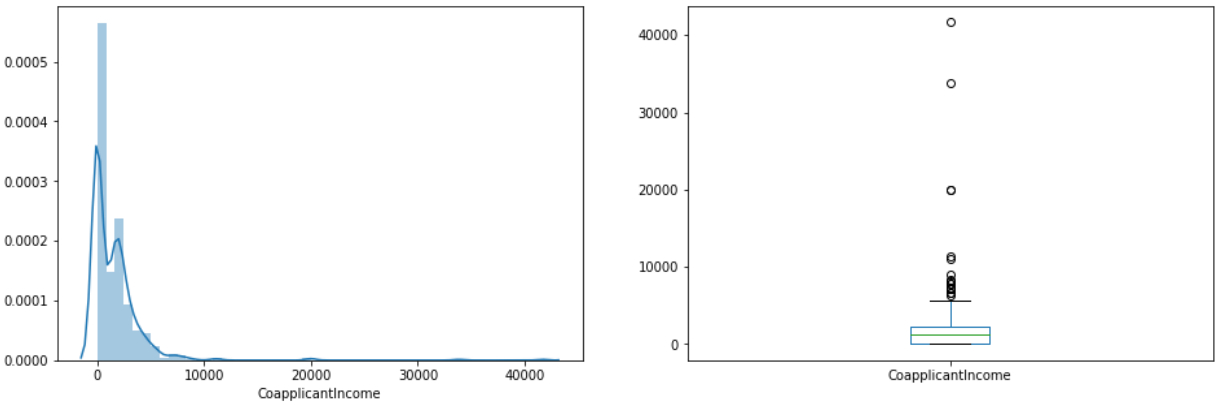


In [53]:
```
train.boxplot(column='ApplicantIncome',by="Education" )
plt.suptitle(" ")
plt.show()
```



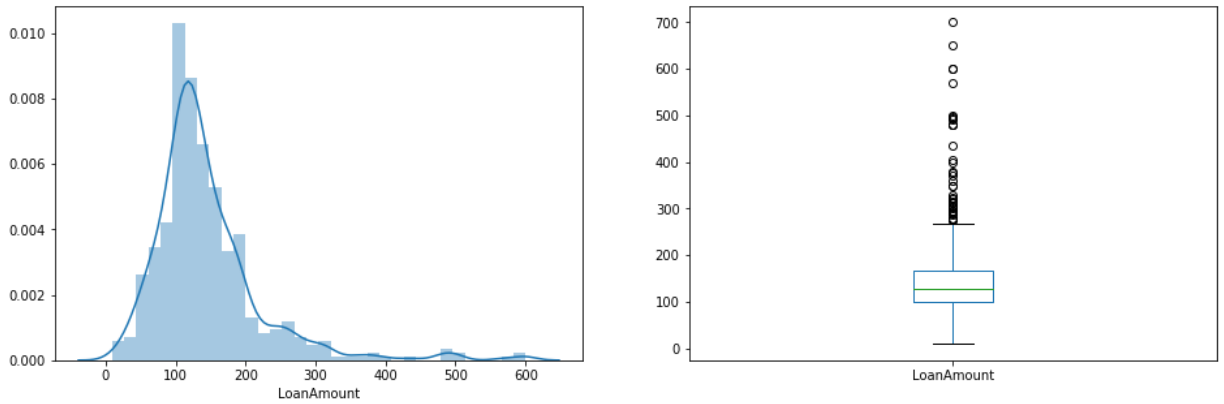
In [54]:
```
#"CoapplicantIncome" distribution
```

In [55]:
```
plt.figure(1)
plt.subplot(121)
sns.distplot(train["CoapplicantIncome"]);

plt.subplot(122)
train["CoapplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```



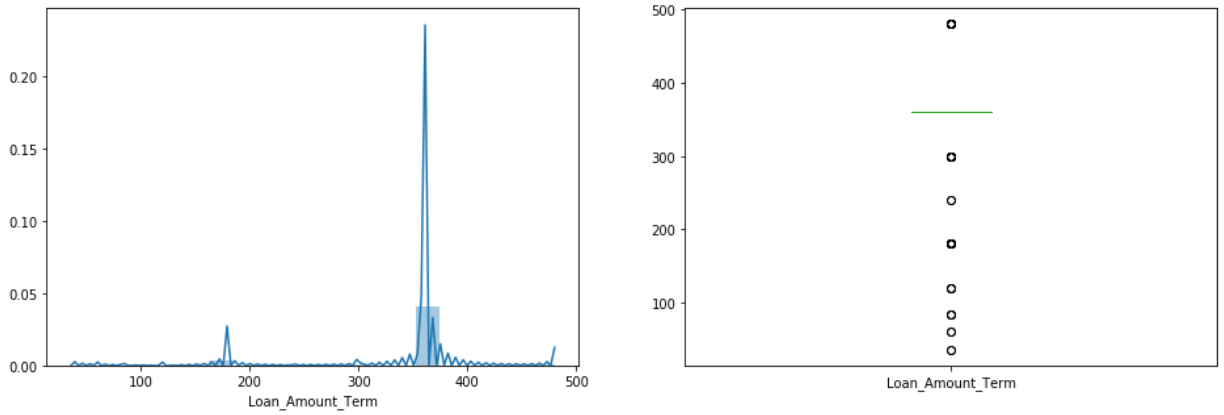
In [56]:
```
#distribution of "LoanAmount" variable
```

In [57]:
```
plt.figure(1)
plt.subplot(121)
df=train.dropna()
sns.distplot(df['LoanAmount']);

plt.subplot(122)
train['LoanAmount'].plot.box(figsize=(16,5))

plt.show()
```



In [58]:
```
#distribution of "LoanAmountTerm" variable
```

In [59]:
```
plt.figure(1)
plt.subplot(121)
df = train.dropna()
sns.distplot(df["Loan_Amount_Term"]);

plt.subplot(122)
df["Loan_Amount_Term"].plot.box(figsize=(16,5))
plt.show()
```

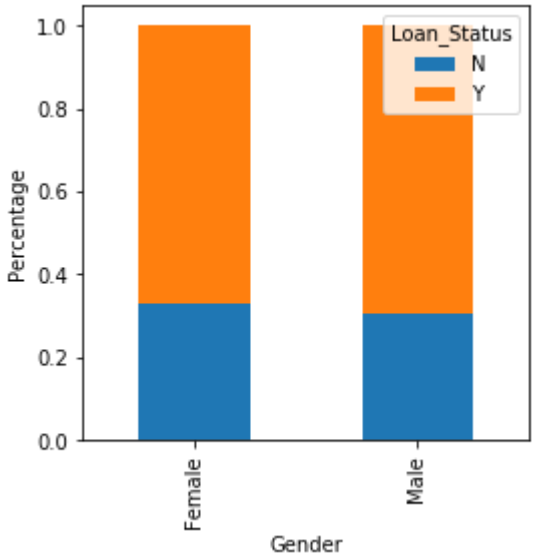


In [60]:
```
#bivariate analysis
```

In [61]:
```
#Relation between "Loan_Status" and "Gender"
```

In [62]:
```
print(pd.crosstab(train["Gender"],train["Loan_Status"]))
Gender = pd.crosstab(train["Gender"],train["Loan_Status"])
Gender.div(Gender.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figs
plt.xlabel("Gender")
plt.ylabel("Percentage")
plt.show()
```

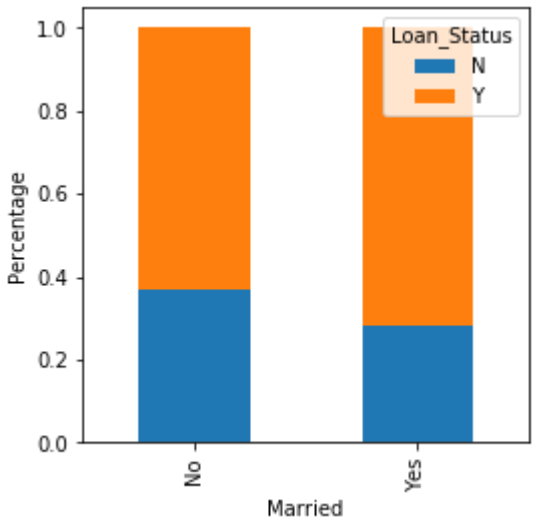
```
Loan_Status    N    Y
Gender
Female        37   75
Male         150  339
```



In [63]:
```
#Relation between "Loan_Status" and "Married"
```

In [64]:
```
print(pd.crosstab(train["Married"],train["Loan_Status"]))
Married=pd.crosstab(train["Married"],train["Loan_Status"])
Married.div(Married.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,fi
plt.xlabel("Married")
plt.ylabel("Percentage")
plt.show()
```

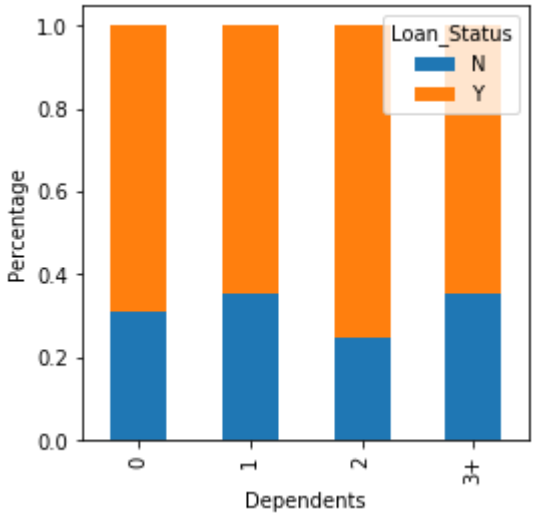
```
Loan_Status    N    Y
Married
No            79  134
Yes          113  285
```



In [65]:
```
#Relation between "Loan_Status" and "Dependents"
```

In [66]:
```
print(pd.crosstab(train['Dependents'],train["Loan_Status"]))
Dependents = pd.crosstab(train['Dependents'],train["Loan_Status"])
Dependents.div(Dependents.sum(1).astype(float),axis=0).plot(kind="bar",stacked=Tr
plt.xlabel("Dependents")
plt.ylabel("Percentage")
plt.show()
```

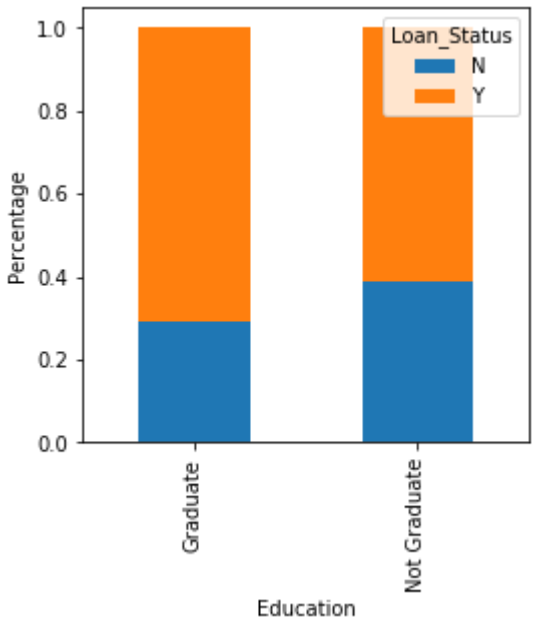
```
Loan_Status    N    Y
Dependents
0            107  238
1             36   66
2             25   76
3+            18   33
```



In [67]:
```
#Relation between "Loan_Status" and "Education"
```

In [68]:
```
print(pd.crosstab(train["Education"],train["Loan_Status"]))
Education = pd.crosstab(train["Education"],train["Loan_Status"])
Education.div(Education.sum(1).astype(float),axis=0).plot(kind="bar",stacked=Tru
plt.xlabel("Education")
plt.ylabel("Percentage")
plt.show()
```

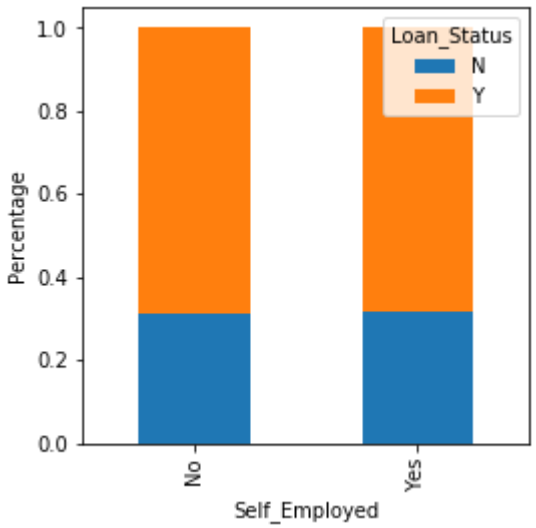
```
Loan_Status     N    Y
Education
Graduate      140  340
Not Graduate   52   82
```



In [69]:
```
#Relation between "Loan_Status" and "Self_Employed"
```

In [70]:
```
print(pd.crosstab(train["Self_Employed"],train["Loan_Status"]))
SelfEmployed = pd.crosstab(train["Self_Employed"],train["Loan_Status"])
SelfEmployed.div(SelfEmployed.sum(1).astype(float),axis=0).plot(kind="bar",stacke
plt.xlabel("Self_Employed")
plt.ylabel("Percentage")
plt.show()
```

```
Loan_Status      N    Y
Self_Employed
No             157  343
Yes             26   56
```



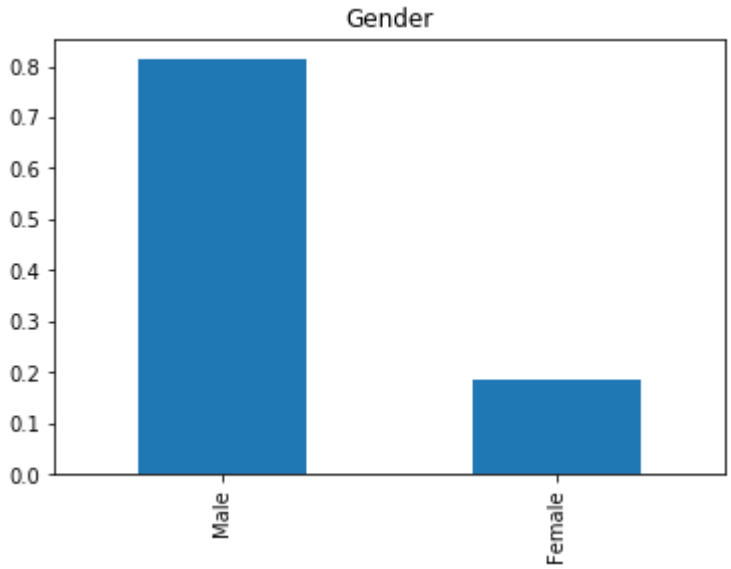
In [71]:
```
#Relation between "Loan_Status" and "Credit_History"
```

In [72]:
```
print(pd.crosstab(train["Credit_History"],train["Loan_Status"]))
CreditHistory = pd.crosstab(train["Credit_History"],train["Loan_Status"])
CreditHistory.div(CreditHistory.sum(1).astype(float),axis=0).plot(kind="bar",sta
plt.xlabel("Credit_History")
plt.ylabel("Percentage")
plt.show()
```

```
Loan_Status      N    Y
Credit_History
0.0             82    7
1.0             97  378
```

In [73]:
```
train.Gender.value_counts(normalize=True).plot(kind = 'bar', title = "Gender")
```

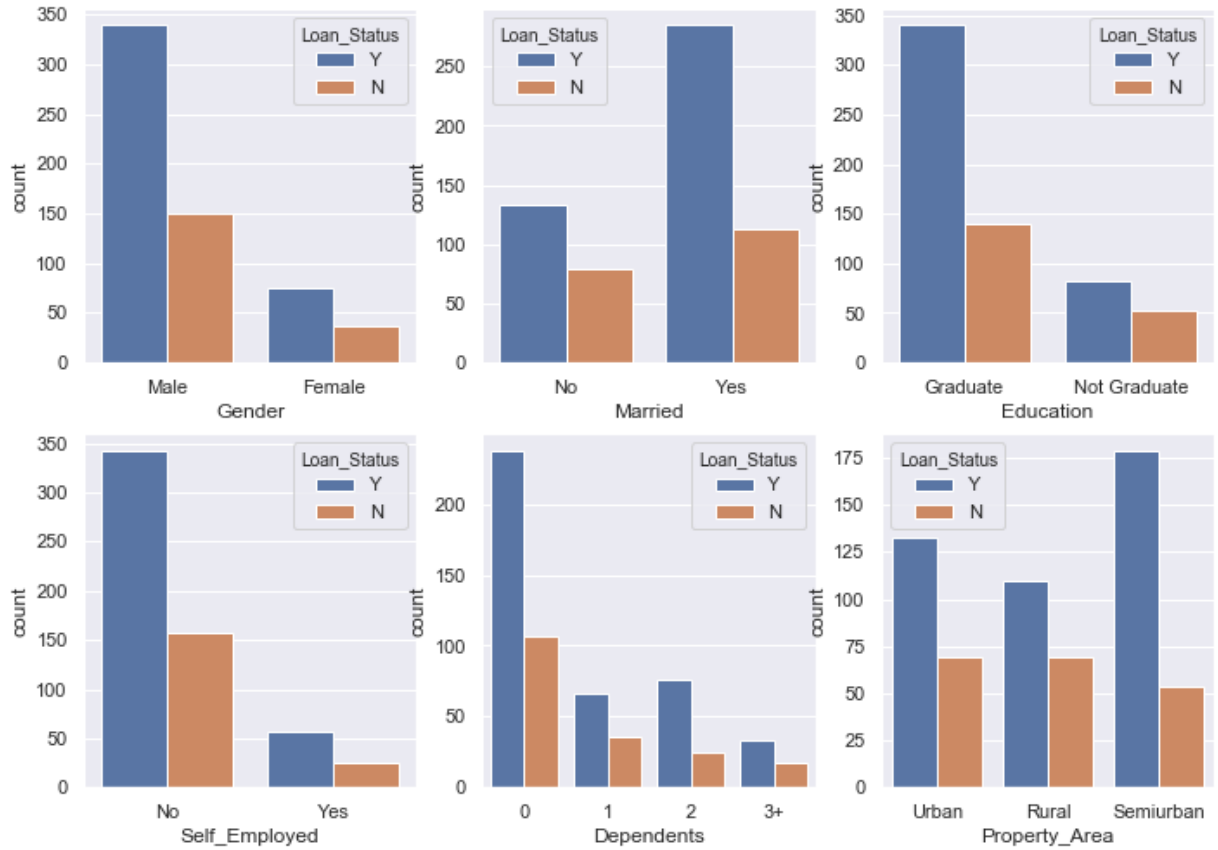
Out[73]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd50b630f0>
```



In [74]:
```
import seaborn as sns
```

In [75]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.subplot(231)
sns.countplot(x="Gender", hue='Loan_Status', data=train)
plt.subplot(232)
sns.countplot(x="Married", hue='Loan_Status', data=train)
plt.subplot(233)
sns.countplot(x="Education", hue='Loan_Status', data=train)
plt.subplot(234)
sns.countplot(x="Self_Employed", hue='Loan_Status', data=train)
plt.subplot(235)
sns.countplot(x="Dependents", hue='Loan_Status', data=train)
plt.subplot(236)
sns.countplot(x="Property_Area", hue='Loan_Status', data=train)
```

Out[75]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd511c3b70>
```



In [76]:
```
train['Dependents'].replace('3+',3,inplace=True)
test['Dependents'].replace('3+',3,inplace=True)
train['Loan_Status'].replace('N', 0,inplace=True)
train['Loan_Status'].replace('Y', 1,inplace=True)
```

In [77]:
```
matrix = train.corr()
f, ax = plt.subplots(figsize=(10, 12))
sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu",annot=True);
```

In [78]:
```
#missing values
```

In [79]:
```
train.isnull().sum()
```

Out[79]:
```
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 [80]:
```
train["Gender"].fillna(train["Gender"].mode()[0],inplace=True)
train["Married"].fillna(train["Married"].mode()[0],inplace=True)
train['Dependents'].fillna(train["Dependents"].mode()[0],inplace=True)
train["Self_Employed"].fillna(train["Self_Employed"].mode()[0],inplace=True)
train["Credit_History"].fillna(train["Credit_History"].mode()[0],inplace=True)
```

In [81]:
```
train["Loan_Amount_Term"].value_counts()
```

Out[81]:
```
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 [82]:
```
train["Loan_Amount_Term"].fillna(train["Loan_Amount_Term"].mode()[0],inplace=Tru
```

In [83]:
```
train["Loan_Amount_Term"].value_counts()
```

Out[83]:
```
360.0    526
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 [84]:
```
train["LoanAmount"].fillna(train["LoanAmount"].median(),inplace=True)
```

In [85]:
```
train.isnull().sum()
```

Out[85]:
```
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 [86]:
```
test.isnull().sum()
```

Out[86]:
```
Loan_ID              0
Gender              11
Married              0
Dependents          10
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     6
Credit_History      29
Property_Area        0
dtype: int64
```

In [87]:
```
test["Gender"].fillna(test["Gender"].mode()[0],inplace=True)
test['Dependents'].fillna(test["Dependents"].mode()[0],inplace=True)
test["Self_Employed"].fillna(test["Self_Employed"].mode()[0],inplace=True)
test["Loan_Amount_Term"].fillna(test["Loan_Amount_Term"].mode()[0],inplace=True)
test["Credit_History"].fillna(test["Credit_History"].mode()[0],inplace=True)
test["LoanAmount"].fillna(test["LoanAmount"].median(),inplace=True)
```

In [88]:
```
test.isnull().sum()
```

Out[88]:
```
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
dtype: int64
```

In [89]:
```
#total income:we will combine the applicant and co-applicant income, since if the
#will also be high
```

In [90]:
```
train["TotalIncome"]=train["ApplicantIncome"]+train["CoapplicantIncome"]
```

In [91]:
```
train[["TotalIncome"]].head()
```

Out[91]:

	TotalIncome
0	5849.0
1	6091.0
2	3000.0
3	4941.0
4	6000.0

In [92]:
```
test["TotalIncome"]=test["ApplicantIncome"]+test["CoapplicantIncome"]
```

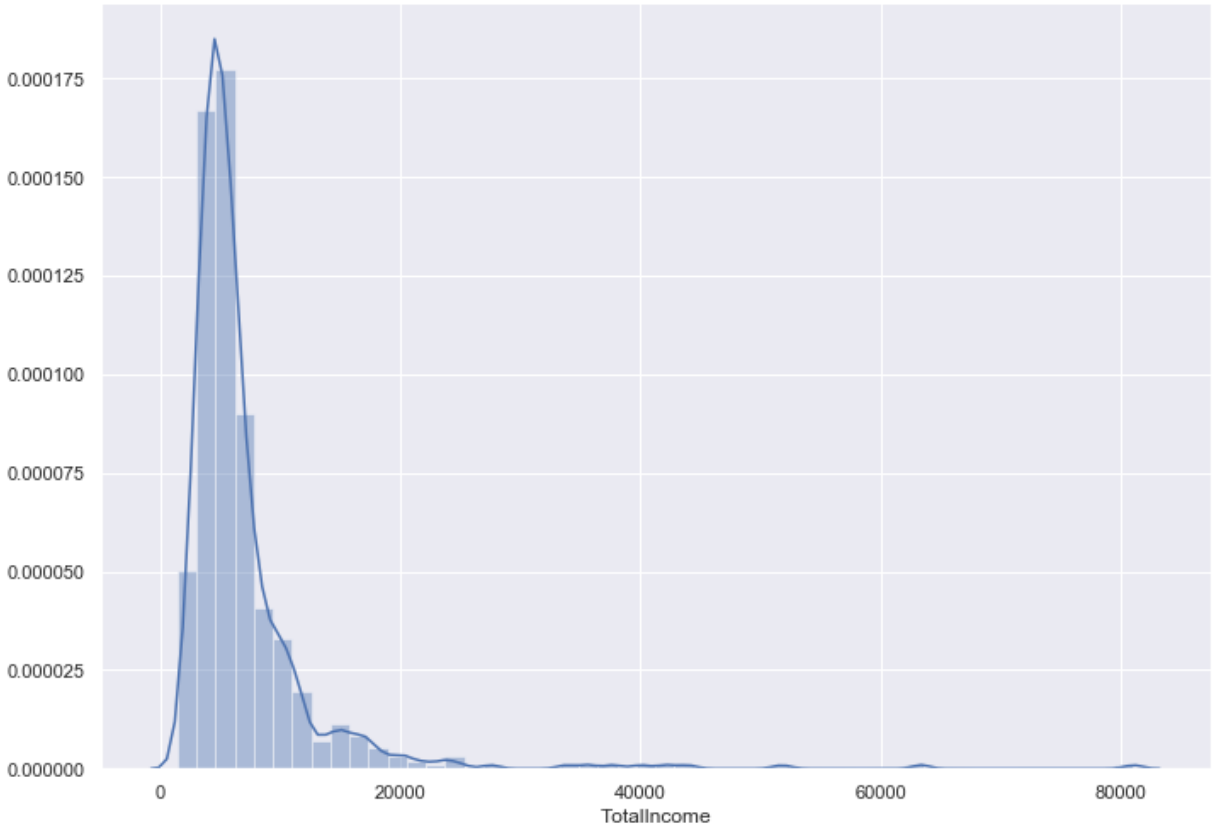
In [93]:
```
test[["TotalIncome"]].head()
```

Out[93]:

	TotalIncome
0	5720
1	4576
2	6800
3	4886
4	3276

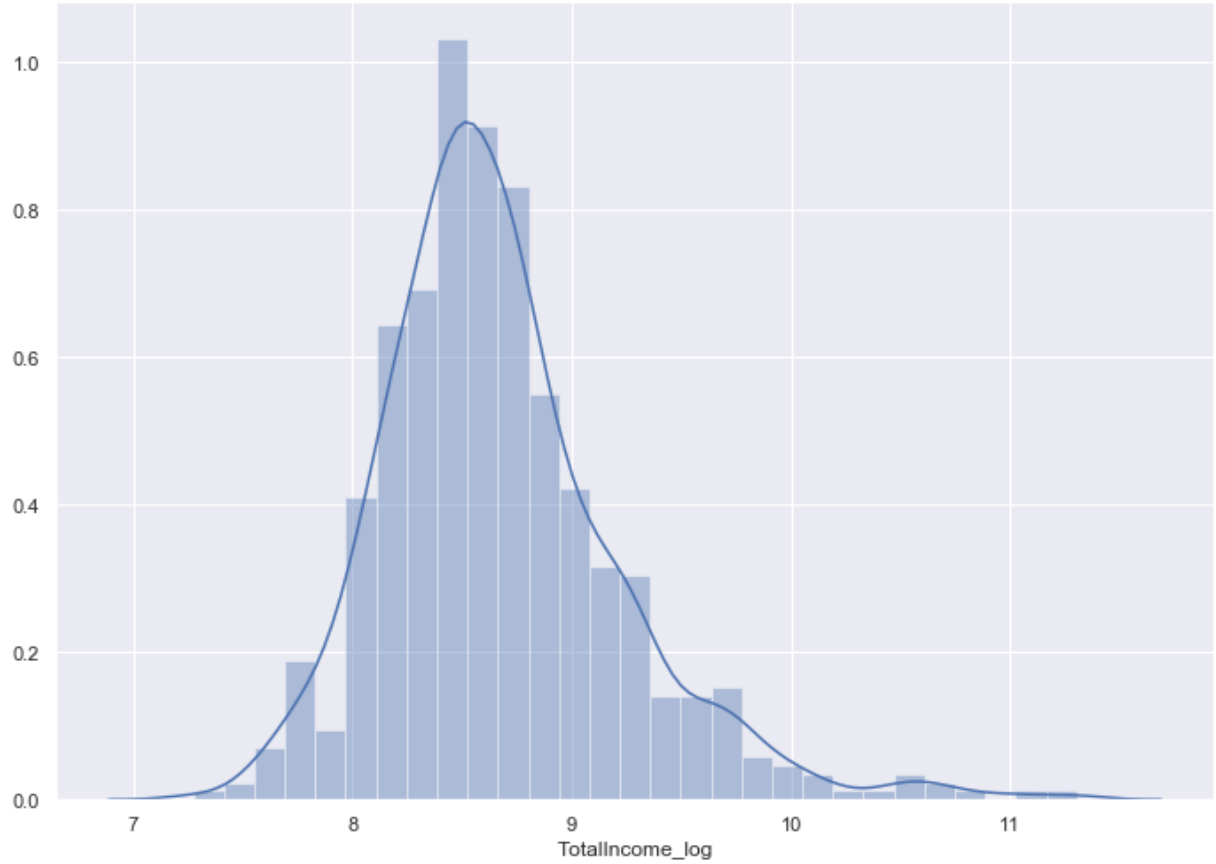
In [94]:
```
sns.distplot(train["TotalIncome"])
```

Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd5138d898>
```



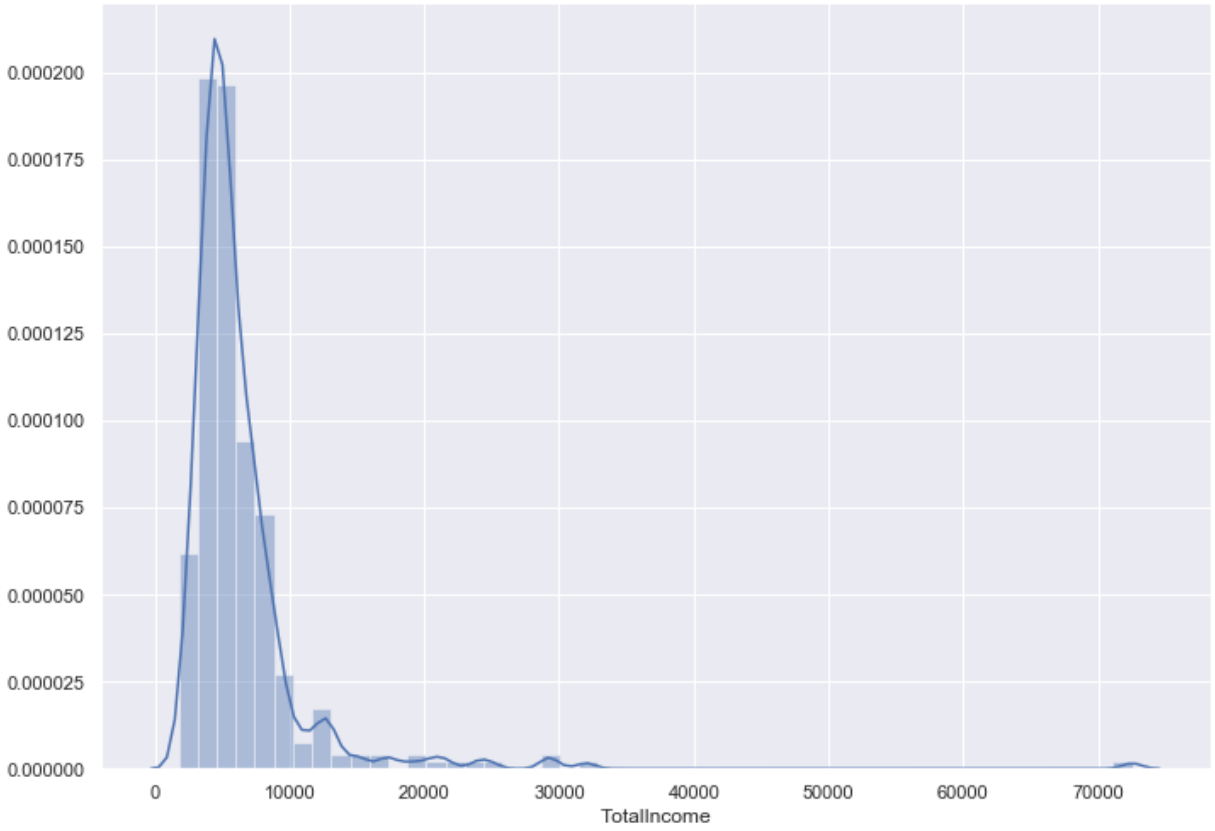
In [95]:
```
train["TotalIncome_log"]=np.log(train["TotalIncome"])
sns.distplot(train["TotalIncome_log"])
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd5094cba8>



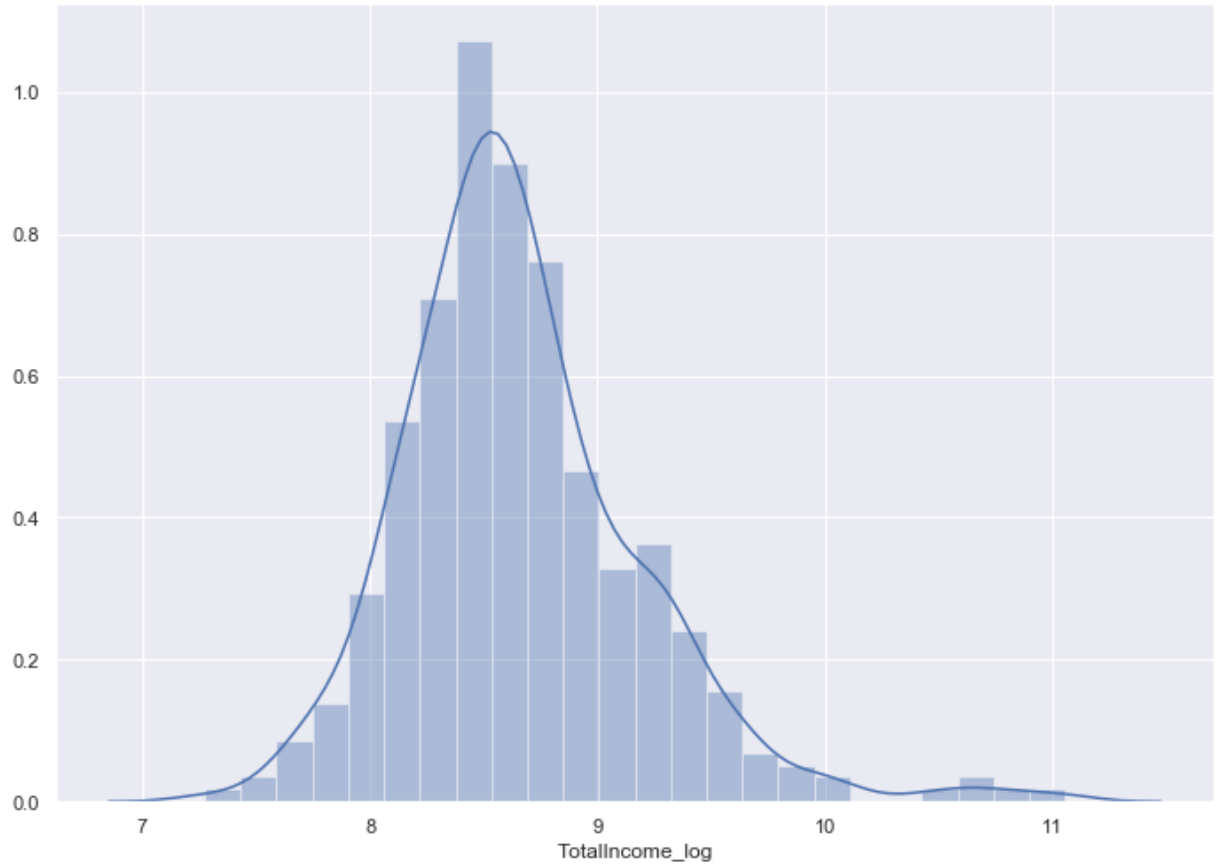
In [96]:
```
sns.distplot(test["TotalIncome"])
```

Out[96]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd50b43c50>
```



In [97]:
```
test["TotalIncome_log"] = np.log(train["TotalIncome"])
sns.distplot(test["TotalIncome_log"])
```

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd5108f4e0>



In [98]:
```
#EMI: the amount to be paid every month. if the emi is high people may not be abl
#the EMI by taking the ratio of loan amount with respect to loan amount term.
```

In [99]:
```
train["EMI"]=train["LoanAmount"]/train["Loan_Amount_Term"]
test["EMI"]=test["LoanAmount"]/test["Loan_Amount_Term"]
```

In [100]:
```
train[["EMI"]].head()
```

Out[100]:

	EMI
0	0.355556
1	0.355556
2	0.183333
3	0.333333
4	0.391667

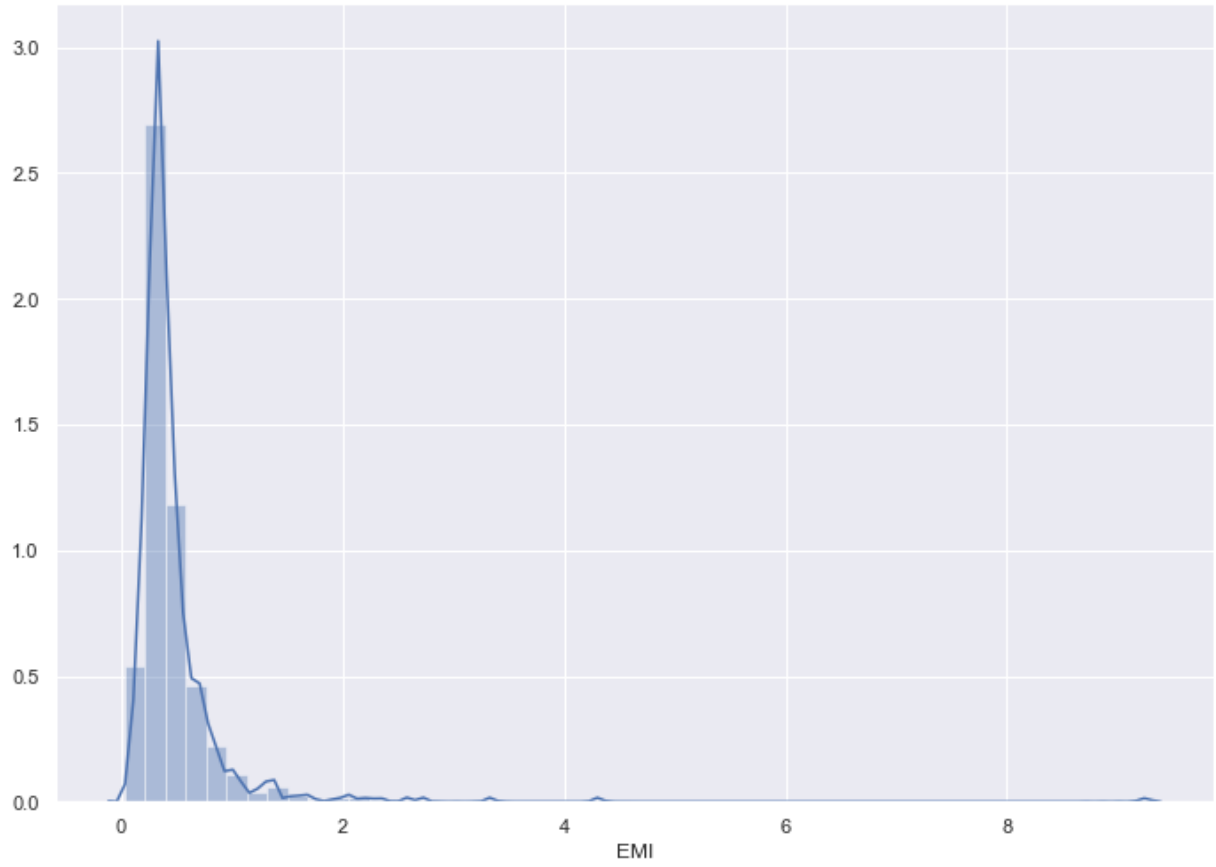
In [101]:
```
test[["EMI"]].head()
```

Out[101]:

	EMI
0	0.305556
1	0.350000
2	0.577778
3	0.277778
4	0.216667

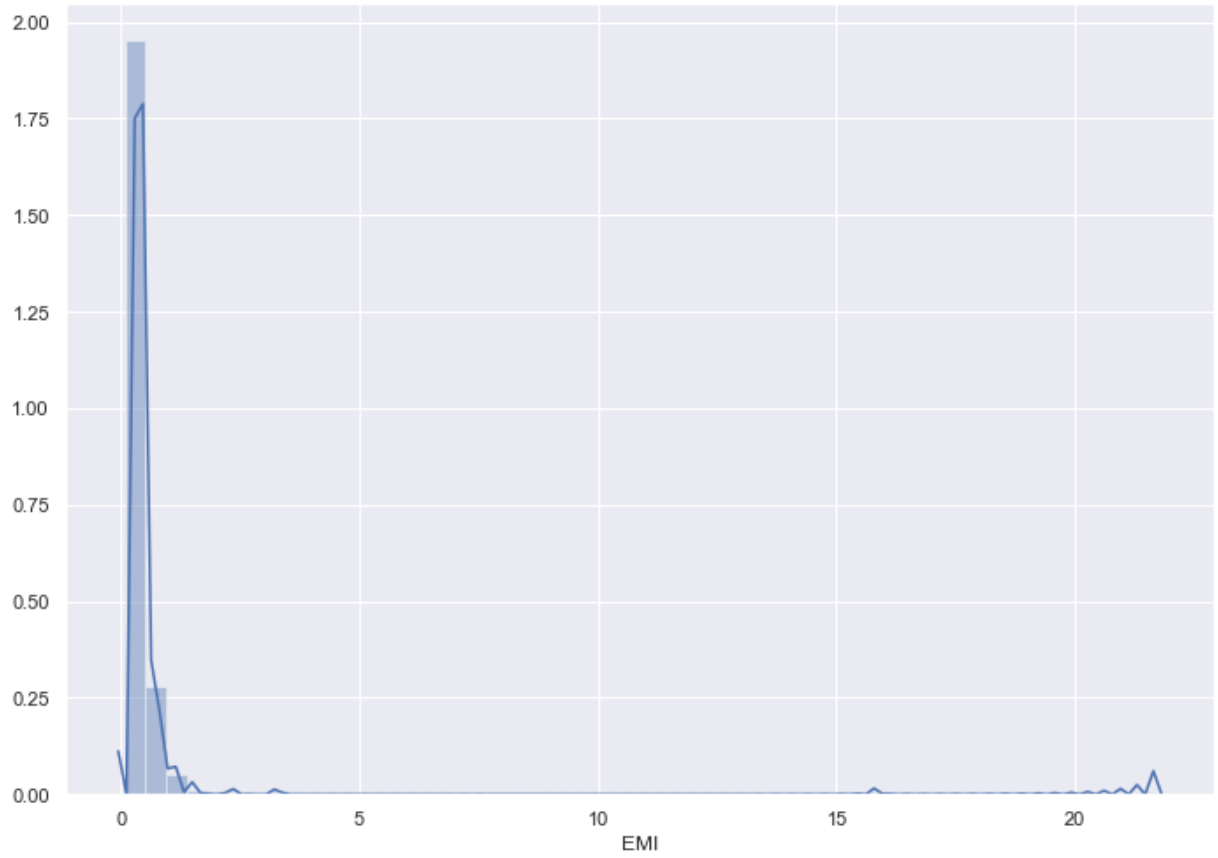
In [102]:
```
sns.distplot(train["EMI"])
```

Out[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd50c10550>
```



In [103]:
```
sns.distplot(test["EMI"])
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd50ceada0>
```



In [104]:
```
#balance income: it is the income left after the EMI has been paid.
```

In [105]:
```
train["Balance_Income"] = train["TotalIncome"]-train["EMI"]*1000 # To make the u
test["Balance_Income"] = test["TotalIncome"]-test["EMI"]
```

In [106]:
```
train[["Balance_Income"]].head()
```

Out[106]:

	Balance_Income
0	5493.444444
1	5735.444444
2	2816.666667
3	4607.666667
4	5608.333333

In [107]:
```
test[["Balance_Income"]].head()
```

Out[107]:

	Balance_Income
0	5719.694444
1	4575.650000
2	6799.422222
3	4885.722222
4	3275.783333

In [108]:
```
#now we have to drop the variables we used to create the new variables, since the
#and also to remove the noise/null values.
```

In [109]:
```
train=train.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan_Amount
```

In [110]:
```
train.head()
```

Out[110]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Ar
0	LP001002	Male	No	0	Graduate	No	1.0	Urb
1	LP001003	Male	Yes	1	Graduate	No	1.0	Ru
2	LP001005	Male	Yes	0	Graduate	Yes	1.0	Urb
3	LP001006	Male	Yes	0	Not Graduate	No	1.0	Urb
4	LP001008	Male	No	0	Graduate	No	1.0	Urb

In [111]:
```
test = test.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan_Amount
```

In [112]:
```
test.head()
```

Out[112]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Ar
0	LP001015	Male	Yes	0	Graduate	No	1.0	Urb
1	LP001022	Male	Yes	1	Graduate	No	1.0	Urb
2	LP001031	Male	Yes	2	Graduate	No	1.0	Urb
3	LP001035	Male	Yes	2	Graduate	No	1.0	Urb
4	LP001051	Male	No	0	Not Graduate	No	1.0	Urb

In [113]:
```
#Logistic regression
```

In [114]:
```
train=train.drop("Loan_ID",axis=1)
test=test.drop("Loan_ID",axis=1)
```

In [115]:
```
train.head(3)
```

Out[115]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_S
0	Male	No	0	Graduate	No	1.0	Urban	
1	Male	Yes	1	Graduate	No	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	1.0	Urban	

In [116]:
```
test.head(3)
```

Out[116]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	TotalInc
0	Male	Yes	0	Graduate	No	1.0	Urban	
1	Male	Yes	1	Graduate	No	1.0	Urban	
2	Male	Yes	2	Graduate	No	1.0	Urban	

In [117]:
```
#dropping thr target variable into another dataset
X=train.drop("Loan_Status",1)
```

In [118]:
```
X.head(2)
```

Out[118]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	TotalInc
0	Male	No	0	Graduate	No	1.0	Urban	58
1	Male	Yes	1	Graduate	No	1.0	Rural	60

In [119]:
```
y=train[["Loan_Status"]]
```

In [120]:
```
y.head(2)
```

Out[120]:

	Loan_Status
0	1
1	0

In [121]:
```
X = pd.get_dummies(X)
```

In [122]:
```
X.head(3)
```

Out[122]:

	Credit_History	TotalIncome	TotalIncome_log	EMI	Balance_Income	Gender_Female	Gende
0	1.0	5849.0	8.674026	0.355556	5493.444444	0	
1	1.0	6091.0	8.714568	0.355556	5735.444444	0	
2	1.0	3000.0	8.006368	0.183333	2816.666667	0	

In [123]:
```
train=pd.get_dummies(train)
test=pd.get_dummies(test)
```

In [124]:
```
train.head(3)
```

Out[124]:

	Credit_History	Loan_Status	TotalIncome	TotalIncome_log	EMI	Balance_Income	Gender_F
0	1.0	1	5849.0	8.674026	0.355556	5493.444444	
1	1.0	0	6091.0	8.714568	0.355556	5735.444444	
2	1.0	1	3000.0	8.006368	0.183333	2816.666667	

3 rows × 21 columns

In [125]:
```
test.head(3)
```

Out[125]:

	Credit_History	TotalIncome	TotalIncome_log	EMI	Balance_Income	Gender_Female	Gende
0	1.0	5720	8.674026	0.305556	5719.694444	0	
1	1.0	4576	8.714568	0.350000	4575.650000	0	
2	1.0	6800	8.006368	0.577778	6799.422222	0	

In [126]:
```
from sklearn.model_selection import train_test_split
```

In [127]:
```
x_train,x_cv,y_train,y_cv=train_test_split(X,y,test_size=0.3,random_state=1)
```

In [128]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [129]:
```
logistic_model = LogisticRegression(random_state=1)
```

In [130]:
```
logistic_model.fit(x_train,y_train)
```

Out[130]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=1, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [131]:
```
pred_cv_logistic=logistic_model.predict(x_cv)
```

In [132]:
```
score_logistic =accuracy_score(pred_cv_logistic,y_cv)*100
```

In [133]:
```
score_logistic
```

Out[133]:
```
78.91891891891892
```

In [134]:
```
pred_test_logistic = logistic_model.predict(test)
```

In [138]:
```
#decision tree
```

In [139]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [140]:
```
tree_model = DecisionTreeClassifier(random_state=1)
```

In [141]:
```
tree_model.fit(x_train,y_train)
```

Out[141]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=1, splitter='best')
```

In [142]:
```
pred_cv_tree=tree_model.predict(x_cv)
```

In [143]:
```
score_tree =accuracy_score(pred_cv_tree,y_cv)*100
```

In [144]:
```
score_tree
```

Out[144]:
```
70.27027027027027
```

In [145]:
```
pred_test_tree = tree_model.predict(test)
```

In [146]:
```
#random forest
```

In [147]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [148]:
```
forest_model = RandomForestClassifier(random_state=1,max_depth=10,n_estimators=5
```

In [149]:
```
forest_model.fit(x_train,y_train)
```

Out[149]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=10, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

In [150]:
```
pred_cv_forest=forest_model.predict(x_cv)
```

In [151]:
```
score_forest = accuracy_score(pred_cv_forest,y_cv)*100
```

In [152]:
```
score_forest
```

Out[152]:
```
77.29729729729729
```

In [153]:
```
pred_test_forest=forest_model.predict(test)
```

In [154]:
```
#random forest with grid search
```

In [155]:
```
from sklearn.model_selection import GridSearchCV
```

In [156]:
```
paramgrid = {'max_depth': list(range(1,20,2)),'n_estimators':list(range(1,200,20
```

In [157]:
```
grid_search = GridSearchCV(RandomForestClassifier(random_state=1),paramgrid)
```

In [158]:
```
grid_search.fit(x_train,y_train)
```

Out[158]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=Non
e,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False, random_state=1,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
                         'n_estimators': [1, 21, 41, 61, 81, 101, 121, 141, 16
1,
                                          181]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [159]:
```
grid_search.best_estimator_
```

Out[159]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=5, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=41,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

In [160]:
```
#building a model using the optimized values
grid_forest_model = RandomForestClassifier(random_state=1,max_depth=3,n_estimato
```

In [161]:
```
grid_forest_model.fit(x_train,y_train)
```

Out[161]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=101,
                       n_jobs=None, oob_score=False, random_state=1, verbose=0,
                       warm_start=False)
```

In [162]:
```
pred_grid_forest = grid_forest_model.predict(x_cv)
```

In [163]:
```
score_grid_forest = accuracy_score(pred_grid_forest,y_cv)*100
```

In [164]:
```
score_grid_forest
```

Out[164]:
```
78.37837837837837
```

In [165]:
```
pred_grid_forest_test = grid_forest_model.predict(test)
```

In [166]:
```
#XGBoost
```

In [169]:
```
pip install xgboost
```

```
Collecting xgboost
  Downloading https://files.pythonhosted.org/packages/b7/5f/857d1fac0c0abca187fad4ca03eb9de3aeb2564cb21d42e2d1645a373d19/xgboost-1.4.0-py3-none-win_amd64.whl (https://files.pythonhosted.org/packages/b7/5f/857d1fac0c0abca187fad4ca03eb9de3aeb2564cb21d42e2d1645a373d19/xgboost-1.4.0-py3-none-win_amd64.whl) (97.8MB)
Requirement already satisfied: numpy in c:\users\indra\anaconda3\lib\site-packages (from xgboost) (1.16.4)
Requirement already satisfied: scipy in c:\users\indra\anaconda3\lib\site-packages (from xgboost) (1.2.1)
Installing collected packages: xgboost
Successfully installed xgboost-1.4.0
Note: you may need to restart the kernel to use updated packages.
```

In [170]:
```
from xgboost import XGBClassifier
```

In [171]:
```
xgb_model = XGBClassifier(n_estimators=50,max_depth=4)
```

In [172]:
```
xgb_model.fit(x_train,y_train)
```

```
[17:31:21] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[172]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=50, n_jobs=8, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', use_label_encoder=True,
              validate_parameters=1, verbosity=None)
```

In [173]:
```
pred_xgb=xgb_model.predict(x_cv)
```

In [174]:
```
score_xgb = accuracy_score(pred_xgb,y_cv)*100
```

In [175]:
```
score_xgb
```

Out[175]:
```
77.29729729729729
```

In [176]:
```
#Finally,
# Logistic regression model : 78.918 % accuracy
# Decision tree model: 70.270 % accuracy
# Random forest model: 77.297 % accuracy
# Random forest with grid search: 78.378 % accuracy
# XGBClassifier model: 77.297 % accuracy
```

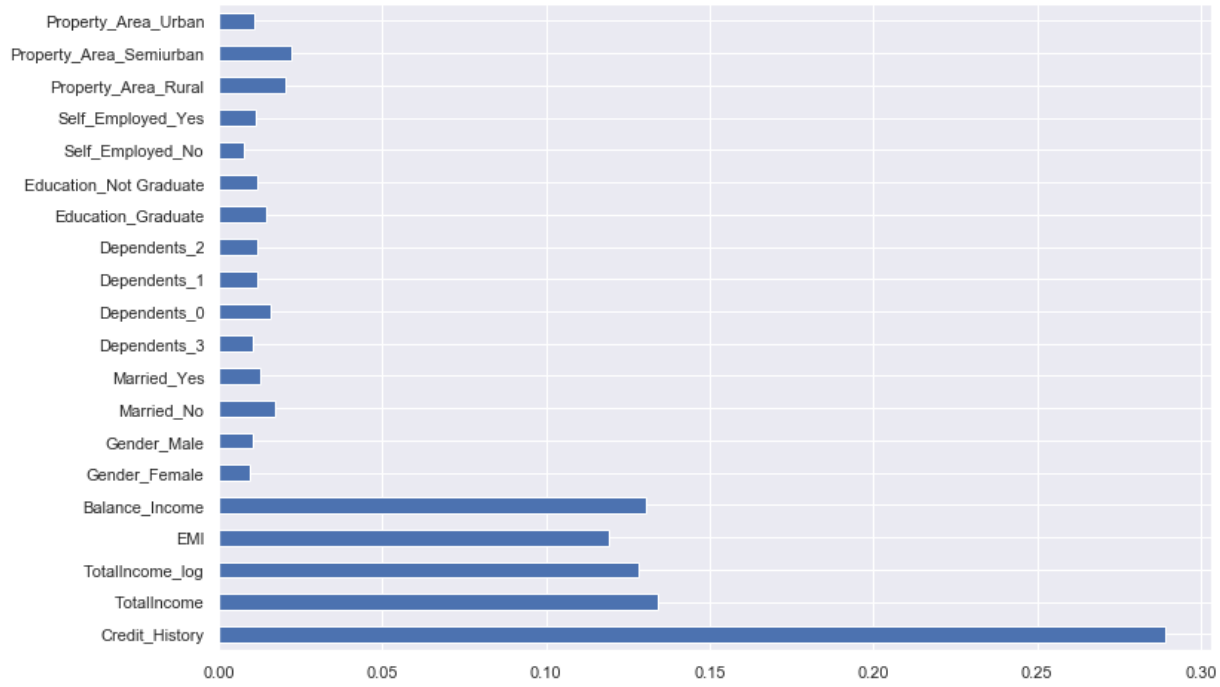
In [177]:
```
# from the above results, the logistic regression model has the highest accuracy
```

In [178]:
```
#now we can find which feature is the most important for deciding the loan status
#the random forest model.
```

In [179]:
```
importances = pd.Series(forest_model.feature_importances_,index=X.columns)
```

In [180]:
```
importances.plot(kind='barh', figsize=(12,8))
```

Out[180]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd5398bcf8>
```



In [181]:
```
#from the above graph it is understood that credit history plays an important ro
```

In [ ]:
```

```