In [1]:
```
import pandas as pd
import numpy as np                          # For mathematical calculations
import seaborn as sns                       # For data visualization
import matplotlib.pyplot as plt             # For plotting graphs
%matplotlib inline
import warnings                             # To ignore any warnings
```

In [2]:
```
warnings.filterwarnings("ignore")
```

In [3]:
```
train = pd.read_csv("C:/Users/Admin/Desktop/ML/train_u6lujuX_CVtuZ9i.csv")
```

In [4]:
```
test = pd.read_csv("C:/Users/Admin/Desktop/ML/test_Y3wMUE5_7gLdaTN.csv")
```

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

In [6]:
```
test_original=test.copy()
```

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

Out[7]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmoun
t',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

In [8]:
```
test.columns
```

Out[8]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmoun
t',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

In [9]:
```
train.dtypes
```

Out[9]:
```
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 [10]:
```
train.shape, test.shape
```

Out[10]:
```
((614, 13), (368, 12))
```

In [11]:
```python
train['Loan_Status'].value_counts()
```

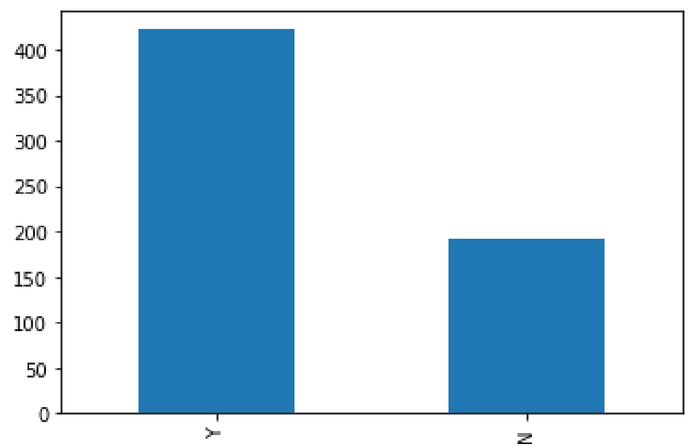
Out[11]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [12]:
```python
# Normalize can be set to True to print proportions instead of number
train['Loan_Status'].value_counts(normalize=True)
```

Out[12]:
```
Y    0.687296
N    0.312704
Name: Loan_Status, dtype: float64
```

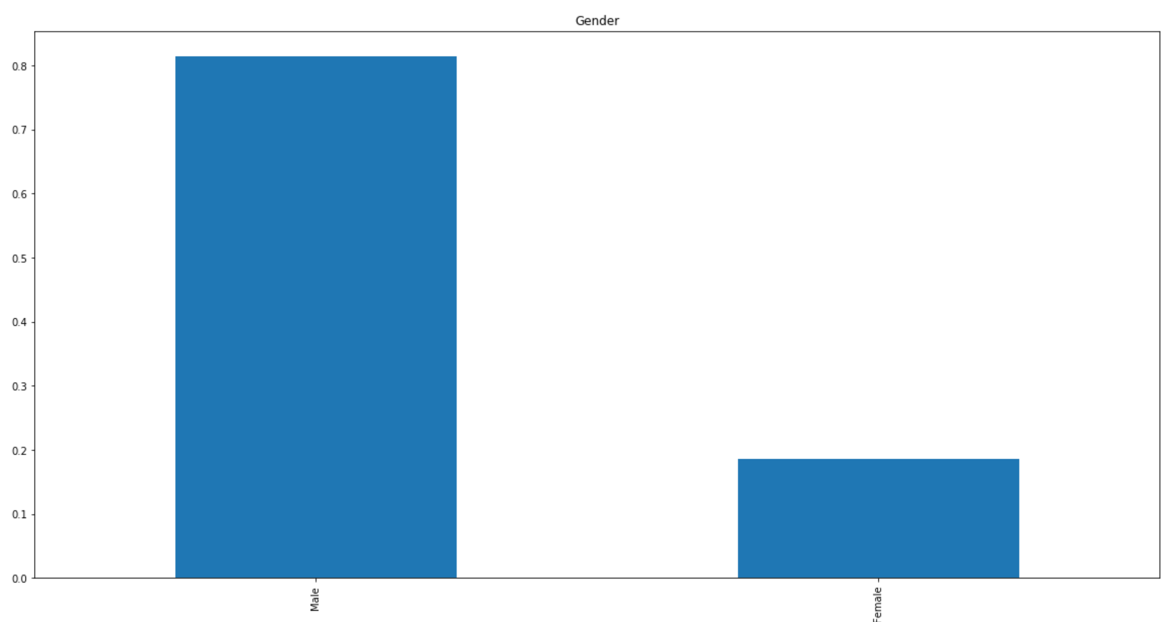
In [13]:
```python
train['Loan_Status'].value_counts().plot.bar()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0xcf30150>



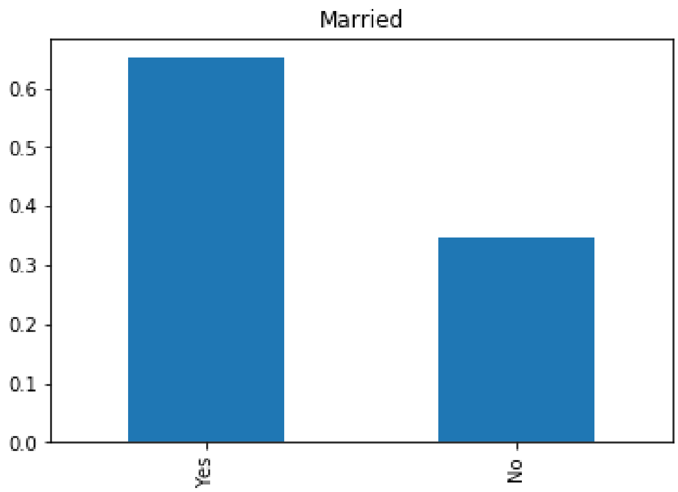
In [14]:
```python
train['Gender'].value_counts(normalize=True).plot.bar(figsize=(20,10), tit
le= 'Gender')
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0xdfbfa10>



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

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0xe029c10>

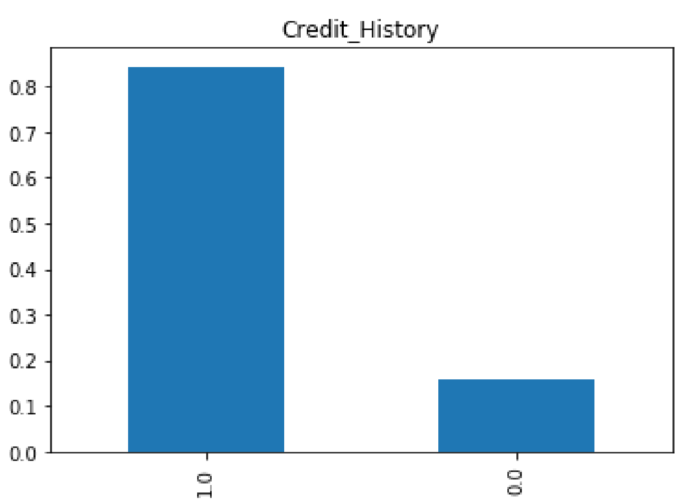


In [16]:
```
train['Self_Employed'].value_counts(normalize=True).plot.bar(title= 'Self_
Employed')
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0xe46b810>

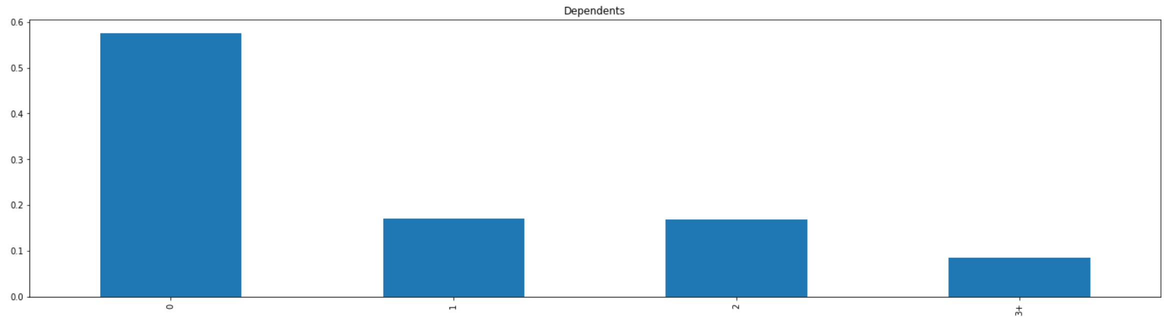
In [17]:
```
train['Credit_History'].value_counts(normalize=True).plot.bar(title= 'Cred
it_History')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0xe05eff0>



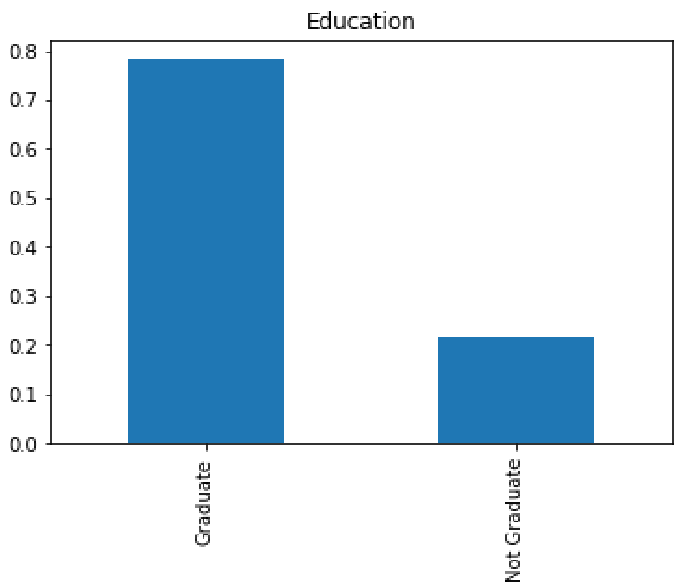
In [18]:
```
train['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6),
title= 'Dependents')
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0xe0a7a30>



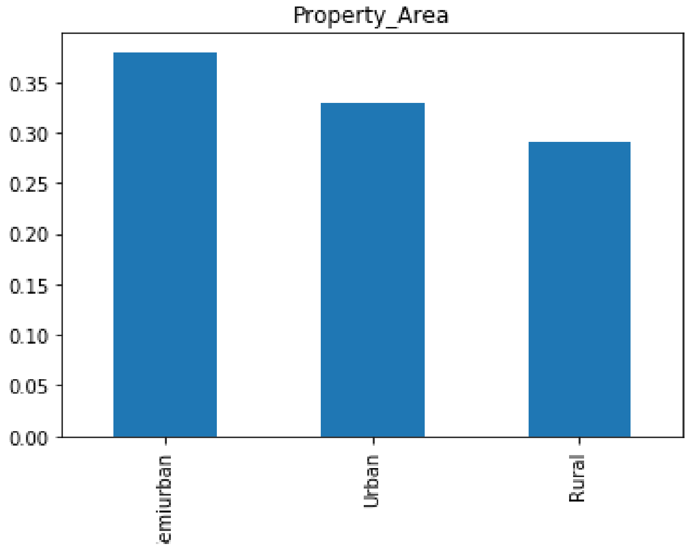
In [19]:
```
train['Education'].value_counts(normalize=True).plot.bar(title= 'Educatio
n')
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0xe0e1cf0>

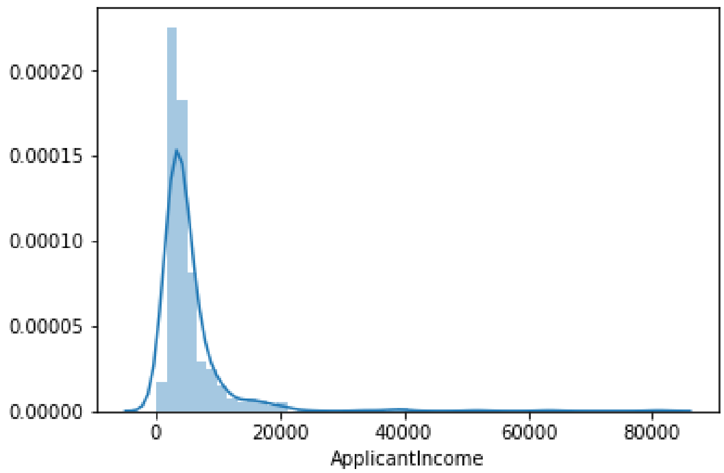


In [20]:
```
train['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Prope
rty_Area')
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0xe401750>

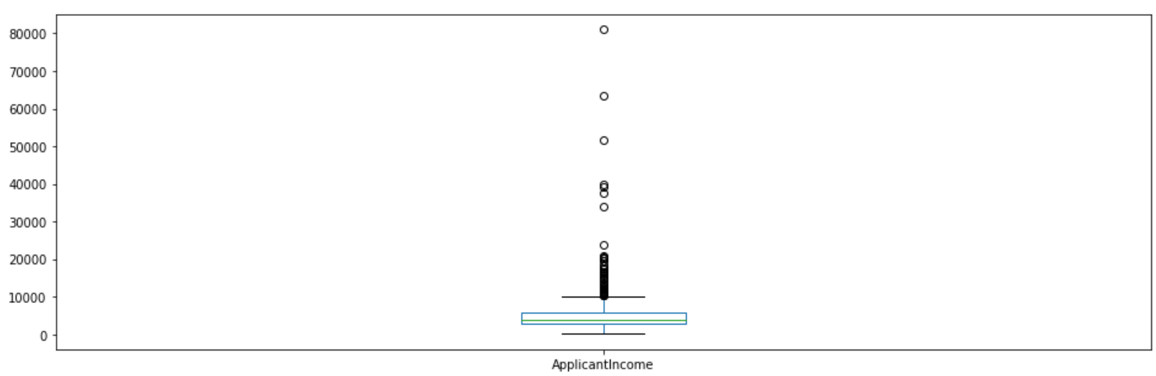


In [21]:
```
sns.distplot(train['ApplicantIncome']);
```



In [22]:
```
train['ApplicantIncome'].plot.box(figsize=(16,5))
```

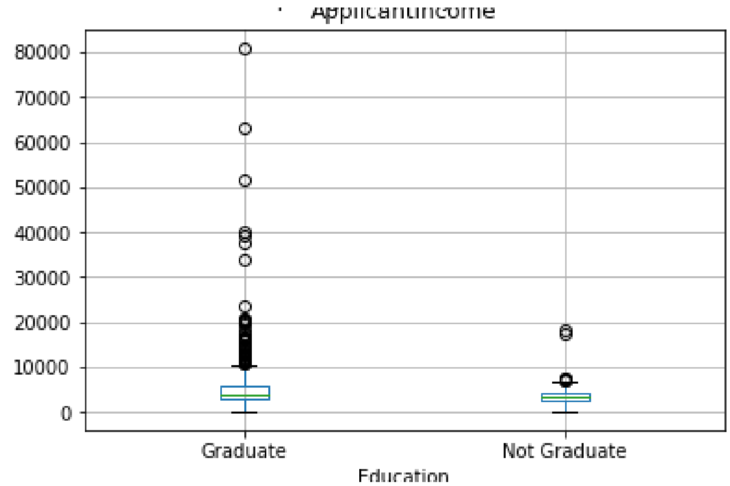
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0xe1a6370>



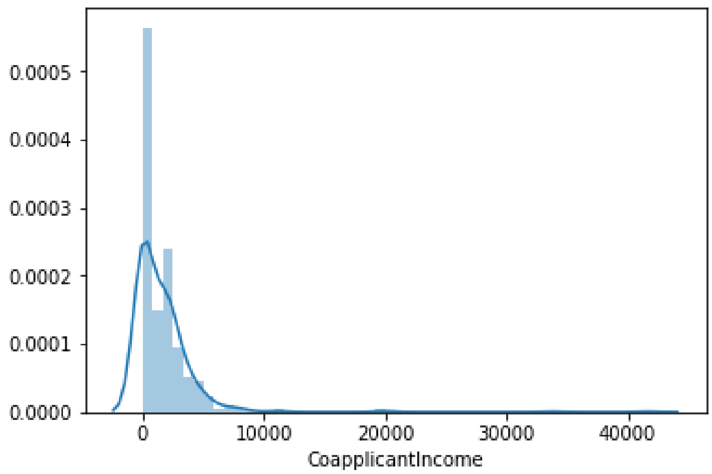
In [23]:
```
train.boxplot(column='ApplicantIncome', by = 'Education')
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0xe1e1fb0>

Boxplot grouped by Education

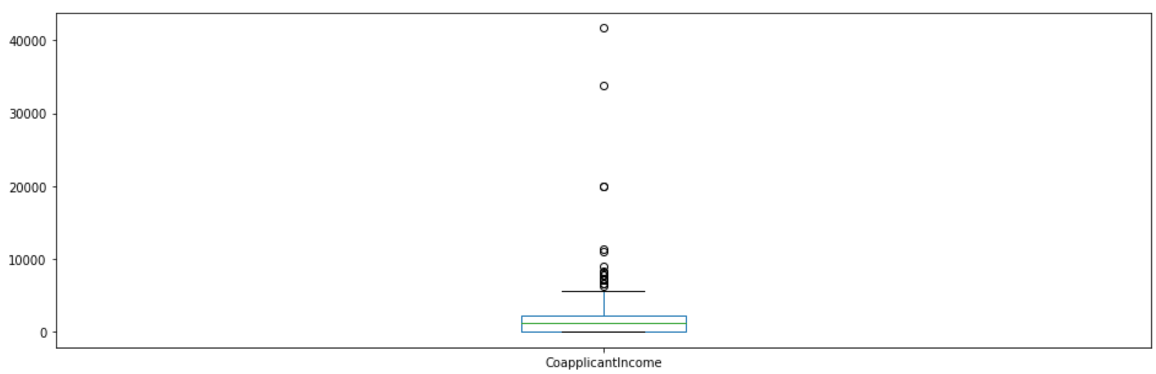


In [24]:
```
sns.distplot(train['CoapplicantIncome']);
```

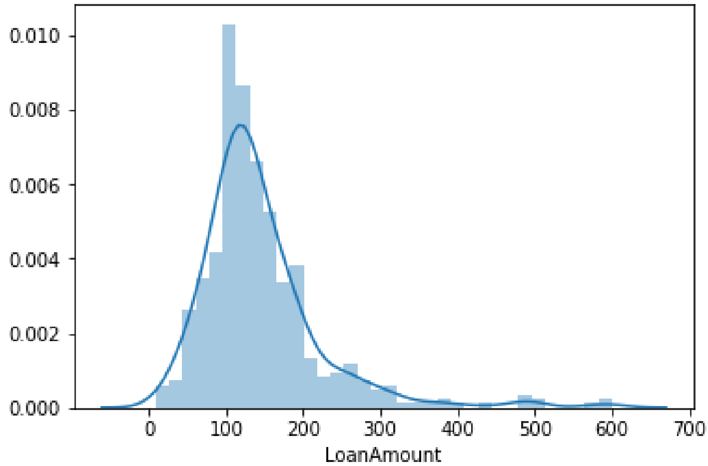


In [25]:
```
train['CoapplicantIncome'].plot.box(figsize=(16,5))
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xe155810>
```

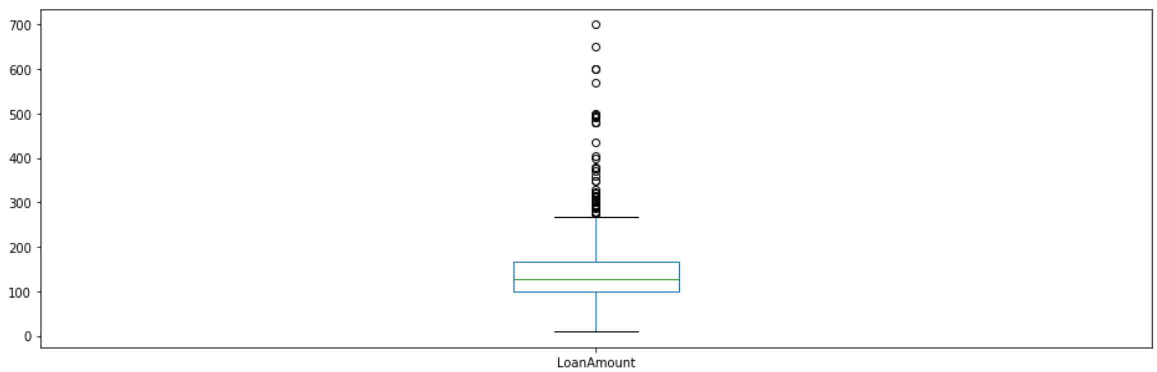


In [26]:
```
df=train.dropna()
sns.distplot(df['LoanAmount']);
```



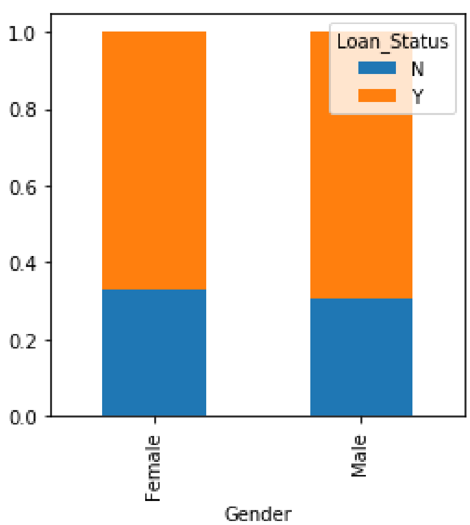
In [27]:
```
train['LoanAmount'].plot.box(figsize=(16,5))
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0xea5ad10>



In [28]:
```
Gender=pd.crosstab(train['Gender'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4))
```

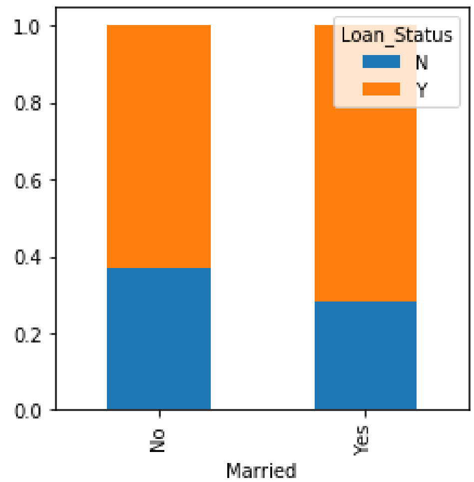
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xe184ab0>



In [29]:
```
Married=pd.crosstab(train['Married'],train['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4))
```

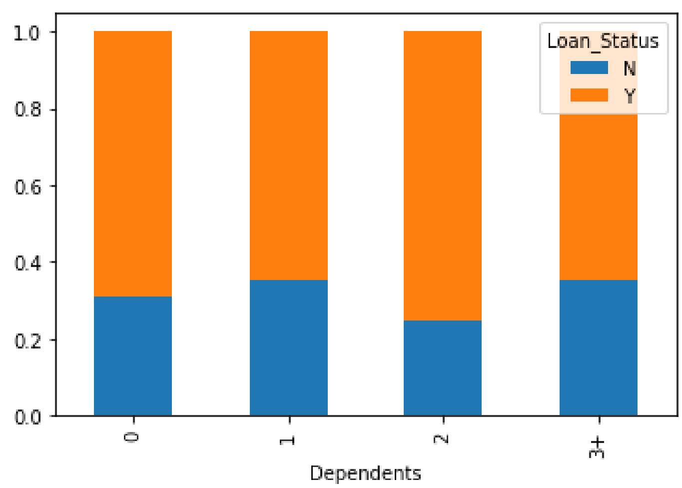
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0xeb21b50>

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0xeb21b50>



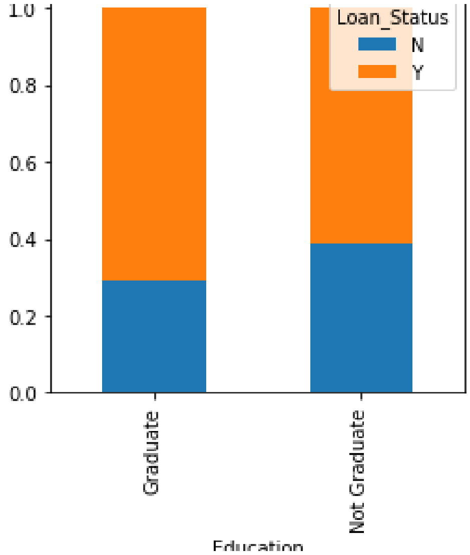
In [30]:
```
Dependents=pd.crosstab(train['Dependents'],train['Loan_Status'])
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", s
tacked=True)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0xeb5f930>



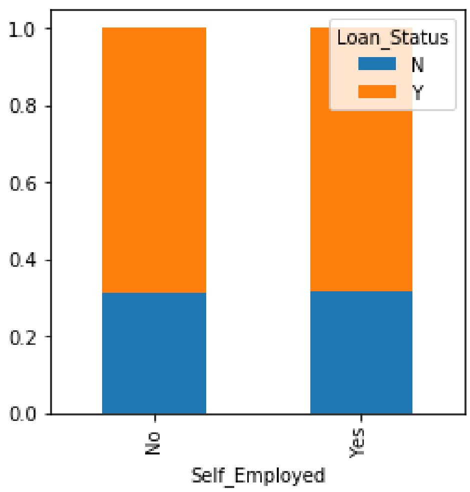
In [31]:
```
Education=pd.crosstab(train['Education'],train['Loan_Status'])
Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", sta
cked=True, figsize=(4,4))
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0xebae2d0>



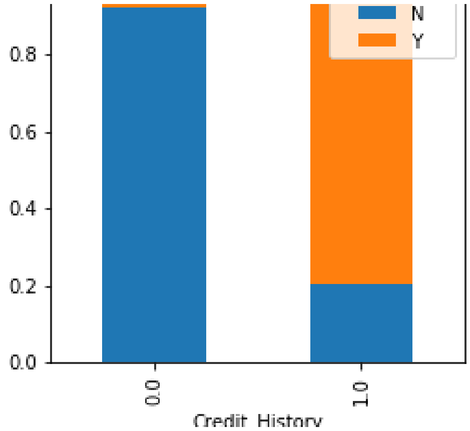
In [32]:
```
Self_Employed=pd.crosstab(train['Self_Employed'],train['Loan_Status'])
Self_Employed.div(Self_Employed.sum(1).astype(float), axis=0).plot(kind="b
ar", stacked=True, figsize=(4,4))
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0xebe9430>



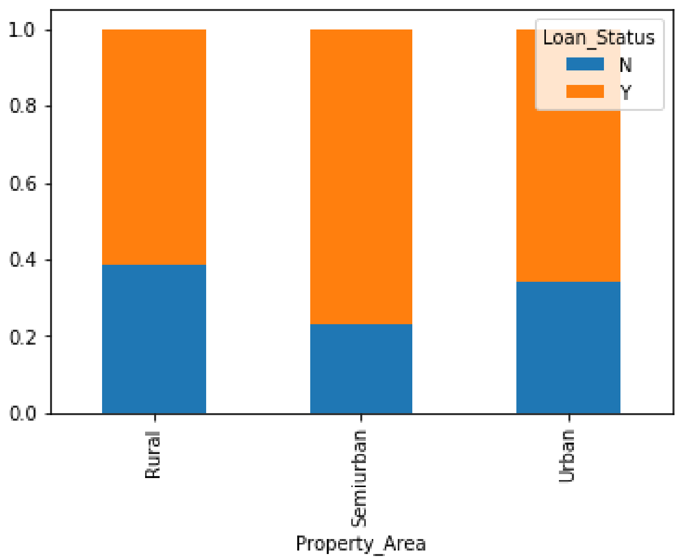
In [33]:
```
Credit_History=pd.crosstab(train['Credit_History'],train['Loan_Status'])
Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind=
"bar", stacked=True, figsize=(4,4))
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0xec43290>



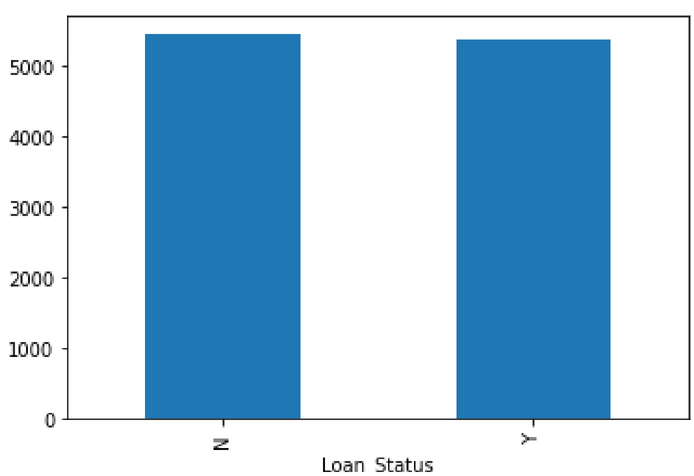
In [34]:
```
Property_Area=pd.crosstab(train['Property_Area'],train['Loan_Status'])
Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind="b
ar", stacked=True)
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0xedbc750>



In [35]:
```
train.groupby('Loan_Status')['ApplicantIncome'].mean().plot.bar()
```

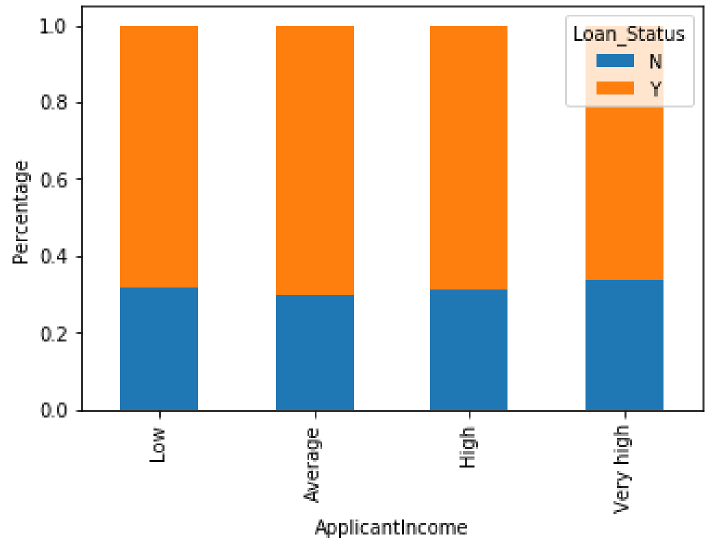
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0xedf7e90>



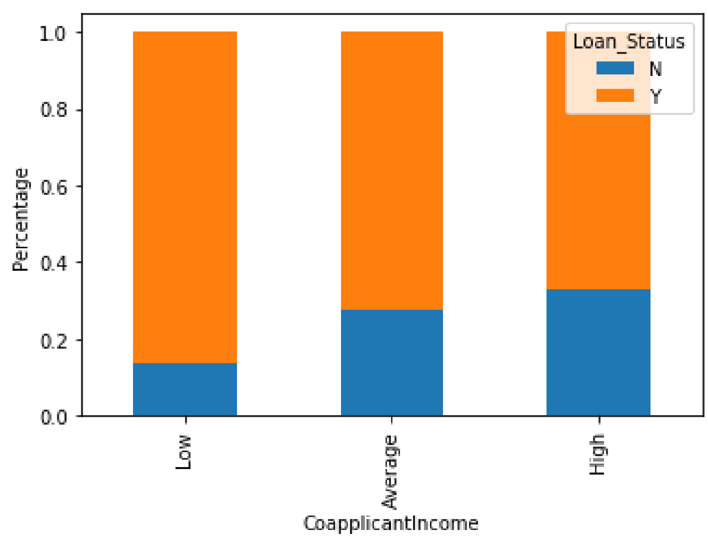
In [36]:
```
bins=[0,2500,4000,6000,81000]
group=['Low','Average','High', 'Very high']
```

```
train['Income_bin']=pd.cut(train['ApplicantIncome'],bins,labels=group)
```

In [37]:
```
Income_bin=pd.crosstab(train['Income_bin'],train['Loan_Status'])
Income_bin.div(Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", s
tacked=True)
plt.xlabel('ApplicantIncome')
P = plt.ylabel('Percentage')
```

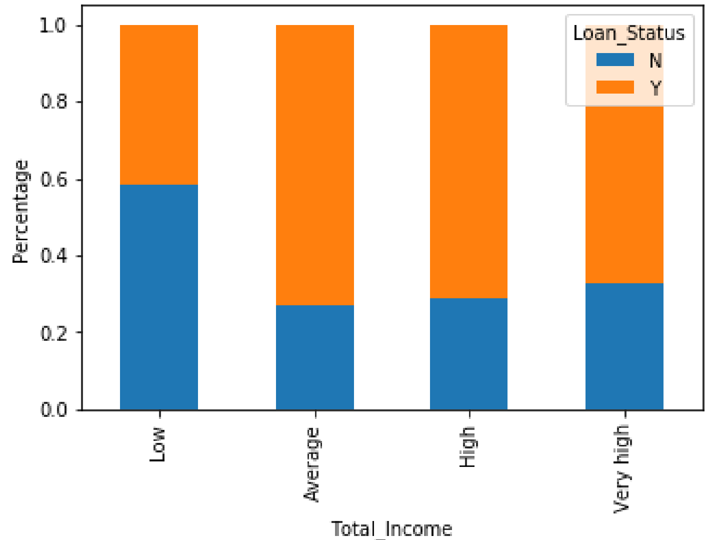


In [38]:
```
bins=[0,1000,3000,42000]
group=['Low','Average','High']
train['Coapplicant_Income_bin']=pd.cut(train['CoapplicantIncome'],bins,lab
els=group)
Coapplicant_Income_bin=pd.crosstab(train['Coapplicant_Income_bin'],train[
'Loan_Status'])
Coapplicant_Income_bin.div(Coapplicant_Income_bin.sum(1).astype(float), ax
is=0).plot(kind="bar", stacked=True)
plt.xlabel('CoapplicantIncome')
P = plt.ylabel('Percentage')
```

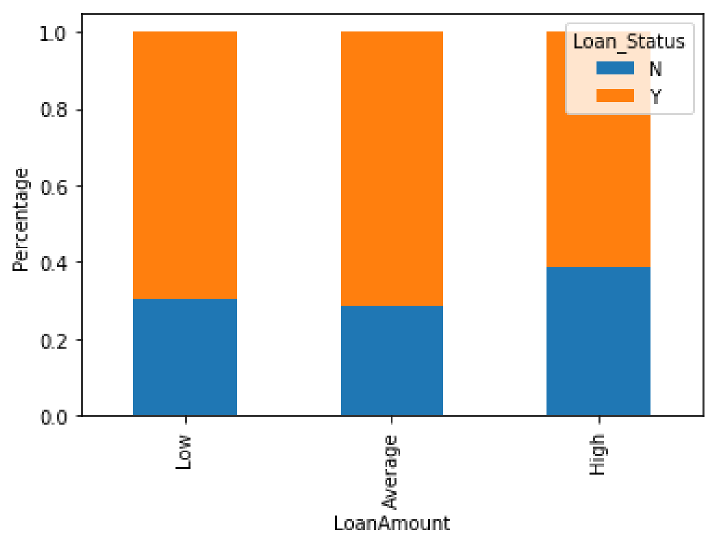


In [39]:
```
train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']
bins=[0,2500,4000,6000,81000]
group=['Low','Average','High', 'Very high']
train['Total_Income_bin']=pd.cut(train['Total_Income'],bins,labels=group)
Total_Income_bin=pd.crosstab(train['Total_Income_bin'],train['Loan_Status'
```

```
])
Total_Income_bin.div(Total_Income_bin.sum(1).astype(float), axis=0).plot(k
ind="bar", stacked=True)
plt.xlabel('Total_Income')
P = plt.ylabel('Percentage')
```



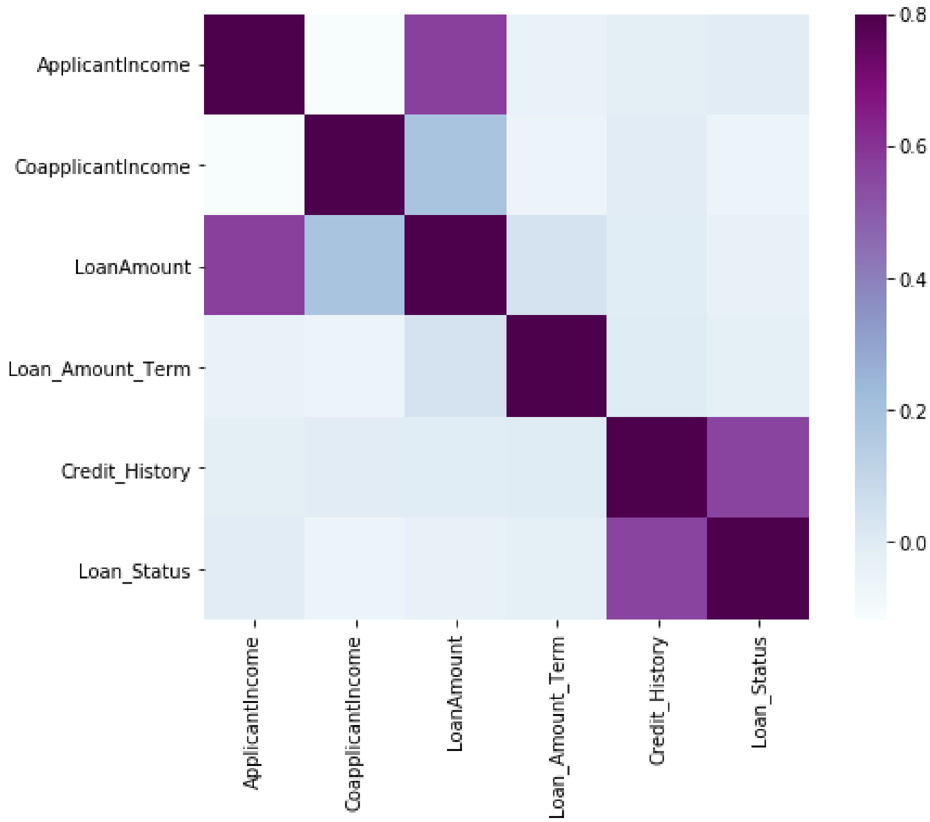
In [40]:
```
bins=[0,100,200,700]
group=['Low','Average','High']
train['LoanAmount_bin']=pd.cut(train['LoanAmount'],bins,labels=group)
LoanAmount_bin=pd.crosstab(train['LoanAmount_bin'],train['Loan_Status'])
LoanAmount_bin.div(LoanAmount_bin.sum(1).astype(float), axis=0).plot(kind=
"bar", stacked=True)
plt.xlabel('LoanAmount')
P = plt.ylabel('Percentage')
```



In [41]:
```
train=train.drop(['Income_bin', 'Coapplicant_Income_bin', 'LoanAmount_bin'
, 'Total_Income_bin', 'Total_Income'], axis=1)
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 [42]:
```
matrix = train.corr()
f, ax = plt.subplots(figsize=(9, 6))
```

```
sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu");
```



In [43]:
```
train.isnull().sum()
```

Out[43]:
```
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 [44]:
```
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=Tr
ue)
train['Credit_History'].fillna(train['Credit_History'].mode()[0], inplace=
True)
```

In [45]:
```
train['Loan_Amount_Term'].value_counts()
```

Out[45]:
```
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 [46]:
```python
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
```

In [47]:
```python
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

In [48]:
```python
train.isnull().sum()
```

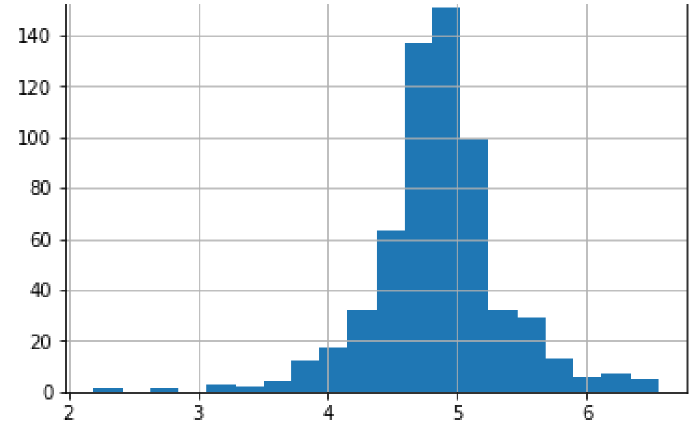
Out[48]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [49]:
```python
test['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
test['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
test['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
test['Credit_History'].fillna(train['Credit_History'].mode()[0], inplace=True)
test['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
test['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

In [50]:
```python
test.isnull().sum()
```

Out[50]:
```
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 [51]:
```python
train['LoanAmount_log'] = np.log(train['LoanAmount'])
train['LoanAmount_log'].hist(bins=20)
test['LoanAmount_log'] = np.log(test['LoanAmount'])
```



In [52]:
```
train=train.drop('Loan_ID',axis=1)
test=test.drop('Loan_ID',axis=1)
```

In [53]:
```
X = train.drop('Loan_Status',1)
y = train.Loan_Status
```

In [54]:
```
X=pd.get_dummies(X)
train=pd.get_dummies(train)
test=pd.get_dummies(test)
```

In [55]:
```
from sklearn.model_selection import train_test_split
x_train, x_cv, y_train, y_cv = train_test_split(X,y, test_size =0.3)
```

In [56]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression()
model.fit(x_train, y_train)
```

Out[56]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [57]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n
_jobs=1,
                   penalty='l2', random_state=1, solver='liblinear', tol=
0.0001,
                   verbose=0, warm_start=False)
```

Out[57]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=1, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [58]:
```
pred_cv = model.predict(x_cv)   #Let’s predict the Loan_Status for validat
ion set and calculate its accuracy.
```

In [59]:
```
accuracy_score(y_cv,pred_cv)    #Let us calculate how accurate our predict
ions are by calculating the accuracy.
```

Out[59]: 0.7891891891891892

Out[59]: 0.7891891891891892

In [60]:
```
pred_test = model.predict(test) #Let’s make predictions for the test datas
et.
```

In [61]:
```
submission=pd.read_csv("C:/Users/Admin/Desktop/ML/Sample_Submission_ZAuTl8
O_FK3zQHh.csv")
```

In [62]:
```
submission['Loan_Status']=pred_test
submission['Loan_ID']=test_original['Loan_ID']
```

In [63]:
```
submission['Loan_Status'].replace(0, 'N',inplace=True)
submission['Loan_Status'].replace(1, 'Y',inplace=True)
```

In [64]:
```
pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('logist
ic.csv')
logistic= pd.read_csv("logistic.csv")
logistic.columns
```

Out[64]:
```
Index(['Unnamed: 0', 'Loan_ID', 'Loan_Status'], dtype='object')
```

In [65]:
```
print(logistic)
```

```
     Unnamed: 0   Loan_ID Loan_Status
0             0  LP001015           N
1             1  LP001015           Y
2             2  LP001022           Y
3             3  LP001031           Y
4             4  LP001035           Y
5             5  LP001051           Y
6             6  LP001054           Y
7             7  LP001055           Y
8             8  LP001056           N
9             9  LP001059           Y
10           10  LP001067           Y
11           11  LP001078           Y
12           12  LP001082           Y
13           13  LP001083           Y
14           14  LP001094           N
15           15  LP001096           Y
16           16  LP001099           Y
17           17  LP001105           Y
18           18  LP001107           Y
19           19  LP001108           Y
20           20  LP001115           Y
21           21  LP001121           Y
22           22  LP001124           Y
23           23  LP001128           Y
24           24  LP001135           Y
25           25  LP001149           Y
26           26  LP001153           N
27           27  LP001163           Y
28           28  LP001169           Y
29           29  LP001174           Y
..          ...       ...         ...
338         338  LP002856           Y
339         339  LP002857           Y
340         340  LP002858           N
341         341  LP002860           Y
342         342  LP002867           Y
```

```
343          343  LP002869         Y
344          344  LP002870         Y
345          345  LP002876         Y
346          346  LP002878         Y
347          347  LP002879         N
348          348  LP002885         Y
349          349  LP002890         Y
350          350  LP002891         Y
351          351  LP002899         Y
352          352  LP002901         Y
353          353  LP002907         Y
354          354  LP002920         Y
355          355  LP002921         N
356          356  LP002932         Y
357          357  LP002935         Y
358          358  LP002952         Y
359          359  LP002954         Y
360          360  LP002962         Y
361          361  LP002965         Y
362          362  LP002969         Y
363          363  LP002971         Y
364          364  LP002975         Y
365          365  LP002980         Y
366          366  LP002986         Y
367          367  LP002989         Y

[368 rows x 3 columns]
```

In [66]:
```
from sklearn.model_selection import StratifiedKFold
```

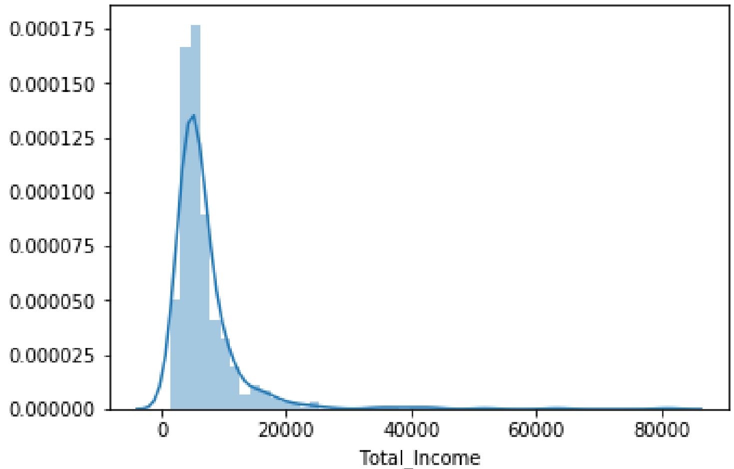
In [67]:
```
i=1
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = LogisticRegression(random_state=1)
    model.fit(xtr, ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
    print('accuracy_score',score)
    i+=1
    pred_test = model.predict(test)
    pred=model.predict_proba(xvl)[:,1]
```

```
1 of kfold 5
accuracy_score 0.7983870967741935

2 of kfold 5
accuracy_score 0.8306451612903226

3 of kfold 5
accuracy_score 0.8114754098360656

4 of kfold 5
accuracy_score 0.7950819672131147

5 of kfold 5
accuracy_score 0.8278688524590164
```

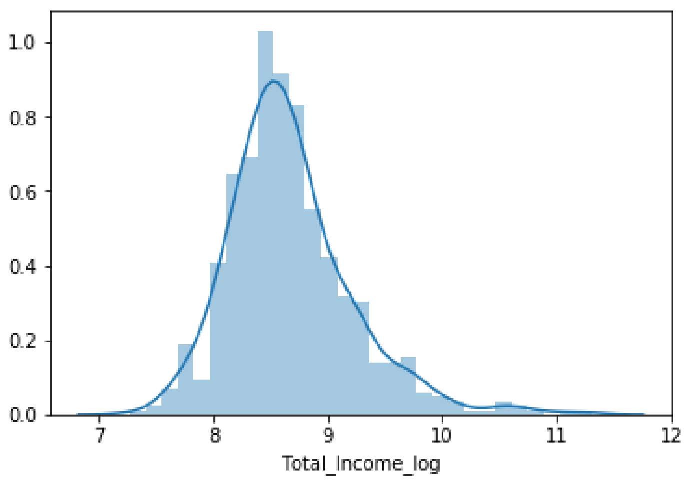
In [68]:
```
train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']
```

```
test['Total_Income']=test['ApplicantIncome']+test['CoapplicantIncome']
```

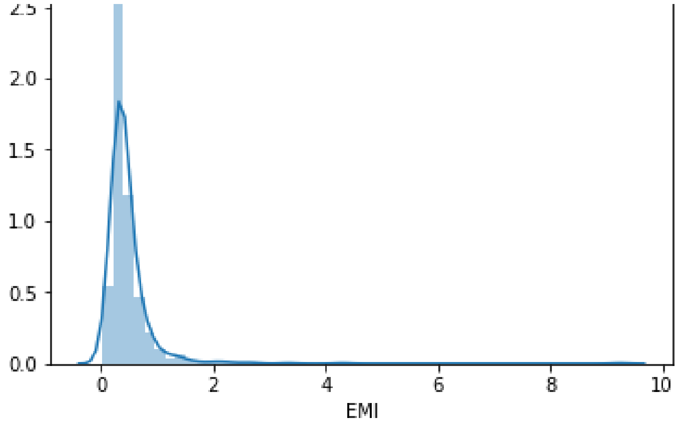
In [69]:
```
sns.distplot(train['Total_Income']);
```



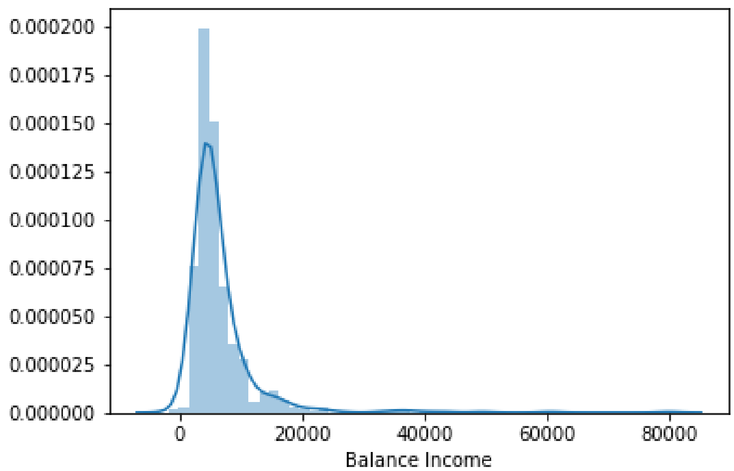
In [70]:
```
train['Total_Income_log'] = np.log(train['Total_Income'])
test['Total_Income_log'] = np.log(test['Total_Income'])
sns.distplot(train['Total_Income_log']);
```



In [71]:
```
train['EMI']=train['LoanAmount']/train['Loan_Amount_Term']
test['EMI']=test['LoanAmount']/test['Loan_Amount_Term']
sns.distplot(train['EMI']);
```



In [72]:
```
train['Balance Income']=train['Total_Income']-(train['EMI']*1000) # Multip
ly with 1000 to make the units equal
test['Balance Income']=test['Total_Income']-(test['EMI']*1000)
sns.distplot(train['Balance Income']);
```



In [73]:
```
train=train.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'L
oan_Amount_Term'], axis=1)
test=test.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loa
n_Amount_Term'], axis=1)
```

In [74]:
```
X = train.drop('Loan_Status',1)
y = train.Loan_Status                # Save target variable in separate da
taset
```

In [75]:
```
i=1
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]

    model = LogisticRegression(random_state=1)
    model.fit(xtr, ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
    print('accuracy_score',score)
    i+=1
    pred_test = model.predict(test)
    pred=model.predict_proba(xvl)[:,1]
```

```
1 of kfold 5
accuracy_score 0.8064516129032258

2 of kfold 5
accuracy_score 0.8306451612903226

3 of kfold 5
accuracy_score 0.7786885245901639

4 of kfold 5
accuracy_score 0.7868852459016393

5 of kfold 5
accuracy_score 0.819672131147541
```

In [76]:

```
submission['Loan_Status']=pred_test # filling Loan_Status with predictions
submission['Loan_ID']=test_original['Loan_ID'] # filling Loan_ID with test
Loan_ID
# replacing 0 and 1 with N and Y
submission['Loan_Status'].replace(0, 'N',inplace=True)
submission['Loan_Status'].replace(1, 'Y',inplace=True)
# Converting submission file to .csv format
pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('Log2.c
sv')
```

In [77]:

```
from sklearn import tree
```

In [78]:

```
i=1
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = tree.DecisionTreeClassifier(random_state=1)
    model.fit(xtr, ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
    print('accuracy_score',score)
    i+=1
    pred_test = model.predict(test)
```

```
1 of kfold 5
accuracy_score 0.7258064516129032

2 of kfold 5
accuracy_score 0.7419354838709677

3 of kfold 5
accuracy_score 0.7049180327868853

4 of kfold 5
accuracy_score 0.680327868852459

5 of kfold 5
accuracy_score 0.7049180327868853
```

In [79]:

```
submission['Loan_Status']=pred_test            # filling Loan_Status with
 predictions
submission['Loan_ID']=test_original['Loan_ID'] # filling Loan_ID with test
Loan_ID
```

```python
# replacing 0 and 1 with N and Y
submission['Loan_Status'].replace(0, 'N',inplace=True)
submission['Loan_Status'].replace(1, 'Y',inplace=True)
# Converting submission file to .csv format
pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('Decisi
on Tree.csv')
```

In [80]:

```python
from sklearn.ensemble import RandomForestClassifier
```

In [81]:

```python
i=1
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = RandomForestClassifier(random_state=1, max_depth=10)
    model.fit(xtr, ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
    print('accuracy_score',score)
    i+=1

    pred_test = model.predict(test)
```

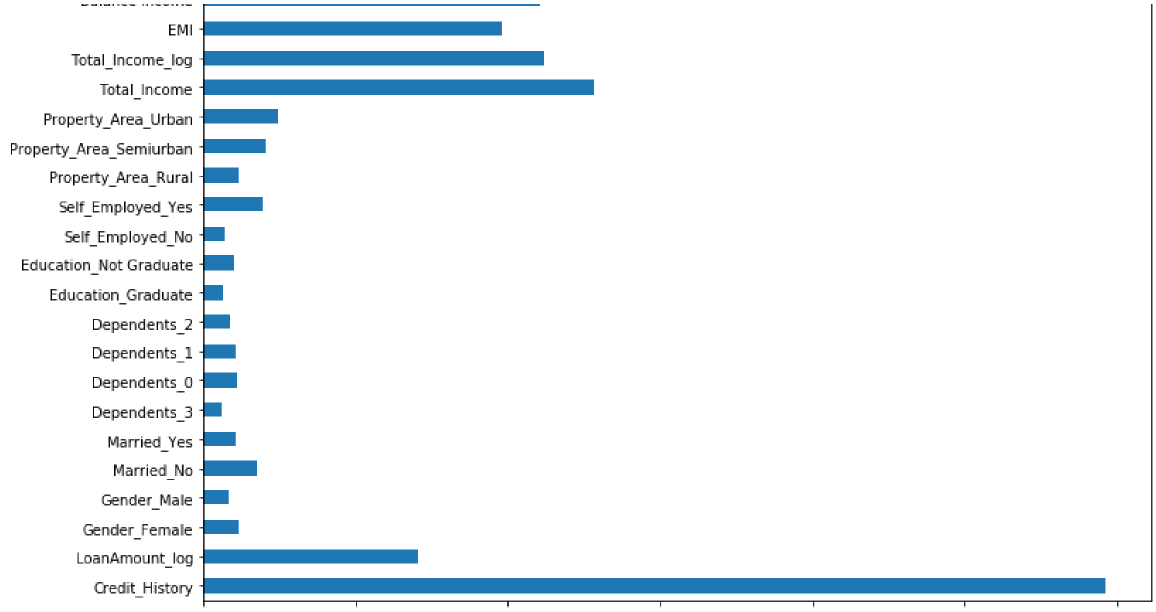
```
1 of kfold 5
accuracy_score 0.8225806451612904

2 of kfold 5
accuracy_score 0.8145161290322581

3 of kfold 5
accuracy_score 0.7377049180327869

4 of kfold 5
accuracy_score 0.7295081967213115

5 of kfold 5
accuracy_score 0.8114754098360656
```

In [82]:

```python
importances=pd.Series(model.feature_importances_, index=X.columns)
importances.plot(kind='barh', figsize=(12,8))
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x536f6b0>
```



EMI
Total_Income_log
Total_Income
Property_Area_Urban
Property_Area_Semiurban
Property_Area_Rural
Self_Employed_Yes
Self_Employed_No
Education_Not Graduate
Education_Graduate
Dependents_2
Dependents_1
Dependents_0
Dependents_3
Married_Yes
Married_No
Gender_Male
Gender_Female
LoanAmount_log
Credit_History

In [ ]: