```
from io import StringIO
from google.colab import files
uploaded=files.upload()
```

Choose Files train_ctrUa4K.csv

- **train_ctrUa4K.csv**(text/csv) - 38011 bytes, last modified: 5/24/2023 - 100% done

```
Saving train_ctrUa4K.csv to train_ctrUa4K.csv
```

```
from io import StringIO
from google.colab import files
uploaded=files.upload()
```

Choose Files test_lAUu6dG.csv

- **test_lAUu6dG.csv**(text/csv) - 21955 bytes, last modified: 5/24/2023 - 100% done

```
Saving test_lAUu6dG.csv to test_lAUu6dG.csv
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
train=pd.read_csv("train_ctrUa4K.csv")
test=pd.read_csv("test_lAUu6dG.csv")
```

```
train_original=train.copy()
test_original=test.copy()
```

```
train.columns
```

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

```
test.columns
```

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

```
train.dtypes
```

```
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
```

```
train.shape, test.shape
```

```
((614, 13), (367, 12))
```

```
train['Loan_Status'].value_counts()
```

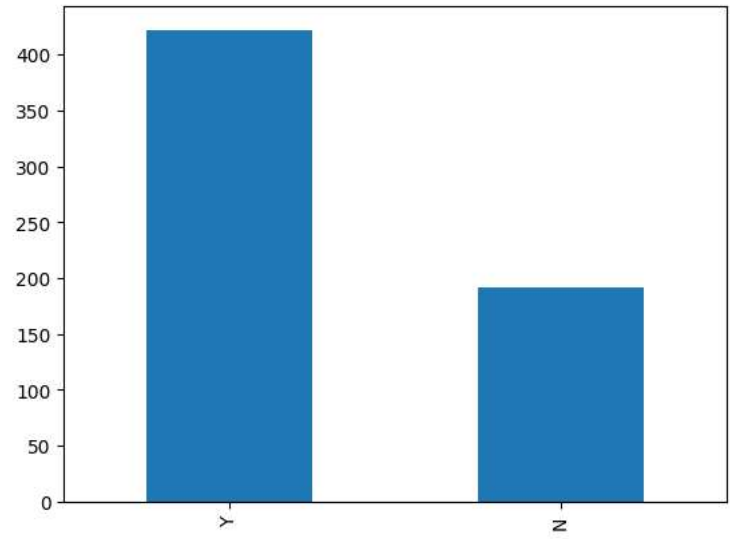
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

```
train['Loan_Status'].value_counts(normalize=True)
```

```
Y    0.687296
N    0.312704
Name: Loan_Status, dtype: float64
```

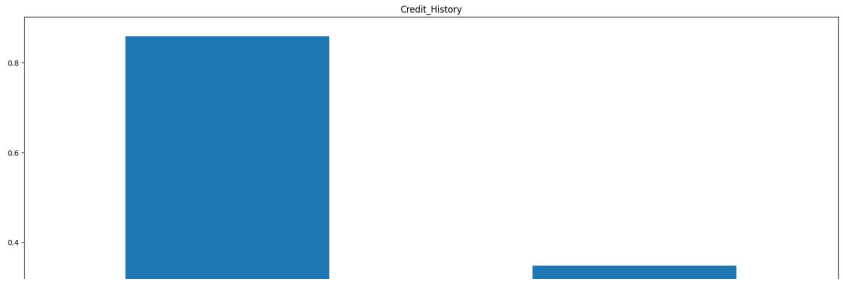
```
train['Loan_Status'].value_counts().plot.bar()
```

```
<Axes: >
```



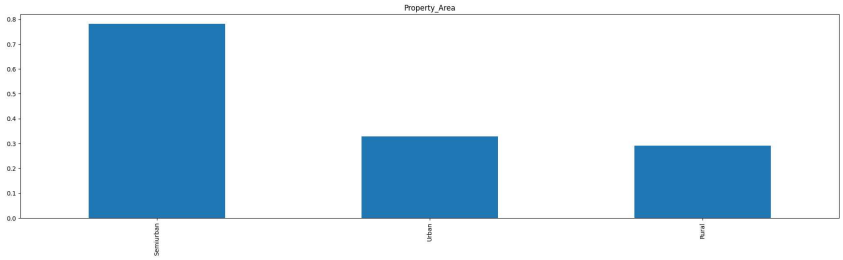

```
train['Gender'].value_counts(normalize=True).plot.bar(figsize=(20,10), title= 'Gender')
train['Married'].value_counts(normalize=True).plot.bar(title= 'Married')
train['Self_Employed'].value_counts(normalize=True).plot.bar(title= 'Self_Employed')
train['Credit_History'].value_counts(normalize=True).plot.bar(title= 'Credit_History')
```

```
<Axes: title={'center': 'Credit_History'}>
```




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

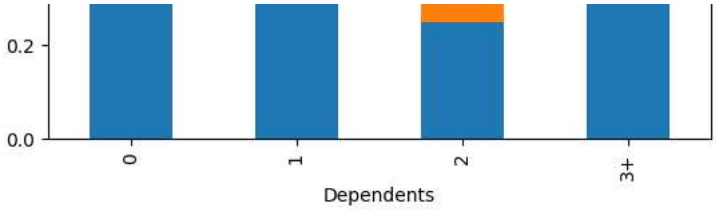
```
<Axes: title={'center': 'Property_Area'}>
```

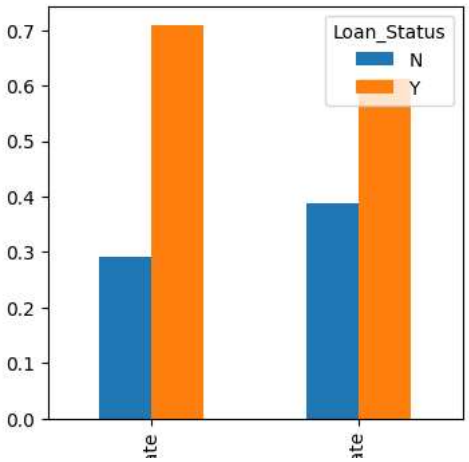



```
sns.distplot(train['ApplicantIncome']);
train['ApplicantIncome'].plot.box(figsize=(16,5))
```

```
<Axes: xlabel='ApplicantIncome', ylabel='Density'>
```

```
Married=pd.crosstab(train['Married'],train['Loan_Status'])
Dependents=pd.crosstab(train['Dependents'],train['Loan_Status'])
Education=pd.crosstab(train['Education'],train['Loan_Status'])
Self_Employed=pd.crosstab(train['Self_Employed'],train['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
plt.show()
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
plt.show()
Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
plt.show()
Self_Employed.div(Self_Employed.sum(1).astype(float),axis=0).plot(kind="bar",figsize=(4,4))
plt.show()
```

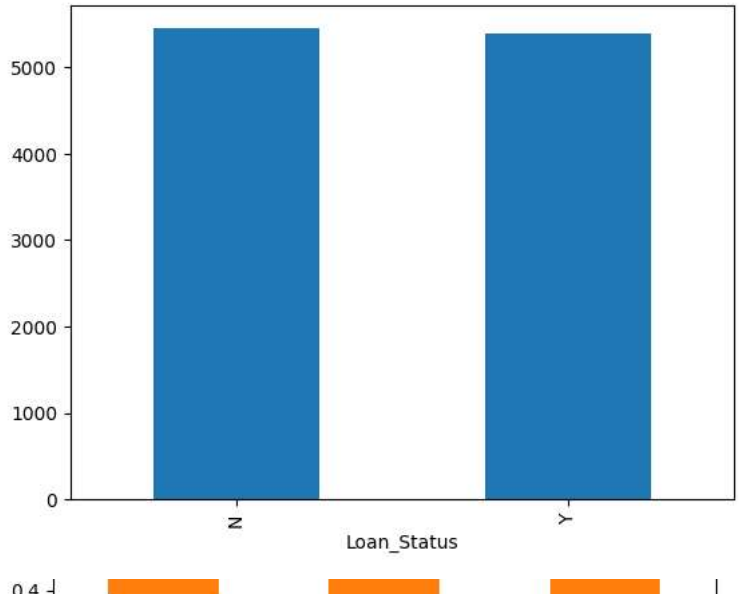




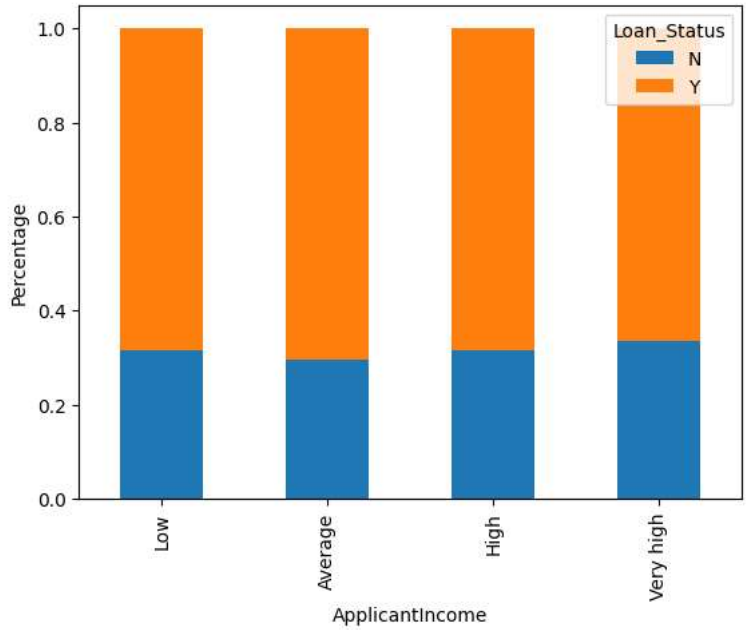

```
Credit_History=pd.crosstab(train['Credit_History'],train['Loan_Status'])
Property_Area=pd.crosstab(train['Property_Area'],train['Loan_Status'])
Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4))
plt.show()
Property_Area.div(Property_Area.sum(1).astype(float),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


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

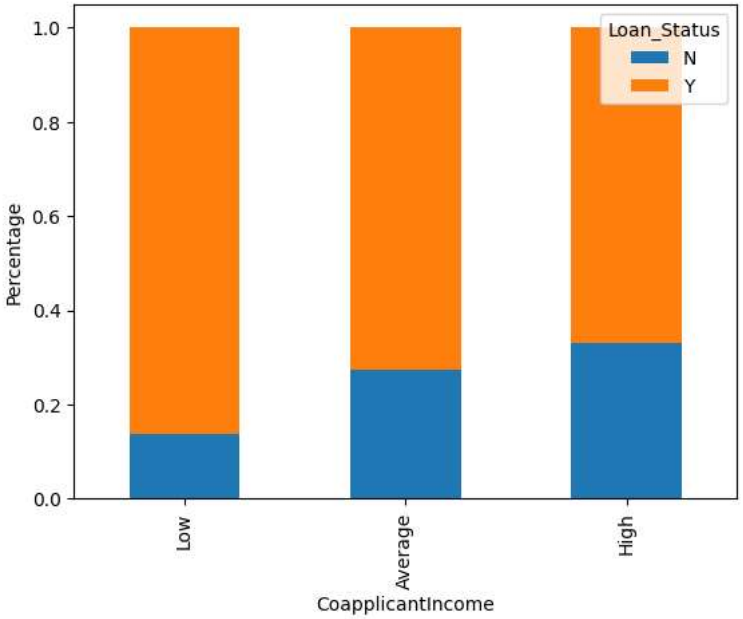
```
<Axes: xlabel='Loan_Status'>
```



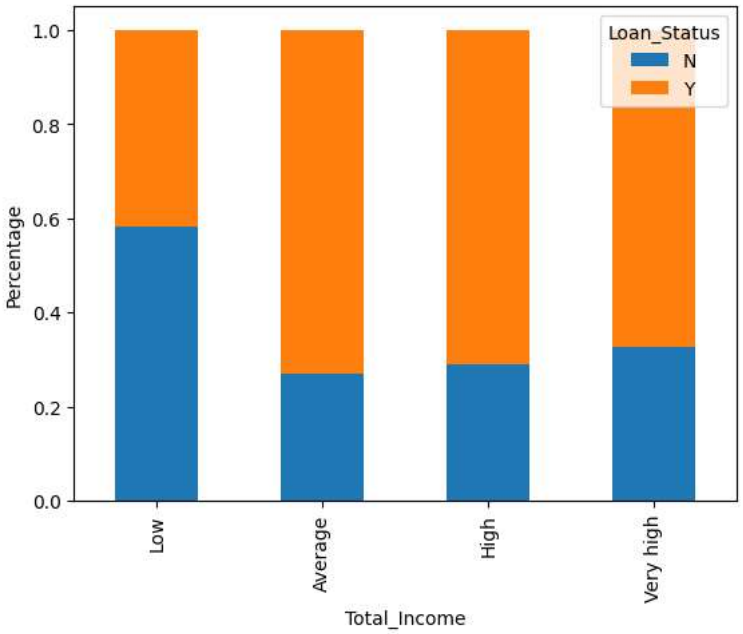

```
bins=[0,2500,4000,6000,81000]
group=['Low','Average','High', 'Very high']
train['Income_bin']=pd.cut(train['ApplicantIncome'],bins,labels=group)
Income_bin=pd.crosstab(train['Income_bin'],train['Loan_Status'])
Income_bin.div(Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
plt.xlabel('ApplicantIncome')
P = plt.ylabel('Percentage')
```



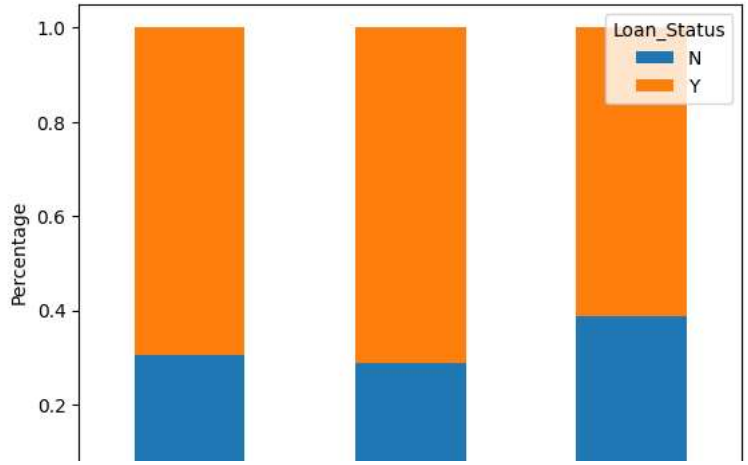

```
bins=[0,1000,3000,42000]
group=['Low','Average','High']
train['Coapplicant_Income_bin']=pd.cut(train['CoapplicantIncome'],bins,labels=group)
Coapplicant_Income_bin=pd.crosstab(train['Coapplicant_Income_bin'],train['Loan_Status'])
Coapplicant_Income_bin.div(Coapplicant_Income_bin.sum(1).astype(float),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




```
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(kind="bar", stacked=True)
plt.xlabel('Total_Income')
P = plt.ylabel('Percentage')
```

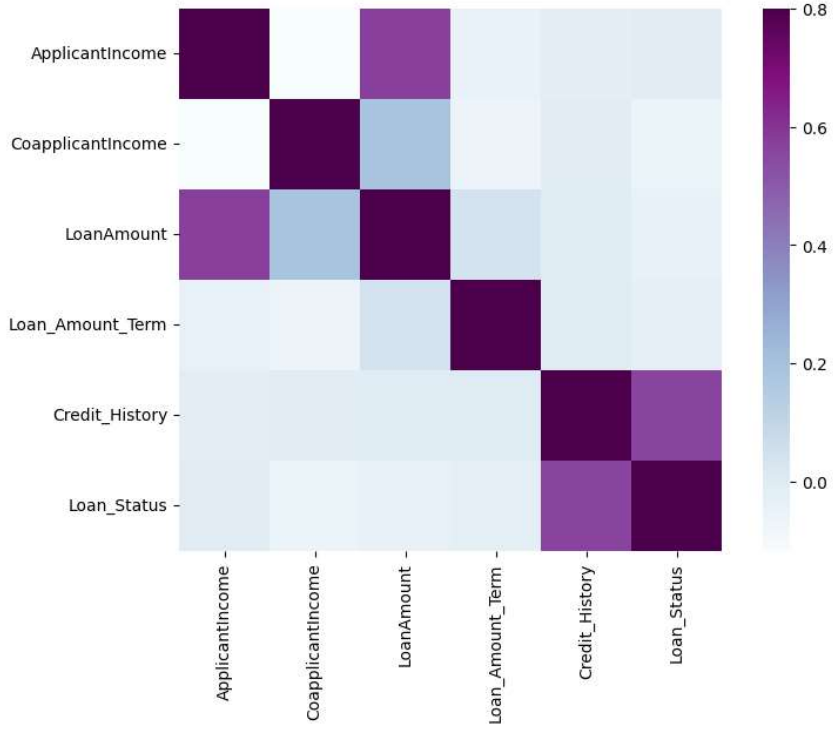



```
bins=[0,100,200,700]
group=['Low','Average','High']
train['LoanAmount_bin']=pd.cut(train['LoanAmount'],bins,labels=group)
LoanAmount_bin=pd.crosstab(train['LoanAmount_bin'],train['Loan_Status'])
LoanAmount_bin.div(LoanAmount_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
plt.xlabel('LoanAmount')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




```
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)
```

```
matrix = train.corr()
f, ax = plt.subplots(figsize=(9, 6))
sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu");
```




```
train.isnull().sum()
```

```
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
```

```
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)
```

```
train['Loan_Amount_Term'].value_counts()
```

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

```
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
```

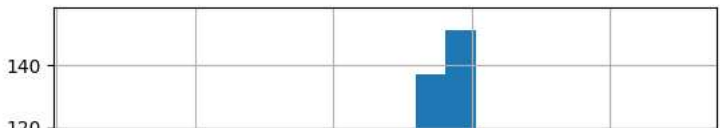
```
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

```
train.isnull().sum()
```

```
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
```

```
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)
```

```
train['LoanAmount_log'] = np.log(train['LoanAmount'])
train['LoanAmount_log'].hist(bins=20)
test['LoanAmount_log'] = np.log(test['LoanAmount'])
```




```
train=train.drop('Loan_ID',axis=1)
test=test.drop('Loan_ID',axis=1)
```

```
X = train.drop('Loan_Status',1)
y = train.Loan_Status
```


```
X=pd.get_dummies(X)
train=pd.get_dummies(train)
test=pd.get_dummies(test)
```

```
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)
```



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
pred_cv = model.predict(x_cv)
```

```
accuracy_score(y_cv,pred_cv)
```

```
0.8324324324324325
```

```
pred_test = model.predict(test)
```

```
from io import StringIO
from google.colab import files
uploaded=files.upload()
```

Choose Files sample_su…49d68Cx.csv

- **sample_submission_49d68Cx.csv**(text/csv) - 4423 bytes, last modified: 5/24/2023 - 100% done

```
Saving sample_submission_49d68Cx.csv to sample_submission_49d68Cx.csv
```

```
submission=pd.read_csv("sample_submission_49d68Cx.csv")
```

```
submission['Loan_Status']=pred_test
submission['Loan_ID']=test_original['Loan_ID']
```

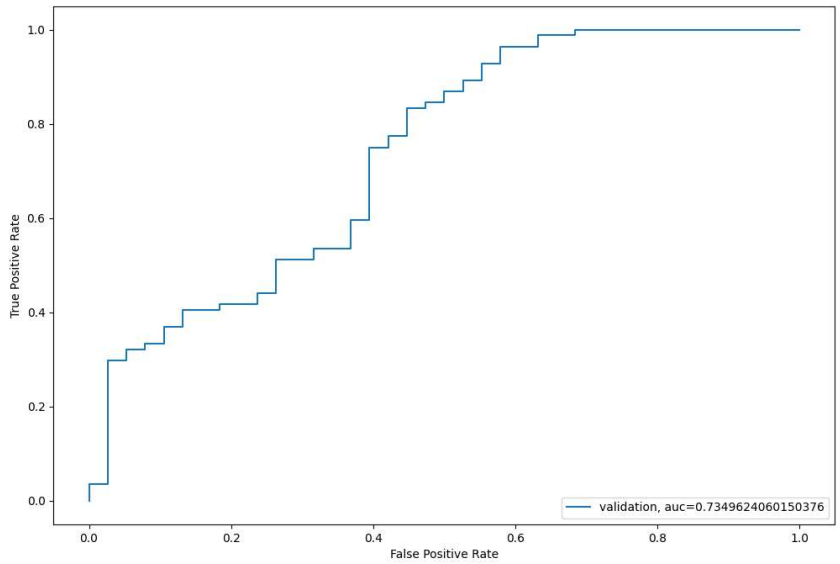
```
submission['Loan_Status'].replace(0, 'N',inplace=True)
submission['Loan_Status'].replace(1, 'Y',inplace=True)
```

```
pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('logistic.csv')
```

```
from sklearn.model_selection import StratifiedKFold
```

```
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.iloc[train_index],X.iloc[test_index]
    ytr,yvl = y.iloc[train_index],y.iloc[test_index]
    model = LogisticRegression(random_state=1)
    model.fit(xtr, ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
```

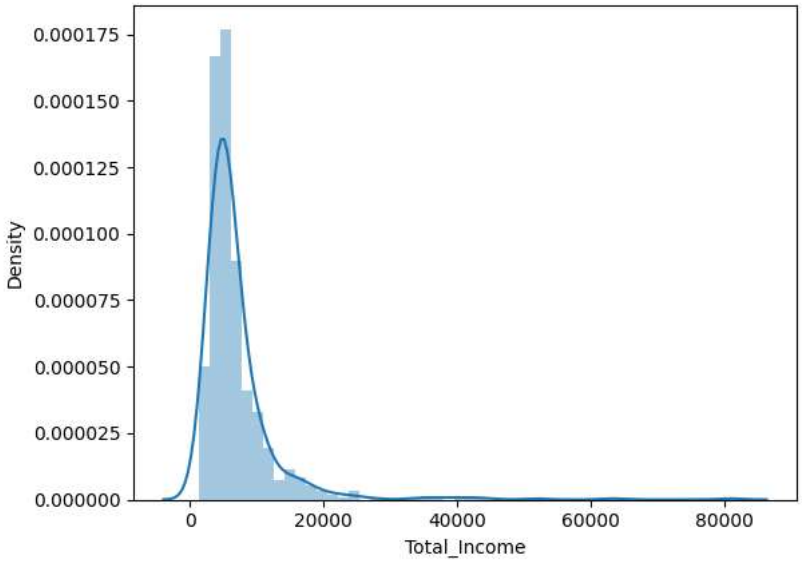
```
        print('accuracy_score',score)
        i+=1

        n1 of kfold 5
        accuracy_score 0.8048780487804879
        n2 of kfold 5
        accuracy_score 0.8373983739837398
        n3 of kfold 5
        accuracy_score 0.7804878048780488
        n4 of kfold 5
        accuracy_score 0.7886178861788617
        n5 of kfold 5
        accuracy_score 0.7950819672131147


pred_test = model.predict(test)
pred=model.predict_proba(xvl)[:,1]


from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(yvl,  pred)
auc = metrics.roc_auc_score(yvl, pred)


plt.figure(figsize=(12,8))
plt.plot(fpr,tpr,label="validation, auc="+str(auc))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.show()
```



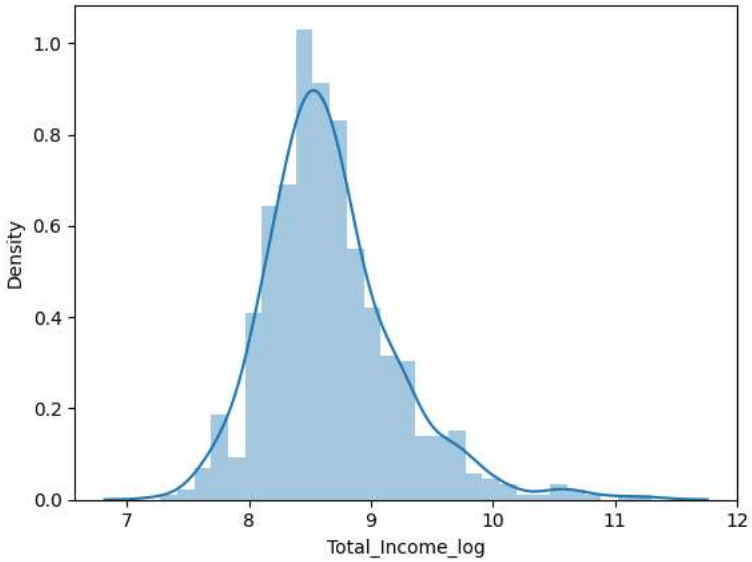

```
submission['Loan_Status']=pred_test
submission['Loan_ID']=test_original['Loan_ID']


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


pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('Logistic.csv')
```

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

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



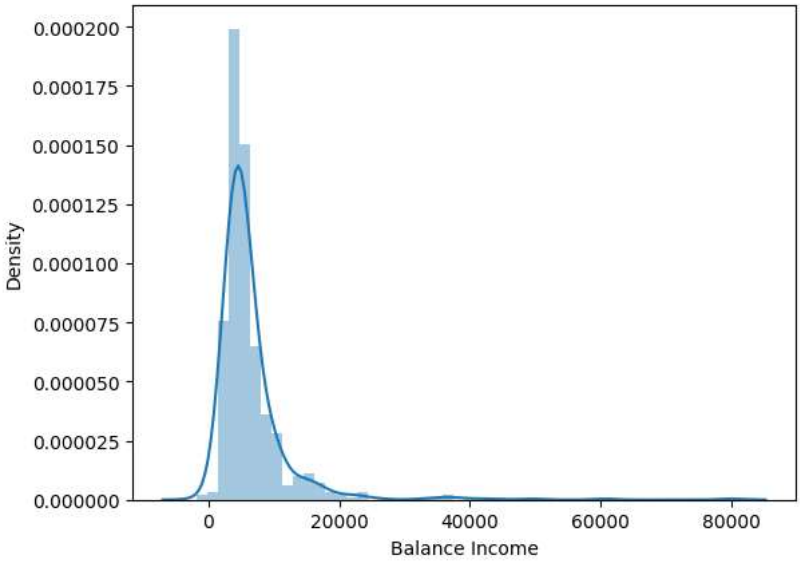

```
train['Total_Income_log'] = np.log(train['Total_Income'])
sns.distplot(train['Total_Income_log']);
test['Total_Income_log'] = np.log(test['Total_Income'])
```




```
train['EMI']=train['LoanAmount']/train['Loan_Amount_Term']
test['EMI']=test['LoanAmount']/test['Loan_Amount_Term']
```

```
sns.distplot(train['EMI']);
```


```
train['Balance Income']=train['Total_Income']-(train['EMI']*1000)
test['Balance Income']=test['Total_Income']-(test['EMI']*1000)
sns.distplot(train['Balance Income']);
```




```
train=train.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
 'Loan_Amount_Term'], axis=1)
test=test.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
 'Loan_Amount_Term'], axis=1)


X = train.drop('Loan_Status',1)
y = train.Loan_Status


from sklearn import tree


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)
```

```
n1 of kfold 5
accuracy_score 0.7398373983739838
n2 of kfold 5
accuracy_score 0.6991869918699187
n3 of kfold 5
accuracy_score 0.7560975609756098
n4 of kfold 5
accuracy_score 0.7073170731707317
n5 of kfold 5
accuracy_score 0.6721311475409836
```

```
submission['Loan_Status']=pred_test
submission['Loan_ID']=test_original['Loan_ID']
submission['Loan_Status'].replace(0, 'N',inplace=True)
submission['Loan_Status'].replace(1, 'Y',inplace=True)
pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('Decision Tree.csv')


from sklearn.ensemble import RandomForestClassifier
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)
```

```
n1 of kfold 5
accuracy_score 0.8292682926829268
n2 of kfold 5
accuracy_score 0.8130081300813008
n3 of kfold 5
accuracy_score 0.7723577235772358
n4 of kfold 5
accuracy_score 0.8048780487804879
n5 of kfold 5
accuracy_score 0.7540983606557377
```

```
from sklearn.model_selection import GridSearchCV


paramgrid = {'max_depth': list(range(1, 20, 2)),
                    'n_estimators': list(range(1, 200, 20))}
grid_search=GridSearchCV(RandomForestClassifier(random_state=1),paramgrid)
from sklearn.model_selection import train_test_split
# Fit the grid search model
grid_search.fit(x_train,y_train)
GridSearchCV(cv=None, error_score='raise',
        estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                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=10, n_jobs=1, oob_score=False,
random_state=1, verbose=0, warm_start=False),
fit_params=None, iid=True, n_jobs=1,
param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
     'n_estimators': [1, 21, 41, 61, 81, 101, 121, 141, 161, 181]},

pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',scoring=None, verbose=0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-104-1bc0f078a2e5> in <cell line: 7>()
      6 grid_search.fit(x_train,y_train)
      7 GridSearchCV(cv=None, error_score='raise',
----> 8         estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
      9                 criterion='gini', max_depth=None, max_features='auto',
     10                 max_leaf_nodes=None, min_impurity_decrease=0.0,

TypeError: RandomForestClassifier.__init__() got an unexpected keyword argument
'min_impurity_split'

SEARCH STACK OVERFLOW
```