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***

warnings.filterwarnings("ignore")

***# Train file will be used for training the model, i.e. our model will learn from this file. It contains all the***

***# independent variables and the target variable. Test file contains all the independent variables, but not the target***

***# variable. We will apply the model to predict the target variable for the test data. Sample submission file contains***

***# the format in which we have to submit our predictions. Reading data***

train = pd.read\_csv('train\_u6lujuX\_CVtuZ9i.csv')

test = pd.read\_csv('test\_Y3wMUE5\_7gLdaTN.csv')

***# Let’s make a copy of train and test data so that even if we***

***# have to make any changes in these datasets we would not lose the original datasets.***

train\_original = train.copy()

test\_original = test.copy()

train.keys()

print(train.columns)

print(test.columns)

***# print data\_types for each variables***

print(train.dtypes)

print(train.shape, test.shape)

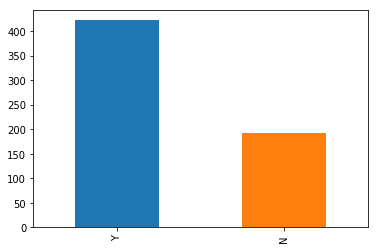
***# Analysis***

train['Loan\_Status'].value\_counts()

***# Normalise can be set to true to print the proportions instead of Numbers.***

train['Loan\_Status'].value\_counts(normalize=True)

train['Loan\_Status'].value\_counts().plot.bar()



***# visualize each variable separately***

***# Categorical features: These features have categories (Gender, Married, Self\_Employed, Credit\_History, Loan\_Status)***

***# Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property\_Area)***

***# Numerical features: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount,***

***# Loan\_Amount\_Term)***

plt.figure(1)

plt.subplot(221)

train['Gender'].value\_counts(normalize=True).plot.bar(figsize=(20, 10), title='Gender')

plt.subplot(222)

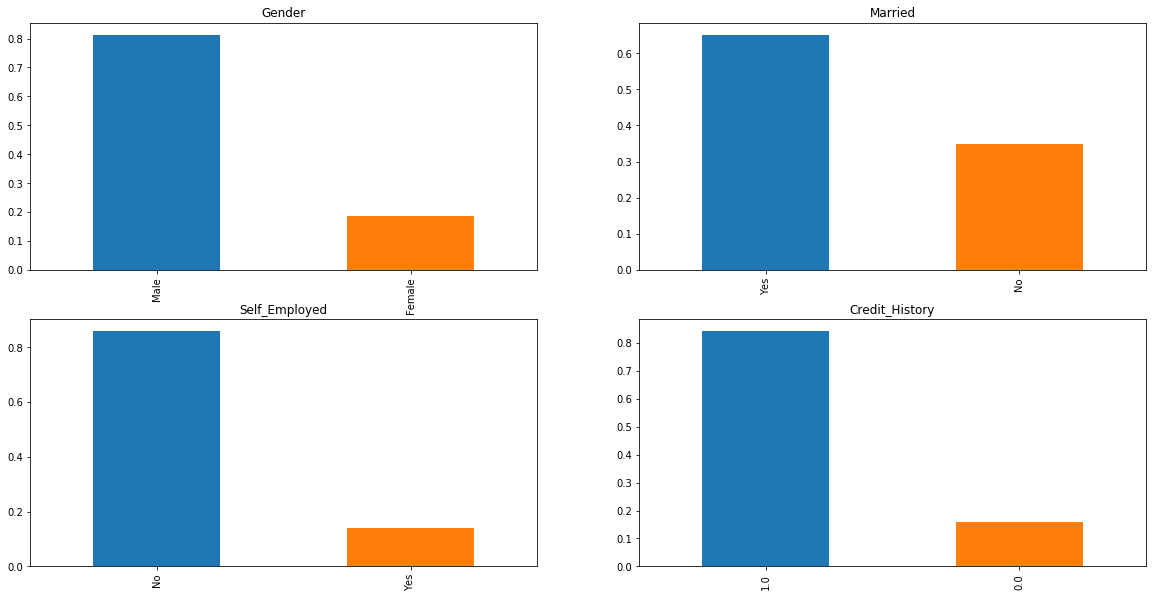
train['Married'].value\_counts(normalize=True).plot.bar(figsize=(20, 10), title='Married')

plt.subplot(223)

train['Self\_Employed'].value\_counts(normalize=True).plot.bar(figsize=(20, 10), title='Self\_Employed')

plt.subplot(224)

train['Credit\_History'].value\_counts(normalize=True).plot.bar(figsize=(20, 10), title='Credit\_History')



plt.show()

***# let’s visualize the ordinal variables.***

plt.figure(1)

plt.subplot(131)

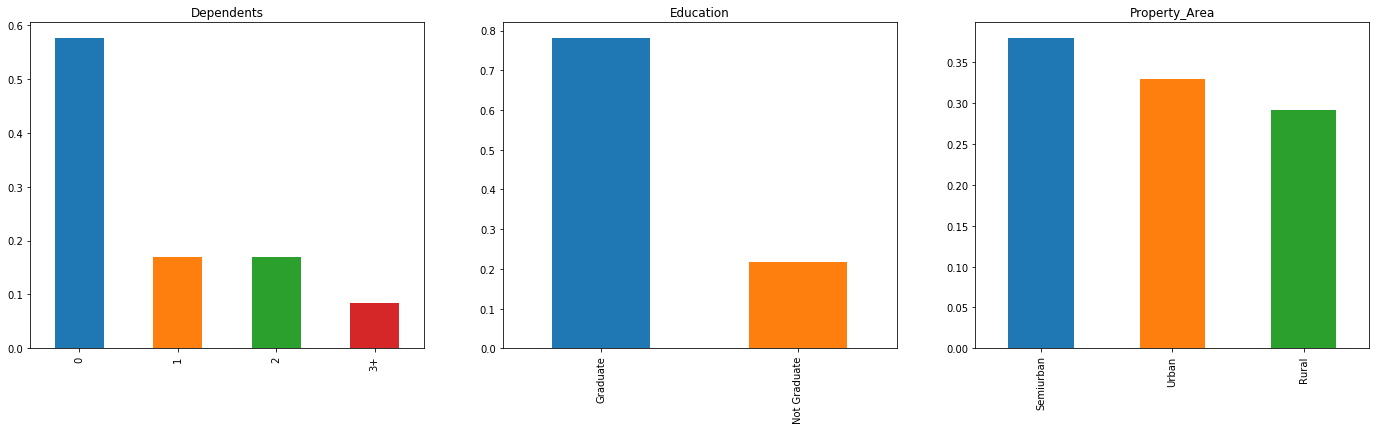
train['Dependents'].value\_counts(normalize=True).plot.bar(figsize=(24, 6), title='Dependents')

plt.subplot(132)

train['Education'].value\_counts(normalize=True).plot.bar(figsize=(24, 6), title='Education')

plt.subplot(133)

train['Property\_Area'].value\_counts(normalize=True).plot.bar(figsize=(24, 6), title='Property\_Area')



***# Lets visualise Numerical data***

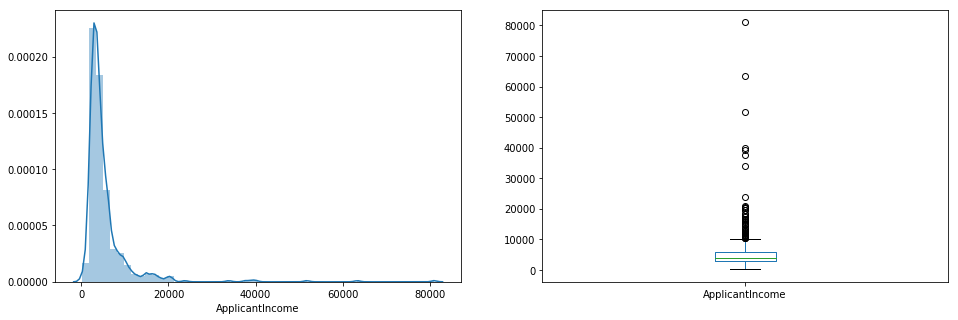
plt.figure(1)

plt.subplot(121)

sns.distplot(train['ApplicantIncome']);

plt.subplot(122)

train['ApplicantIncome'].plot.box(figsize=(16, 5))



***# The boxplot confirms the presence of a lot of extreme values.***

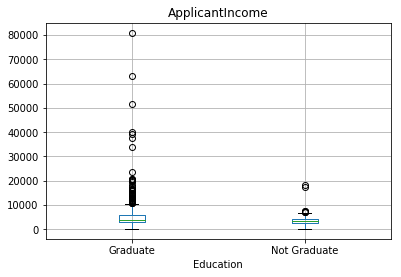
***# This can be attributed to the income disparity in the society.***

***# this can be driven by the fact that we are looking at people with different education levels.***

***# Let us segregate them by Education:***

train.boxplot(column='ApplicantIncome', by='Education')

plt.suptitle("")



***# Let’s look at the Coapplicant income distribution.***

plt.figure(1)

plt.subplot(121)

sns.distplot(train['CoapplicantIncome'])

plt.subplot(122)

train['CoapplicantIncome'].plot.box(figsize=(16, 5))



***# Let’s look at the distribution of LoanAmount variable.***

plt.figure(1)

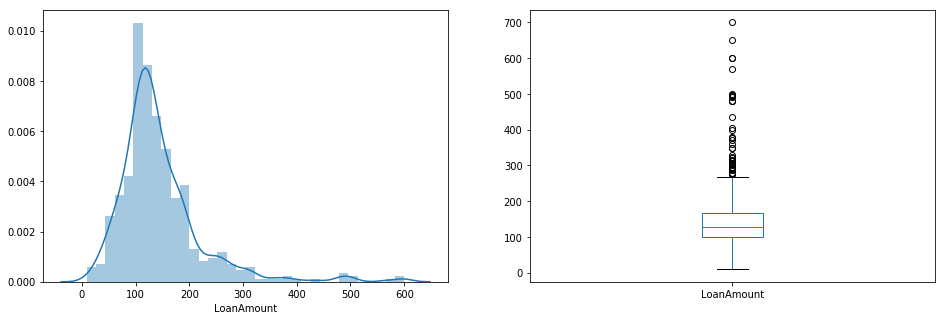
plt.subplot(121)

df = train.dropna()

sns.distplot(df['LoanAmount']);

plt.subplot(122)

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



***# Missing Values and Outliers Treatements***

train.info()

train.isnull().sum()

***# There are missing values in Gender, Married, Dependents, Self\_Employed, LoanAmount,***

***# Loan\_Amount\_Term and Credit\_History features.***

# 1) We will treat the missing values in all the features one by one.

# 2) We can consider these methods to fill the missing values:

# a)For numerical variables: imputation using mean or median

# b)For categorical variables: imputation using mode

# There are very less missing values in Gender, Married, Dependents, Credit\_History and Self\_Employed features so,

# we can fill them using the mode of the features.

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.isnull().sum()

***# let’s try to find a way to fill the missing values in Loan\_Amount\_Term.***

***# We will look at the value count of the Loan amount term variable.***

train['Loan\_Amount\_Term'].value\_counts()

***# It can be seen that in loan amount term variable, the value of 360 is repeating the most. So we will replace the***

***# missing values in this variable using the mode of this variable.***

train['Loan\_Amount\_Term'].fillna(train['Loan\_Amount\_Term'].mode()[0], inplace=True)

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

train.isnull().sum()

***# As we can see that all the missing values have been filled in the train dataset. Let’s fill all the missing values in***

***# the test dataset too with the same approach.***

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)

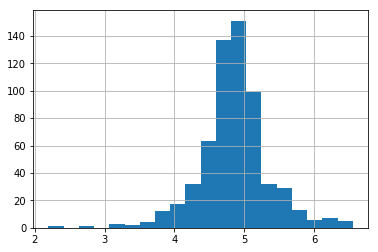
test['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)

***# Let’s visualize the effect of log transformation. We will do the similar changes to the test file simultaneously.***

train['LoanAmount\_log'] = np.log(train['LoanAmount'])

train['LoanAmount\_log'].hist(bins=20)

test['LoanAmount\_log'] = np.log(test['LoanAmount'])



***# Bivariate Analysis***

***# Categorical Independent Variable vs Target Variable***

***# First of all we will find the relation between target variable and categorical independent variables.***

***# Let us look at the stacked bar plot now which will give us the proportion of approved and unapproved loans.***

train.describe()

train.shape

train.dropna()

train.shape

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))



***# It can be inferred that the proportion of male and female applicants is more or less same***

***# for both approved and unapproved loans.***

***# Now let us visualize the remaining categorical variables vs target variable.***

Married = pd.crosstab(train['Married'], train['Loan\_Status'])

Dependents = pd.crosstab(train['Dependents'], train['Loan\_Status'])

Education = pd.crosstab(train['Education'], train['Loan\_Status'])

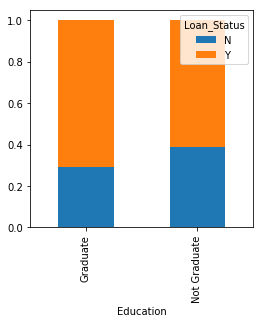
Self\_Employed = pd.crosstab(train['Self\_Employed'], train['Loan\_Status'])

Married.div(Married.sum(1).astype(float), axis=0).plot(kind='bar', stacked='True', figsize=(4, 4))

Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind='bar', stacked='True', figsize=(4, 4))

Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4, 4))

Self\_Employed.div(Self\_Employed.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4, 4))

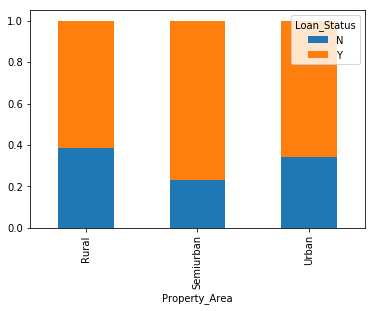
***# Lets look at the relationship between remaining categorical independent variables and Loan\_Status.***

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))

Property\_Area.div(Property\_Area.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)

***# let’s visualize numerical independent variables with respect to target variable.***

***train.groupby('Loan\_Status')['ApplicantIncome'].mean().plot.bar()***



bins = [0, 2500, 4000, 6000, 81000]

group = ['HIgh', 'Average', 'Low', 'Very high']

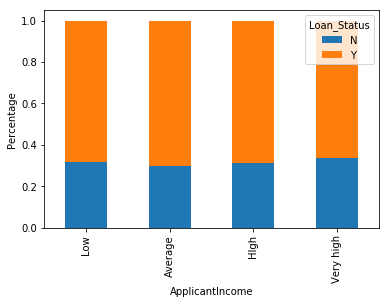
train['Income\_bin'] = pd.cut(train['ApplicantIncome'], bins, right=True, labels=group)

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')

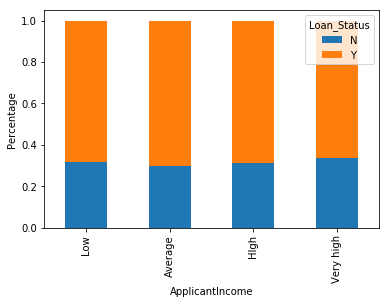


Income\_bin = pd.crosstab(train['Income\_bin'], train['Loan\_Status'])

Income\_bin.div(Income\_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)

plt.xlabel('ApplicantIncome')

plt.ylabel('Percentage')



***# We will analyze the coapplicant income and loan amount variable in similar manner.***

bins = [0, 1000, 3000, 42000]

group = ['Low', 'Average', 'High']

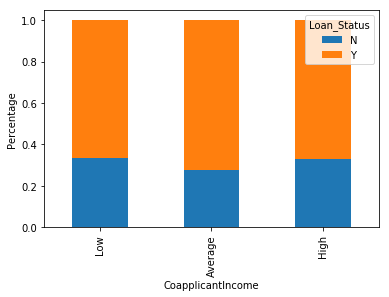
train['Coapplicant\_Income\_bin'] = pd.cut(train['CoapplicantIncome'], bins, include\_lowest=True, labels=group)

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')



***# Let us combine the Applicant Income and Coapplicant Income and see the combined effect of***

***# Total Income on the Loan\_Status.***

train['Total\_Income'] = train['ApplicantIncome'] + train['CoapplicantIncome']

bins = [0, 2500, 4000, 6000, 81000]

group = ['Very High', 'High', 'Low', 'Average']

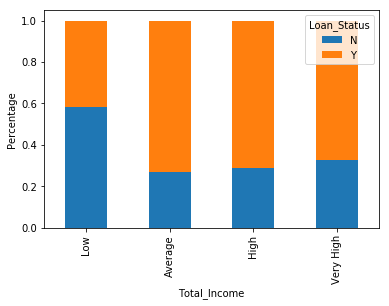
train['Total\_Income\_bin'] = pd.cut(train['Total\_Income'], bins, labels=group)

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')



***# Let’s visualize the Loan amount variable.***

bins = [0, 100, 200, 700]

group = ['Average', 'Low', 'High']

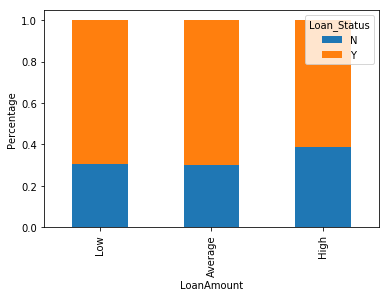
train['LoanAmount\_bin'] = pd.cut(train['LoanAmount'], bins, labels=group)

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)

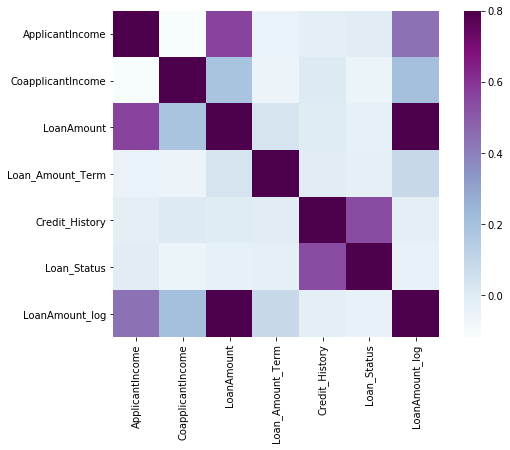
***# lets look at the correlation between all the numerical variables***

***# The variables with darker color means their correlation is more.***

matrix = train.corr()

f, ax = plt.subplots(figsize=(9, 6))

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



***# Model Building : Part I***

***# Let us make our model to predict the target variable.***

***# We will start with Logistic Regression which is used for predicting binary outcome.***

***# Lets drop the Loan\_ID variable as it do not have any effect on the loan status***

train = train.drop('Loan\_ID', axis=1)

test = test.drop('Loan\_ID', axis=1)

X = train.drop('Loan\_Status', 1)

y = train.Loan\_Status

***# we will make dummy variables for the categorical variables***

X = pd.get\_dummies(X)

train = pd.get\_dummies(train)

test = pd.get\_dummies(test)

***# We will use the train\_test\_split function from sklearn to divide our train dataset.***

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)

***# The dataset has been divided into training and validation part.***

***# Let us import LogisticRegression and accuracy\_score from sklearn and fit the logistic regression model.***

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

model = LogisticRegression()

model.fit(x\_train, y\_train)

***# Let’s predict the Loan\_Status for validation set and calculate its accuracy.***

pred\_cv = model.predict(x\_cv)

***# Let us calculate how accurate our predictions are by calculating the accuracy.***

accuracy\_score(y\_cv, pred\_cv)

***# Let’s make predictions for the test dataset.***

pred\_test = model.predict(test)

***# Lets import the submission file which we have to submit on the solution checker.***

submission = pd.read\_csv("Sample\_Submission\_ZAuTl8O\_FK3zQHh.csv")

***# We only need the Loan\_ID and the corresponding Loan\_Status for the final submission.***

***# we will fill these columns with the Loan\_ID of test dataset and the predictions that we made***

submission['Loan\_Status'] = pred\_test

submission['Loan\_ID'] = test\_original['Loan\_ID']

***# Logistic Regression using stratified k-folds cross validation***

from sklearn.model\_selection import StratifiedKFold

***# Now let’s make a cross validation logistic model with stratified 5 folds and make predictions for test dataset.***

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]

***# The mean validation accuracy for this model turns out to be 0.81. Let us visualize the roc curve.***

from sklearn import metrics

fpr, tpr, \_ = metrics.roc\_curve(yvl, pred)

auc = metrics.roc\_auc\_score(yvl, pred)

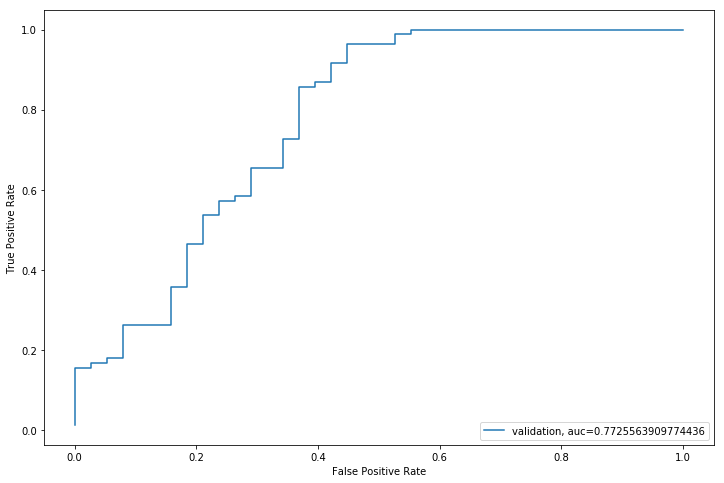
plt.figure(figsize=(12, 8))

plt.plot(fpr, tpr, label="validation, auc=" + str(auc))

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc=4)



submission['Loan\_Status'] = pred\_test

submission['Loan\_ID'] = test\_original['Loan\_ID']

***# Remember we need predictions in Y and N. So let’s convert 1 and 0 to Y and N.***

submission['Loan\_Status'].replace(0, 'N', inplace=True)

submission['Loan\_Status'].replace(1, 'Y', inplace=True)

***# Lets convert the submission to .csv format and make submission to check the accuracy on the leaderboard.***

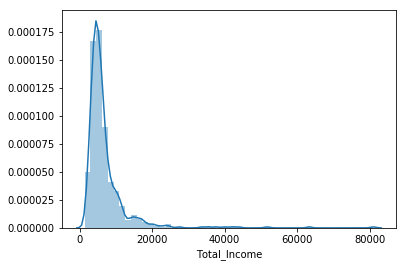
pd.DataFrame(submission, columns=['Loan\_ID', 'Loan\_Status']).to\_csv('Logistic1.csv')

train['Total\_Income'] = train['ApplicantIncome'] + train['CoapplicantIncome']

test['Total\_Income'] = test['ApplicantIncome'] + test['CoapplicantIncome']

***# Let’s check the distribution of Total Income.***

sns.distplot(train['Total\_Income']);

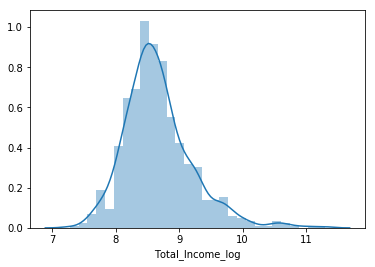


***# let’s take the log transformation to make the distribution normal.***

train['Total\_Income\_log'] = np.log(train['Total\_Income'])

sns.distplot(train['Total\_Income\_log']);

test['Total\_Income\_log'] = np.log(test['Total\_Income'])



***# Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided.***

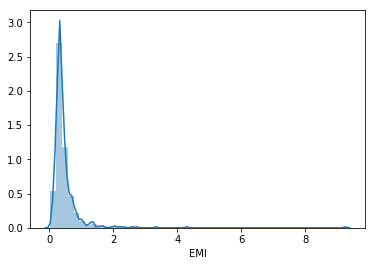
***# Let’s create the EMI feature now.***

train['EMI'] = train['LoanAmount'] / train['Loan\_Amount\_Term']

test['EMI'] = test['LoanAmount'] / test['Loan\_Amount\_Term']

***# Let’s check the distribution of EMI variable.***

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

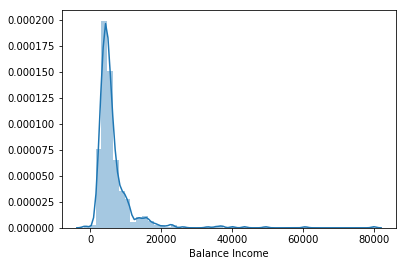


***# Let us create Balance Income feature now and check its distribution.***

train['Balance Income'] = train['Total\_Income'] - (train['EMI'] \* 1000) # Multiply with 1000 to make the units equal

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

***# Logistic Regression***

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]

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.csv')

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)

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('Decision Tree.csv')

***# Random Forest***

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)

***# We will tune the max\_depth and n\_estimators parameters. max\_depth decides the maximum depth of the tree and***

***# n\_estimators decides the number of trees that will be used in random forest model.***

from sklearn.model\_selection import GridSearchCV

***# Provide range for max\_depth from 1 to 20 with an interval of 2 and from 1 to 200 with an***

***# interval of 20 for n\_estimators***

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

x\_train, x\_cv, y\_train, y\_cv = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

***# 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)

***# Estimating the optimized value***

grid\_search.best\_estimator\_

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=41, n\_jobs=1,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

***# So, the optimized value for the max\_depth variable is 3 and for n\_estimator is 41.***

***# Now let’s build the model using these optimized values.***

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=3, n\_estimators=41)

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)

pred2 = model.predict\_proba(test)[:, 1]

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('Random Forest.csv')

***# Let us find the feature importance now, i.e. which features are most important for this problem.***

***# We will use feature importances attribute of sklearn to do so.***

importances = pd.Series(model.feature\_importances\_, index=X.columns)

importances.plot(kind='barh', figsize=(12, 8))

plt.show()

