**Loan Eligiblity Prediction using Logistic Regression**

**Aim:**

Develop a predictive model using logistic regression to assess the likelihood of loan approval based on applicant information.

Description:

In this mini-project, there will bea dataset containing information about loan applicants, including attributes such as income, credit history, loan amount, etc. The aim is to build a logistic regression model that can predict whether a loan application is likely to be approved or not.

**Key steps:**

Data Collection: Obtain a dataset containing information about loan applicants. You can use publicly available datasets from sources like Kaggle or UCI Machine Learning Repository.

1. Data Preprocessing: Explore the dataset to understand its structure and identify any missing values or inconsistencies. Perform data cleaning, including handling missing values, encoding categorical variables, and scaling numerical features if necessary.
2. Feature Engineering: Create new features or transform existing ones to improve the predictive power of the model. For example, you can derive features like debt-to-income ratio, loan-to-income ratio, etc.
3. Model Building: Build a logistic regression model using the processed dataset. Split the data into training and testing sets and train the model on the training data.
4. Model Evaluation: Evaluate the performance of the logistic regression model using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC curve.
5. Hyperparameter Tuning: Fine-tune the hyperparameters of the logistic regression model using techniques like grid search or random search to optimize its performance.
6. Model Interpretation: Interpret the coefficients of the logistic regression model to understand the factors that influence loan approval decisions.
7. Deployment: Deploy the trained logistic regression model in a production environment or create a user interface (UI) for end-users to interact with the model.
8. Documentation: Document the entire process, including data preprocessing steps, feature engineering techniques, model building process, evaluation metrics, and deployment instructions.

**Logistic regression:**Logistic regression is a statistical method used for binary classification tasks, where the outcome or target variable is categorical and has two possible outcomes (e.g., yes/no, true/false, 0/1). Despite its name, logistic regression is primarily used for classification rather than regression tasks.

In logistic regression, the goal is to model the probability that an instance belongs to a particular class. It predicts the probability of occurrence of an event by fitting data to a logistic curve, also known as the sigmoid function. The output of logistic regression is a probability score between 0 and 1, which can be interpreted as the likelihood of an instance belonging to the positive class (class 1).

To make predictions, logistic regression applies a threshold (usually 0.5) to the predicted probability score. If the probability is greater than the threshold, the instance is classified as belonging to the positive class; otherwise, it is classified as belonging to the negative class.

Logistic regression is widely used in various fields, including healthcare (e.g., predicting disease outcomes), finance (e.g., credit risk assessment), marketing (e.g., customer churn prediction), and many others, due to its simplicity, interpretability, and effectiveness for binary classification tasks.

**Decription of dataset:**

This dataset appears to be related to loan applications and includes various attributes about the applicants, the loan amounts, and the loan approval status. Here's a breakdown of the columns:

1. Loan\_ID: Unique identifier for each loan application.
2. Gender: Gender of the applicant (Male/Female).
3. Married: Marital status of the applicant (Yes/No).
4. Dependents: Number of dependents the applicant has (0, 1, 2, 3+).
5. Education: Educational qualification of the applicant (Graduate/Not Graduate).
6. Self\_Employed: Whether the applicant is self-employed (Yes/No).
7. ApplicantIncome: Income of the applicant.
8. CoapplicantIncome: Income of the co-applicant.
9. LoanAmount: Loan amount requested (in thousands).

10.Loan\_Amount\_Term: Term of the loan (in months).

11.Credit\_History: Credit history of the applicant (1 indicates good credit history, 0 indicates bad credit history, missing values may indicate no credit history).

12.Property\_Area: The area where the property is located (Urban/Semiurban/Rural).

13.Loan\_Status: Whether the loan was approved (Y) or not (N).

### Observations:

1. Applicant Demographics:
   * Gender: The dataset predominantly features male applicants, with only one female applicant.
   * Marital Status: Most applicants are married.
   * Dependents: Applicants have a range of dependents from 0 to 3+.
2. Income:
   * Applicant and co-applicant incomes vary widely, indicating a diverse set of applicants in terms of earnings.
   * Some applicants have co-applicants contributing to the income, while others do not.

3.Loan Details:

* + Loan amounts requested range from very small amounts (e.g., 17) to large amounts (e.g., 349).
  + Loan terms are mostly 360 months (30 years), with a few exceptions.
  + Credit history is primarily good, with a few applicants having no credit history.

4.Property Area:

* + Applications are spread across urban, semiurban, and rural areas, with a majority from urban areas.

5. Loan Approval:

* + The loan status column indicates whether a loan was approved ('Y') or not ('N').

**Program Implementations ,Predictions , Visualisation:**

**Importing Necessary libraries:**

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk(''):

for filename in filenames:

print(os.path.join(dirname, filename))

import pandas as pd

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.axes

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")

**Load and import data**

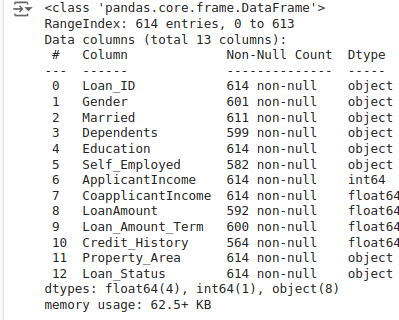
train = pd.read\_csv("train\_ctrUa4K.csv")

test = pd.read\_csv("test\_lAUu6dG.csv")

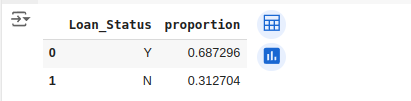
train\_original = train.copy()

test\_original = test.copy()

train.info()

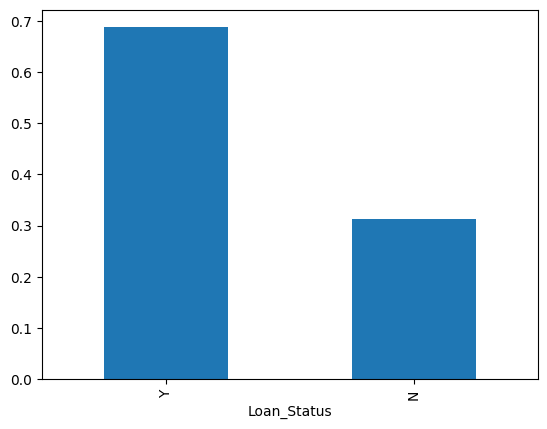


train["Loan\_Status"].value\_counts(normalize = True).reset\_index()

****

**Data Visulisation:**

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

****

plt.figure(1)

plt.subplot(221)

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

plt.subplot(222)

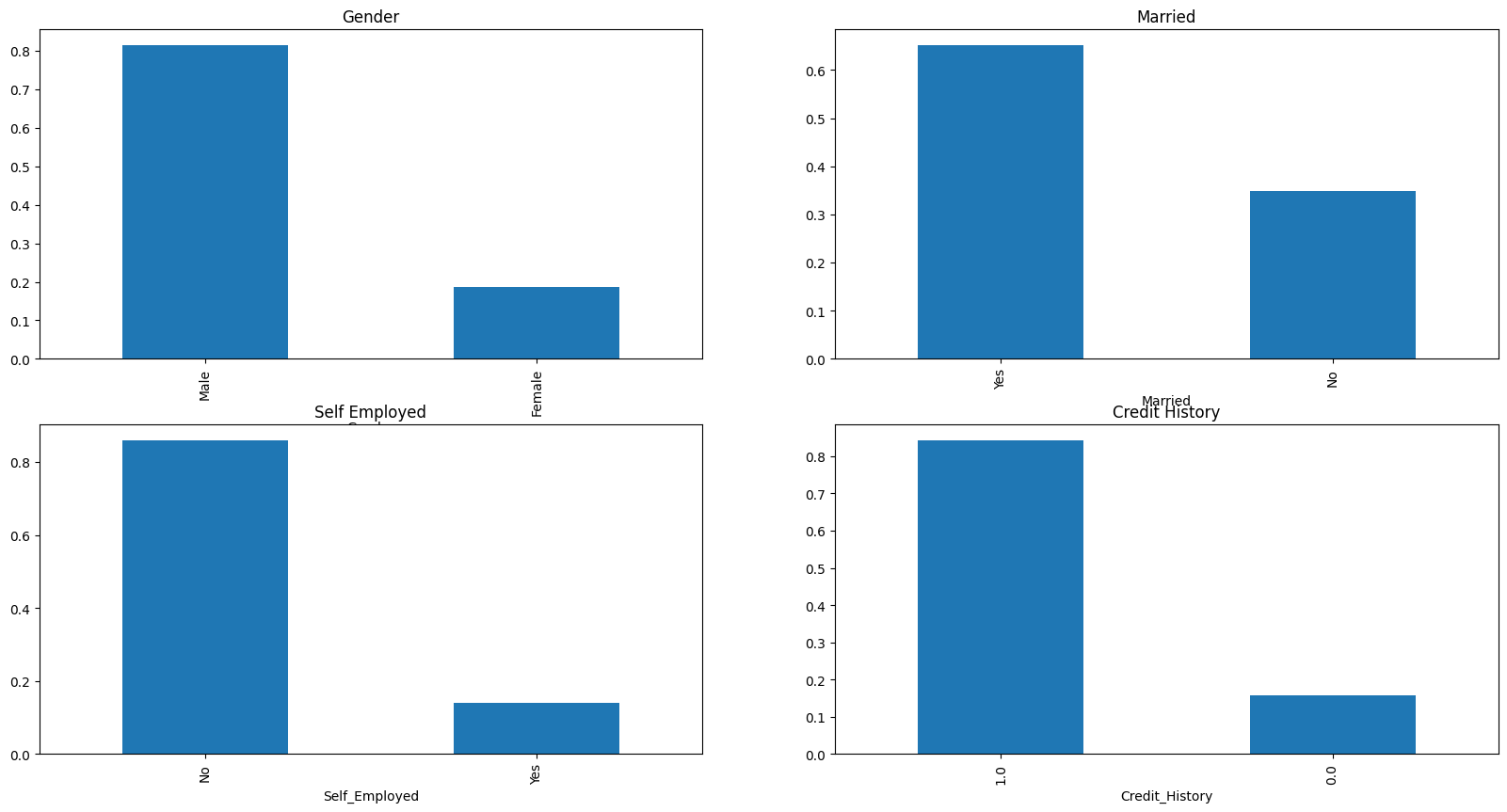
train['Married'].value\_counts(normalize = True).plot.bar(title = 'Married')

plt.subplot(223)

train['Self\_Employed'].value\_counts(normalize = True).plot.bar(title = 'Self Employed')

plt.subplot(224)

train['Credit\_History'].value\_counts(normalize = True).plot.bar(title = 'Credit History')

****

plt.figure(1)

plt.subplot(131)

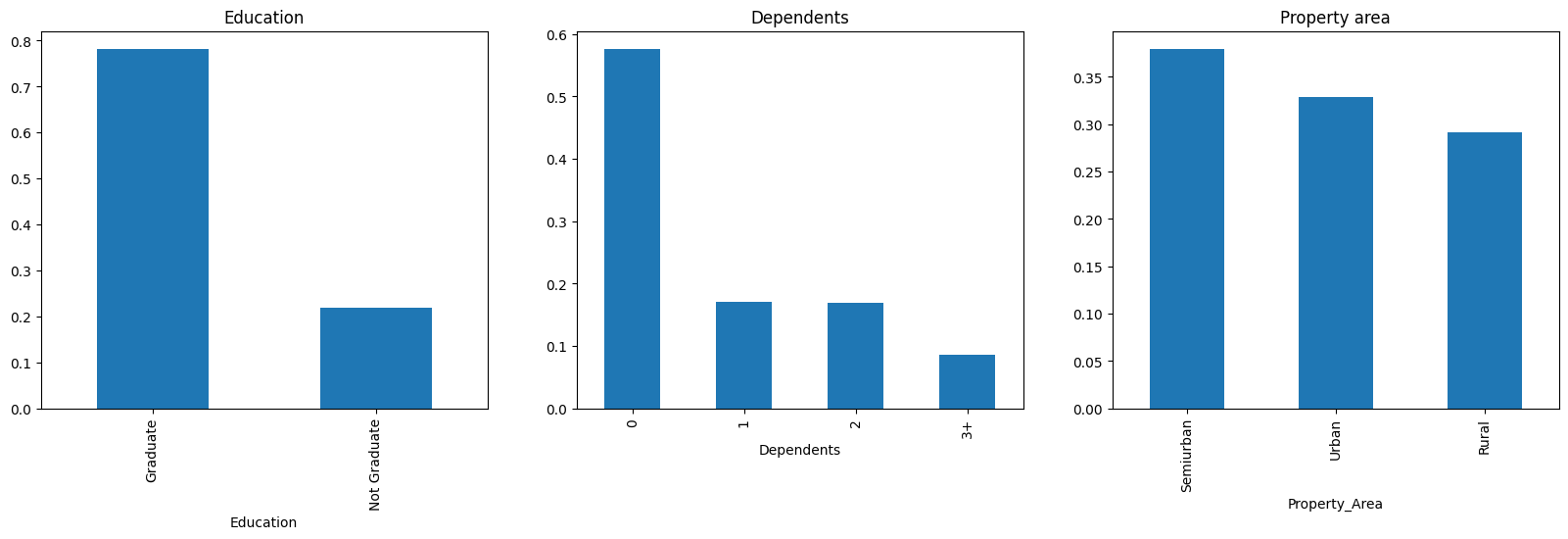
train['Education'].value\_counts(normalize = True).plot.bar(figsize = (20,5), title = 'Education')

plt.subplot(132)

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

plt.subplot(133)

train['Property\_Area'].value\_counts(normalize = True).plot.bar(title = 'Property area')



plt.figure(1)

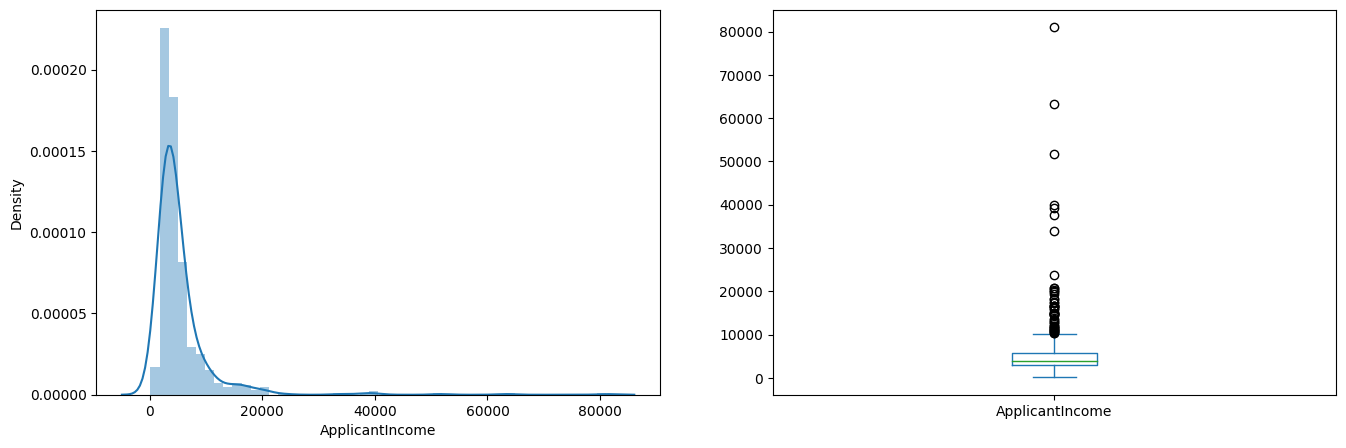
plt.subplot(121)

sns.distplot(train['ApplicantIncome'])

plt.subplot(122)

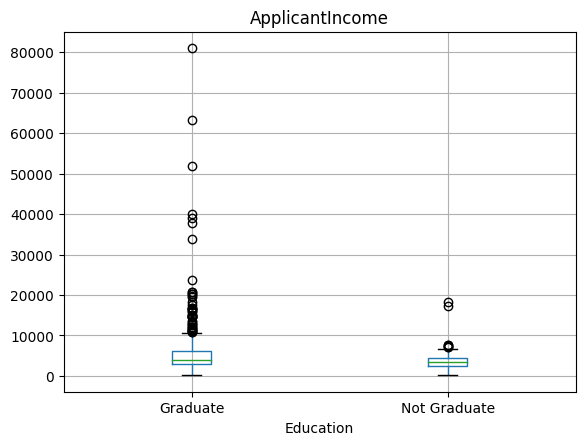
train['ApplicantIncome'].plot.box(figsize = (16,5))

plt.show()



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

plt.suptitle("")



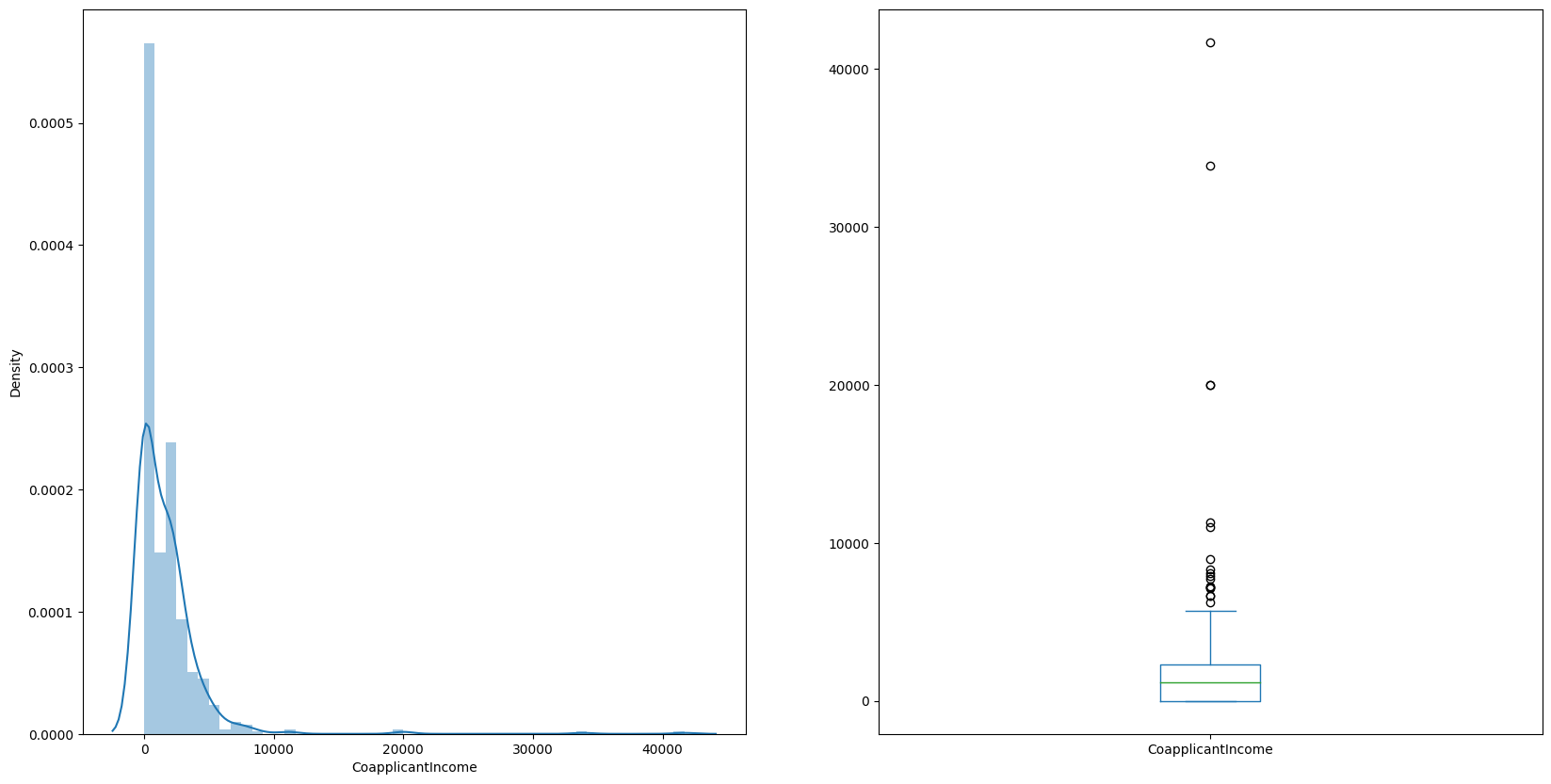
plt.figure(1)

plt.subplot(121)

sns.distplot(train['CoapplicantIncome'])

plt.subplot(122)

train['CoapplicantIncome'].plot.box(figsize =(20,10))



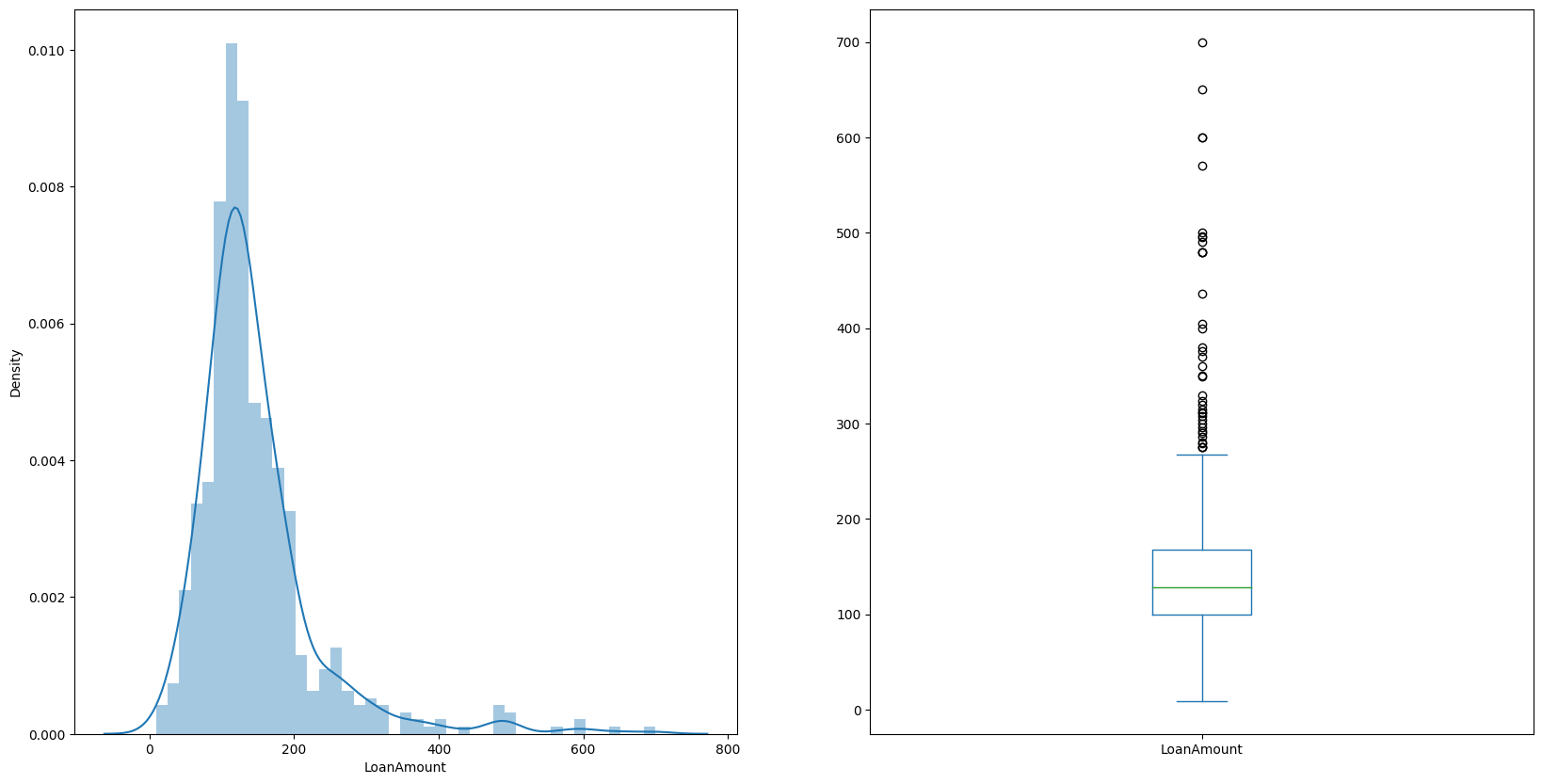
plt.figure(1)

plt.subplot(121)

sns.distplot(train['LoanAmount'])

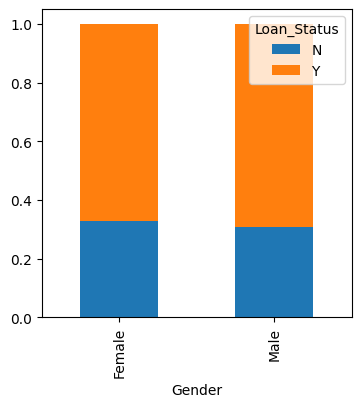
plt.subplot(122)

train['LoanAmount'].plot.box(figsize =(20,10))



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))



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

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

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

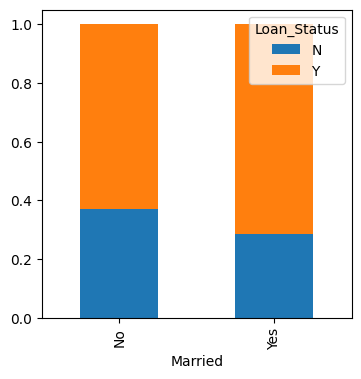
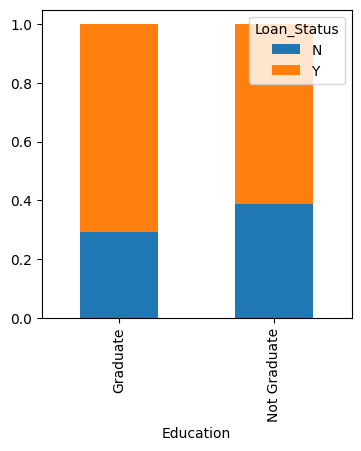
Self\_Employed = pd.crosstab(train['Self\_Employed'],train['Loan\_Status'])

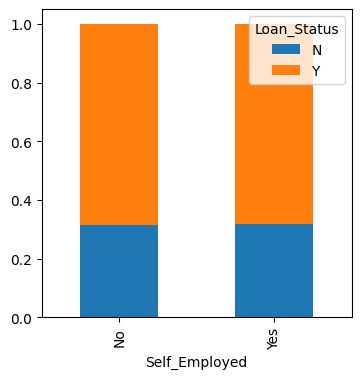
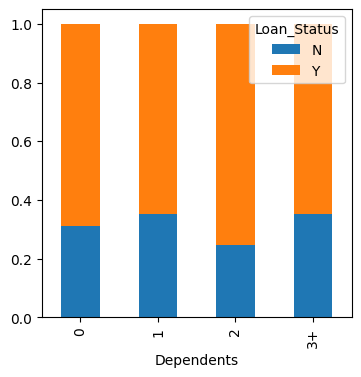
Married.div(Married.sum(1).astype(float), axis = 0).plot(kind = "bar", stacked = True, figsize = (4,4))

Education.div(Education.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))

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



bins = []

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

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

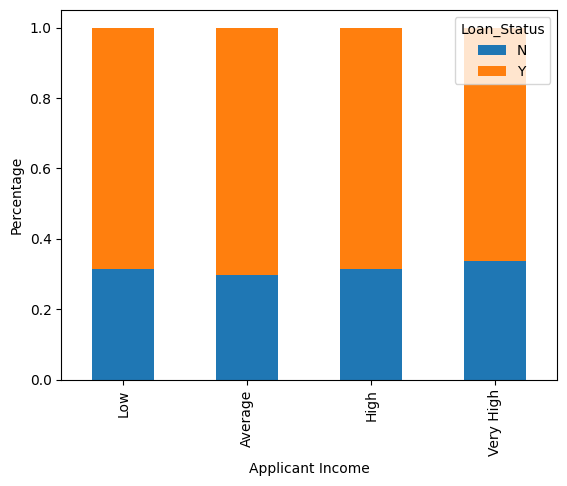
train['Income\_bin']=pd.cut(train['ApplicantIncome'], bins, labels = groups)

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("Applicant Income")

plt.ylabel("Percentage")



bins = [0,1000,3000,42000]

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

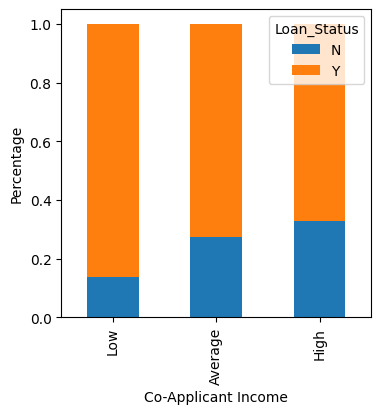
train['Coapplicant\_income\_bin']=pd.cut(train['CoapplicantIncome'], bins, labels = groups)

Coapplicant\_Income = pd.crosstab(train['Coapplicant\_income\_bin'],train['Loan\_Status'])

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

plt.xlabel("Co-Applicant Income")

plt.ylabel("Percentage")



bins = [0,2500,4000,81000]

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

train['Totalincome'] = train['ApplicantIncome'] + train['CoapplicantIncome']

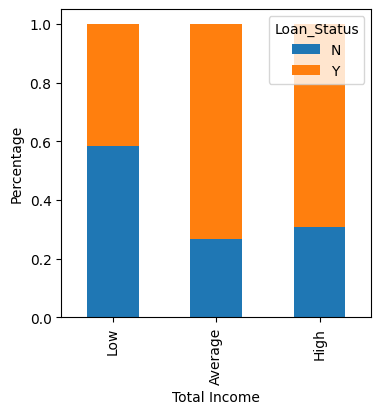
train['Total\_income\_bin']=pd.cut(train['Totalincome'], bins, labels = groups)

Total\_Income = pd.crosstab(train['Total\_income\_bin'],train['Loan\_Status'])

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

plt.xlabel("Total Income")

plt.ylabel("Percentage")



bins=[0,100,200,700]

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

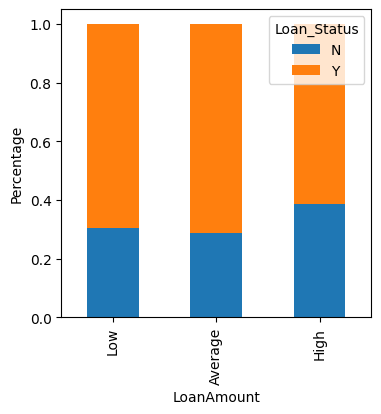
train['LoanAmount\_bin']=pd.cut(train['LoanAmount'],bins,labels=group)

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, figsize = (4,4))

plt.xlabel('LoanAmount')

P = plt.ylabel('Percentage')



# Exclude non-numeric columns

numeric\_train = train.select\_dtypes(include=['float64', 'int64'])

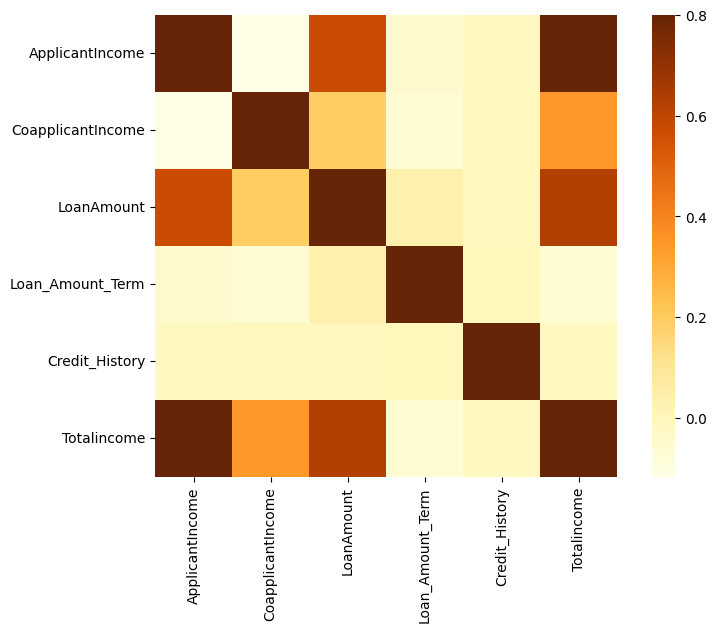
# Calculate correlation matrix

matrix = numeric\_train.corr()

# Plot heatmap

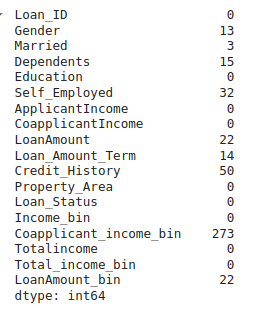
f, ax = plt.subplots(figsize=(9, 6))

sns.heatmap(matrix, vmax=0.8, cmap="YlOrBr", square=True)



**Missing value Imputation:**

train.isnull().sum()

****

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

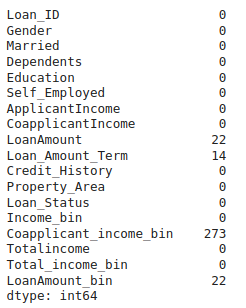
train['Married'].fillna(train['Married'].mode()[0], inplace = True)

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()

****

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

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

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)

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

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

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

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

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

x\_train.head()

x\_train.isnull().values.any()

****

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

model = LogisticRegression()

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



**Prediction and Accuracy:**

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

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

**Accuracy is Predicted As 0.8**

submission1=pd.read\_csv("sample\_submission\_49d68Cx.csv", index\_col=False)

submission1['Loan\_Status']=pred\_test

submission1['Loan\_ID']=test\_original['Loan\_ID']

**Conclusion:**

A predictive model for loan eligibility should weigh these factors appropriately to estimate the likelihood of loan approval. It is recommended to use logistic regression, decision trees, or machine learning algorithms like random forests or gradient boosting for creating robust predictive models, ensuring that the model is trained and validated on historical data to improve accuracy.