Internship Project Presentation

Presented by Parag Patil

Declaration

Student name: **Parag Jayant Patil**
Internal Guide name: **Prof. Ashru Jadhav**
External Guide name (Technokraft) :
Roll number: **46.**
Branch and Year: **TE-AI&DS**
College name: **K.K. Wagh Institute of Engineering Education and Research**
Project title: **Predictive Modeling for Loan Approval**

Contents

Things I Learned

01
Descriptive Statistics

Measures of central tendency:
mean, median, mode
Measures of dispersion:
variance, standard deviation, range
To summarize and describe datasets

02
Python Libraries

NumPy
Pandas
Scikit-learn
Matplotlib
Seaborn

03

Statistical Analysis

- Independent variable
- Dependent variable
- Target variable
- Univariate Analysis
- Bivariate Analysis

04

EDA (Exploratory Data Analysis)

- Data Cleaning and Preprocessing
- Missing value and Outlier detection
- Feature Engineering
- Evaluation Metrics for Classification

05

Model Building

- Logistic Regression
- Decision Tree Algorithm
- Random Forest Algorithm

Problem Definition

Title: Predictive Modeling for Loan Approval

Problem Statement:

The Dream Housing Finance company aims to automate the loan eligibility process for customers applying for home loans online. They want to identify eligible customer segments based on details provided in the application form such as gender, marital status, education, income, credit history, etc. This automation will help them target specific customer segments for loan approval.

Objective:

It is a classification problem where we have to predict whether a loan would be approved or not. In this problem, we have to predict discrete values based on a given set of independent variables (s).

Dataset:

The loan prediction dataset is structured as a CSV file and consists of several columns representing various attributes related to loan applications. Each row in the dataset corresponds to a single loan application, with each column providing specific details about that application. The goal is to predict whether a loan application will be approved or denied based on the provided attributes.

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	Credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

EDA and Pre-Processing

```
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
import warnings                     # To ignore any warnings
warnings.filterwarnings("ignore")

train=pd.read_csv("C:/Users/91862/Desktop/Int//train_ctrUa4K (1).csv")

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')
```

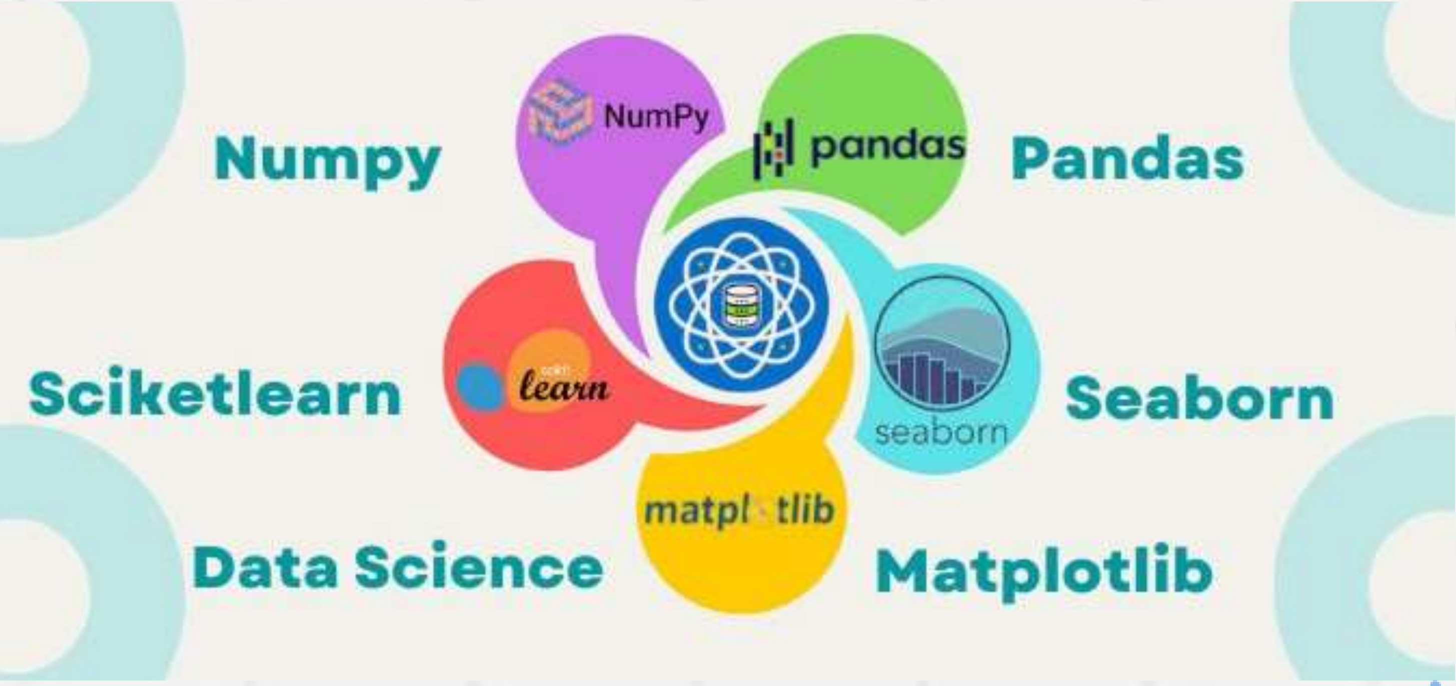
```
train.dtypes
```

```
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
```

```
(614, 13)
```

Importing Python Libraries and Reading a CSV file into a DataFrame, then displaying the column names, data types, and shape of the DataFrame.



```
#Univariate Analysis
```

```
#Independent variable(Categorical)
#Target Variable
train['Loan_Status'].value_counts()
```

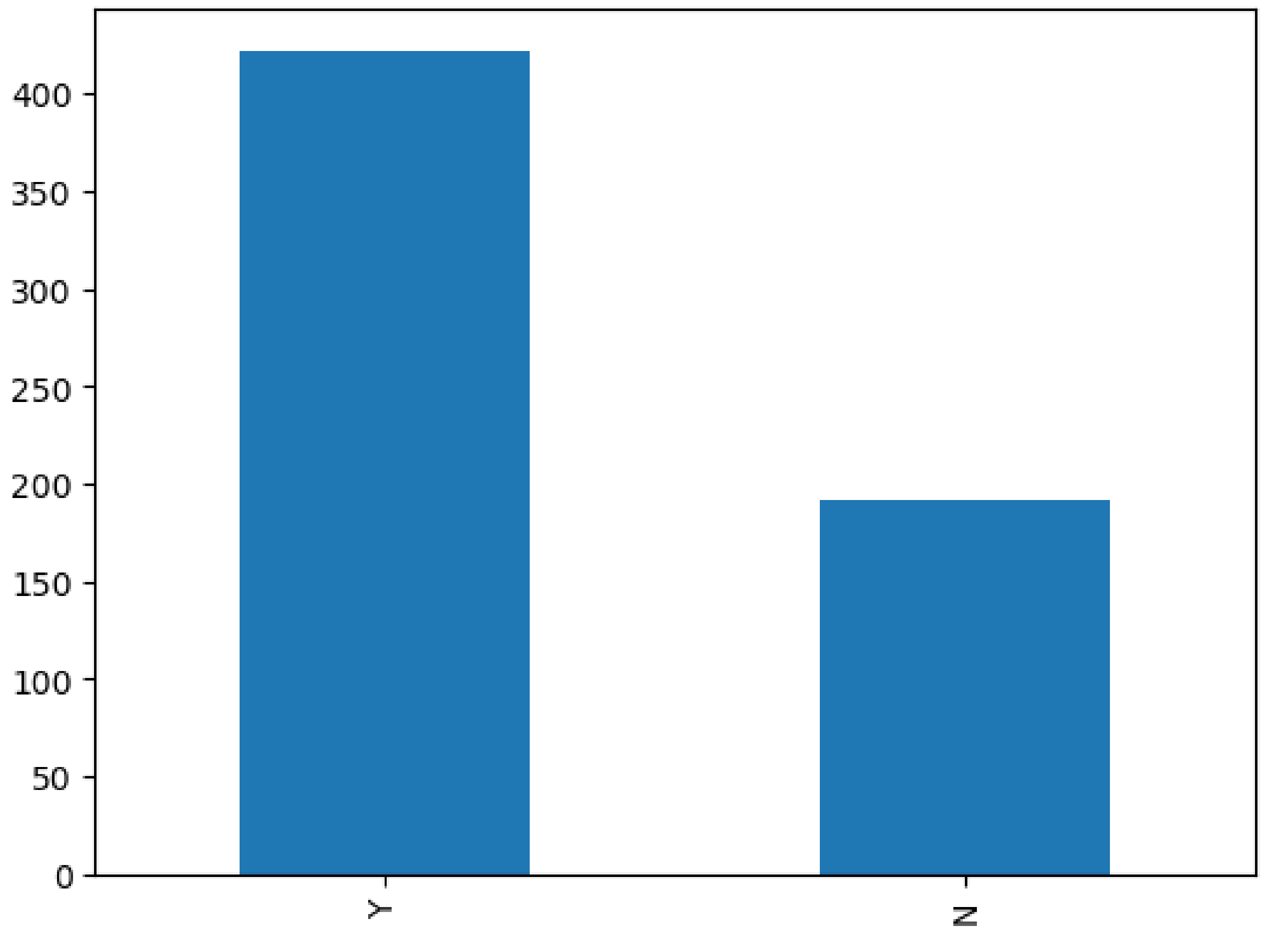
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

```
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: >
```



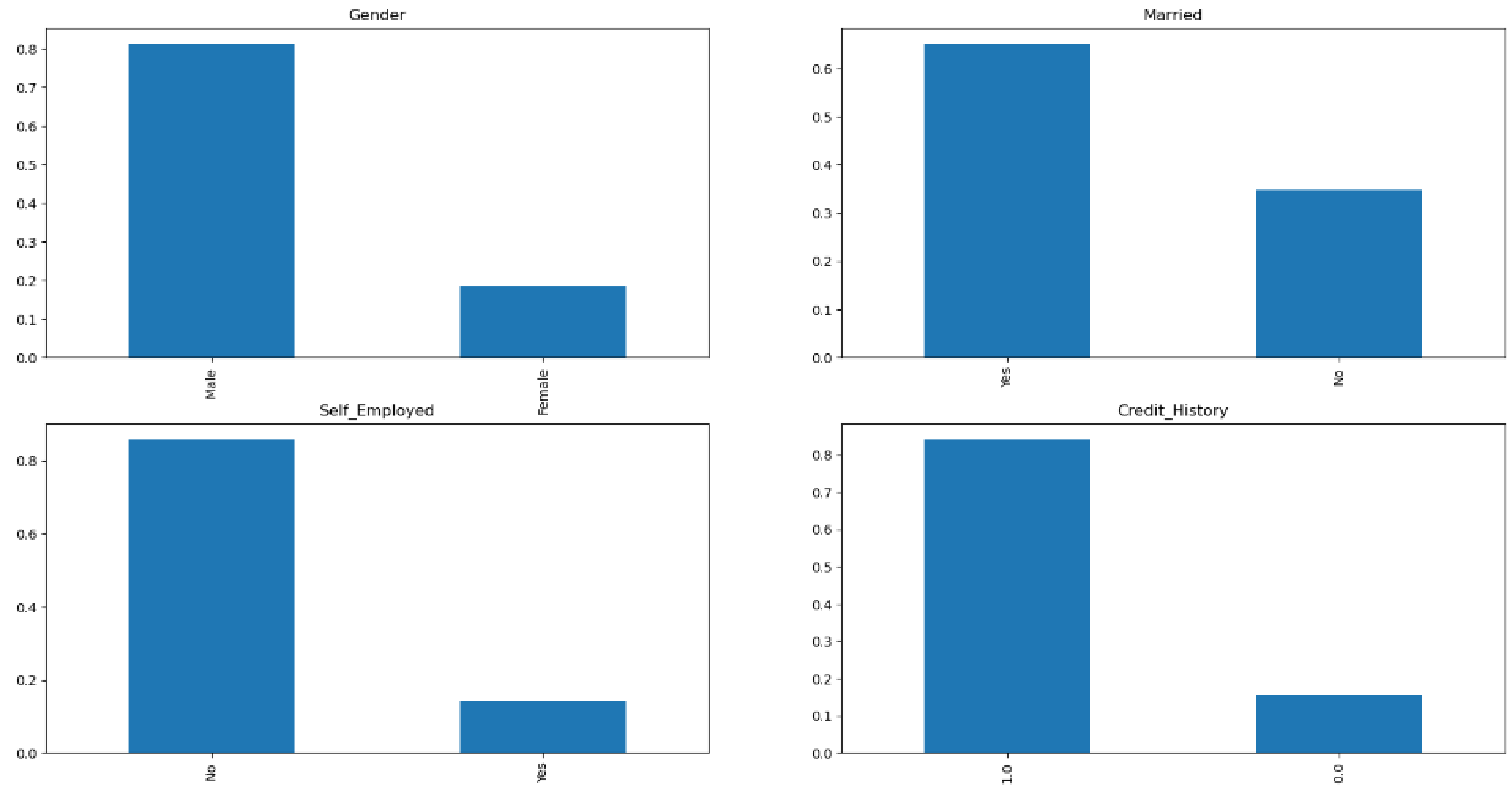
Univariate Analysis

Analyzing data where we analyze each variable individually. For categorical features, we will use bar plots to calculate the number of each category in a particular variable.

Target variable, i.e., Loan_Status. As it is a categorical variable, let us look at its frequency table, percentage distribution, and bar plot

422(around 69%) people out of 614 got the approval.

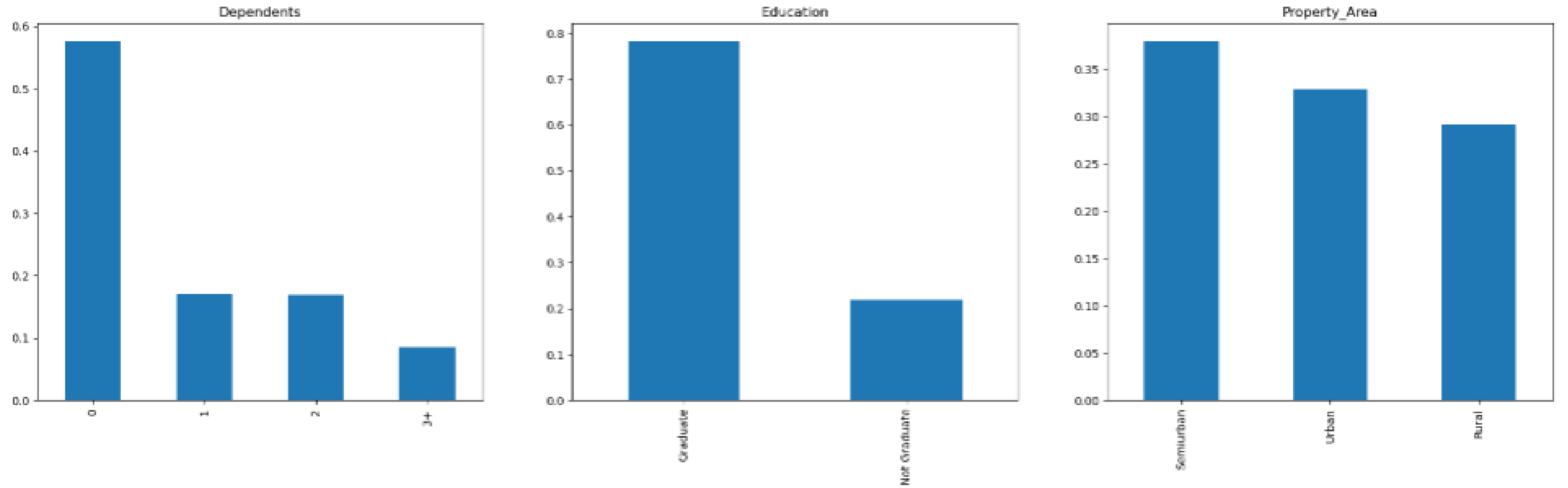
```
#Independent Variable (Categorical)
plt.subplot(221)
train['Gender'].value_counts(normalize=True).plot.bar(figsize=(20,10), title= 'Gender')
plt.subplot(222)
train['Married'].value_counts(normalize=True).plot.bar(title= 'Married')
plt.subplot(223)
train['Self_Employed'].value_counts(normalize=True).plot.bar(title= 'Self_Employed')
plt.subplot(224)
train['Credit_History'].value_counts(normalize=True).plot.bar(title= 'Credit_History')
plt.show()
```



Categorical features: These features have categories (Gender, Married, Self_Employed, Credit_History)

- 80% of applicants in the dataset are male.
- Around 65% of the applicants in the dataset are married.
- About 15% of applicants in the dataset are self-employed.
- About 85% of applicants have repaid their debts.

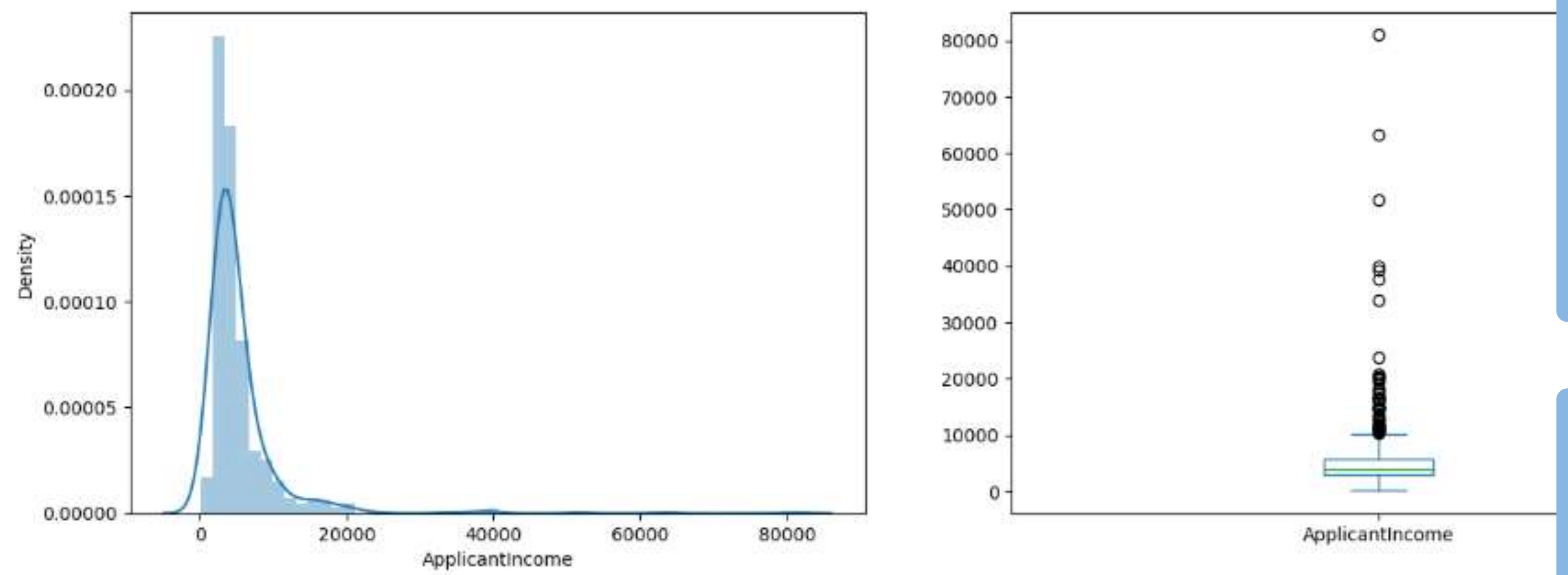
```
#Independent Variable (Ordinal)
plt.subplot(131)
train['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6),title='Dependents')
plt.subplot(132)
train['Education'].value_counts(normalize=True).plot.bar(title= 'Education')
plt.subplot(133)
train['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Property_Area')
plt.show()
```



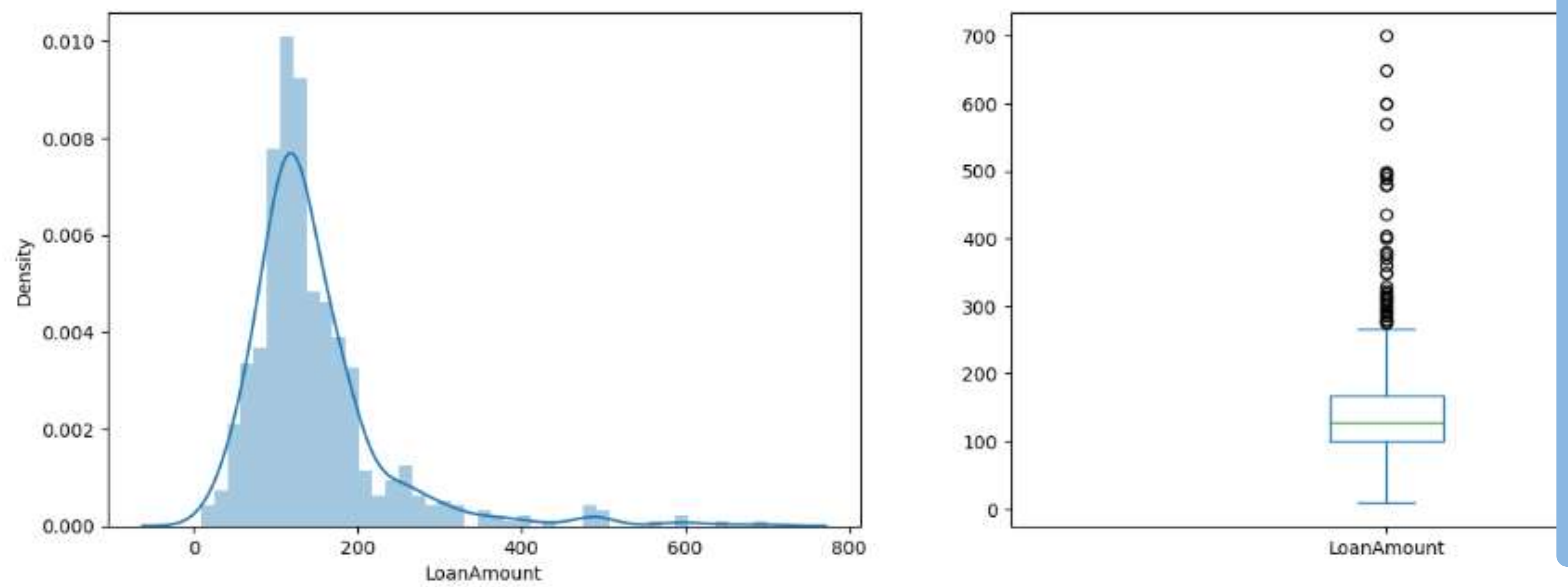
Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property_Area)

- Most of the applicants don’t have dependents.
- About 80% of the applicants are graduates.
- Most of the applicants are from semi-urban areas.

```python
#Independent Variable (Numerical)
plt.subplot(121)
sns.distplot(train['ApplicantIncome']);
plt.subplot(122)
train['ApplicantIncome'].plot.box(figsize=(16,5))
plt.show()
```




```python
#Independent Variable (Numerical)
plt.subplot(121)
df=train.dropna()
sns.distplot(train['LoanAmount']);
plt.subplot(122)
train['LoanAmount'].plot.box(figsize=(16,5))
plt.show()
```



Numerical features: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term)

- It can be inferred that most of the data in the distribution of applicant income are towards the left which means it is not normally distributed.

- It can be inferred that there are lot of outliers in the LoanAmount variable and the distribution is fairly normal.

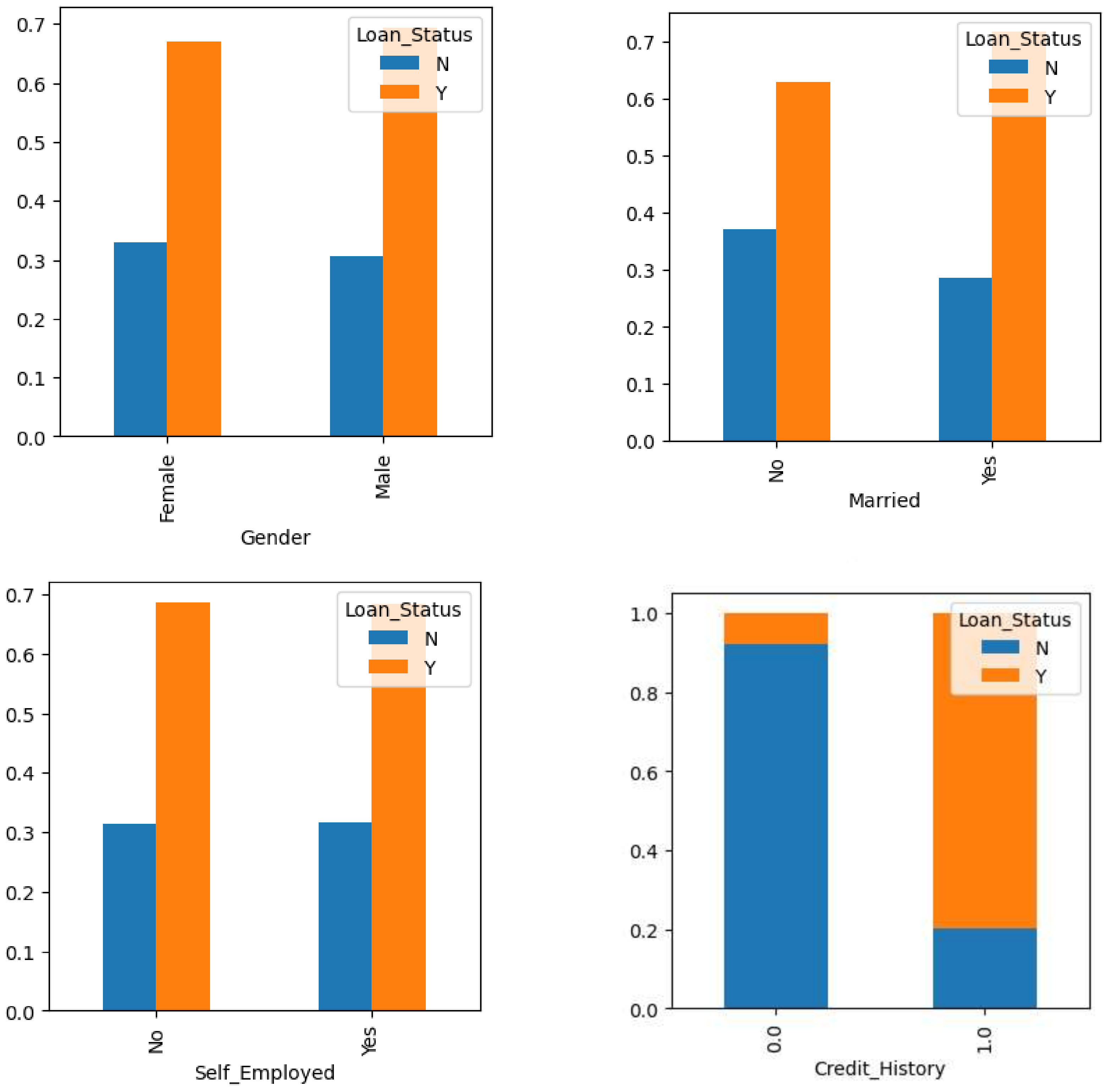
Bivariate Analysis

Assesses the relationship between two variables: for categorical and ordinal independent variables versus the target variable, it evaluates how each category or order influences the target, while for numerical independent variables versus the target variable, it examines how numerical values impact the target.

```
#Bivariate analysis
```

```
#Categorical Independent Variable vs Target Variable
Gender=pd.crosstab(train['Gender'],train['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
Married=pd.crosstab(train['Married'],train['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
Self_Employed=pd.crosstab(train['Self_Employed'],train['Loan_Status'])
Self_Employed.div(Self_Employed.sum(1).astype(float),axis=0).plot(kind="bar",figsize=(4,4))
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))
```

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

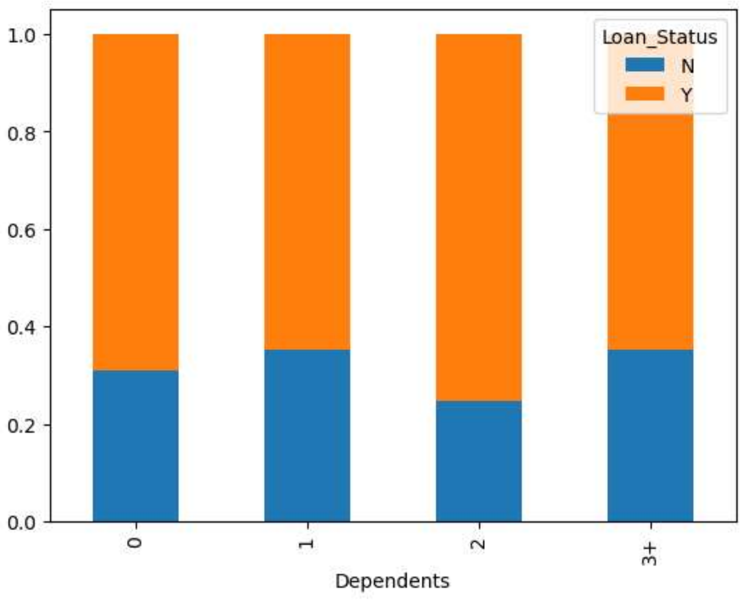


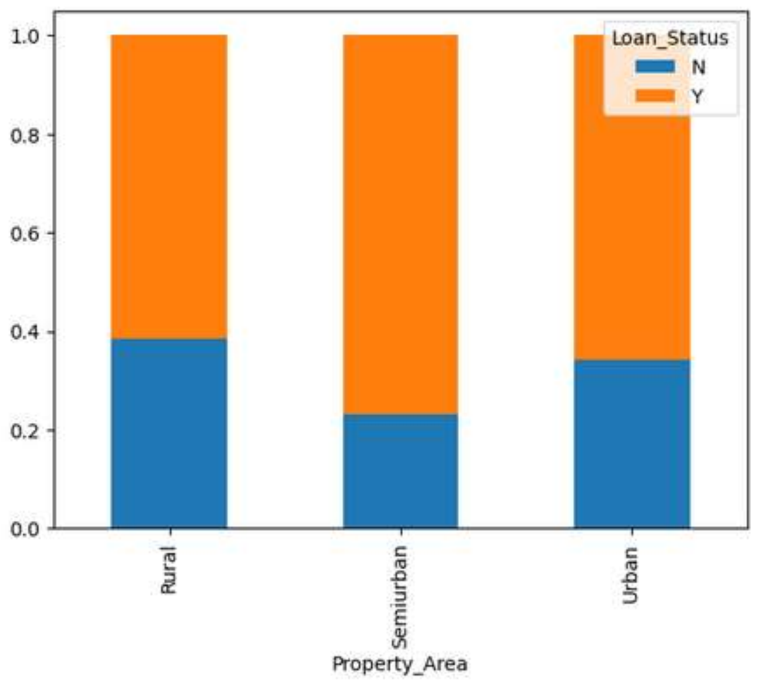
Categorical Independent Variable vs Target Variable

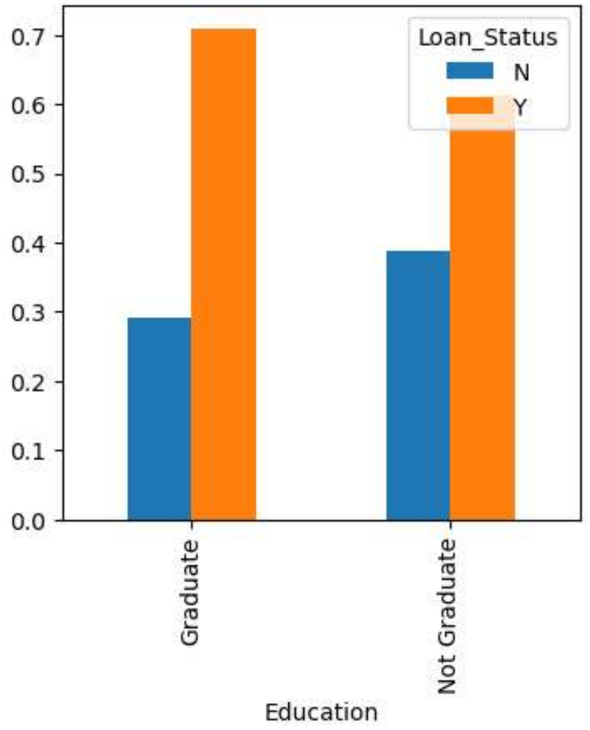
Ordinal Independent Variable vs Target Variable

```python
#Ordinal Independent Variable vs Target Variable
Dependents=pd.crosstab(train['Dependents'],train['Loan_Status'])
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
Education=pd.crosstab(train['Education'],train['Loan_Status'])
Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
Property_Area=pd.crosstab(train['Property_Area'],train['Loan_Status'])
Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
```

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







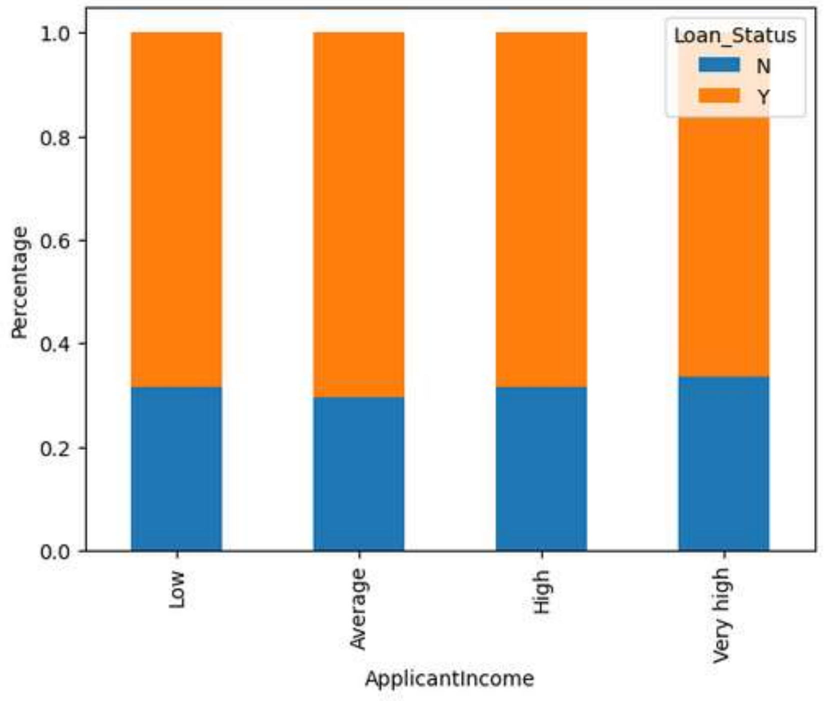
The distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status.

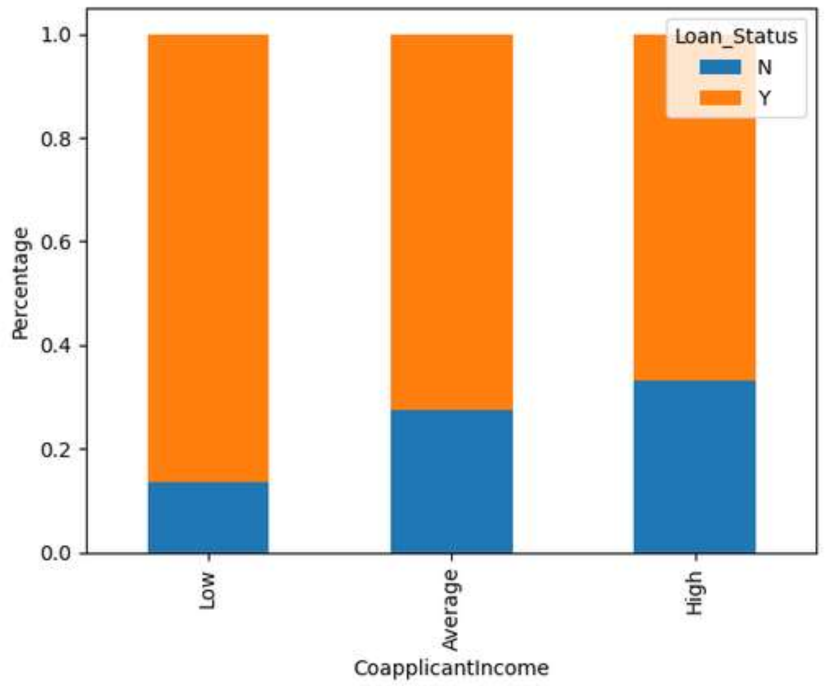
The distribution of applicants living in urban and rural area is similar across both the categories of Loan_Status and a slightly less for semi-urban.

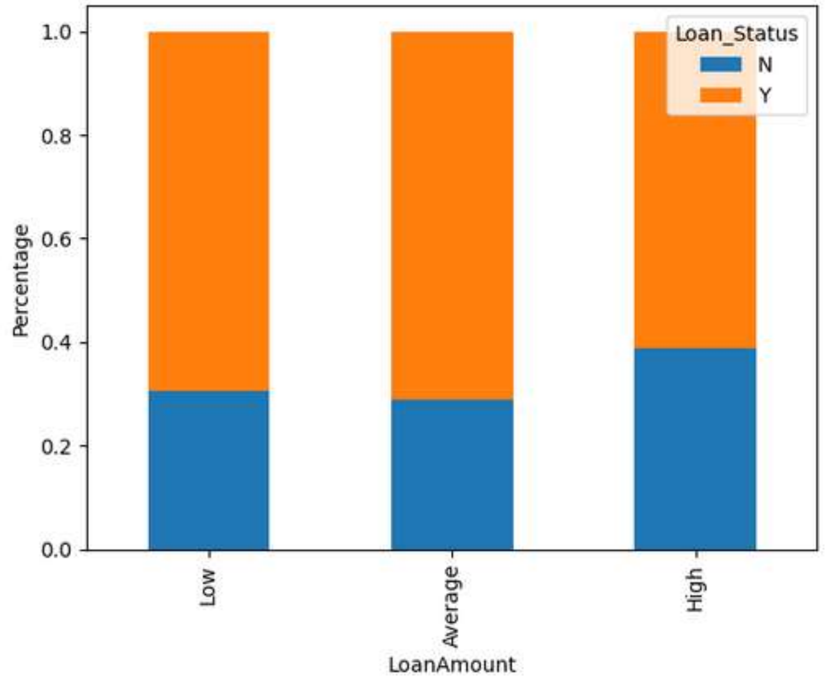
The distribution of applicants whose Loan_Status is approved is more for graduated and is less than Not graduated for not approved.

```
#Numerical Independent Variable vs Target Variable
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('ApplicantIncome')
plt.ylabel('Percentage')
plt.show()
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')
plt.show()
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')
plt.show()
```

Numerical Independent Variable vs Target Variable



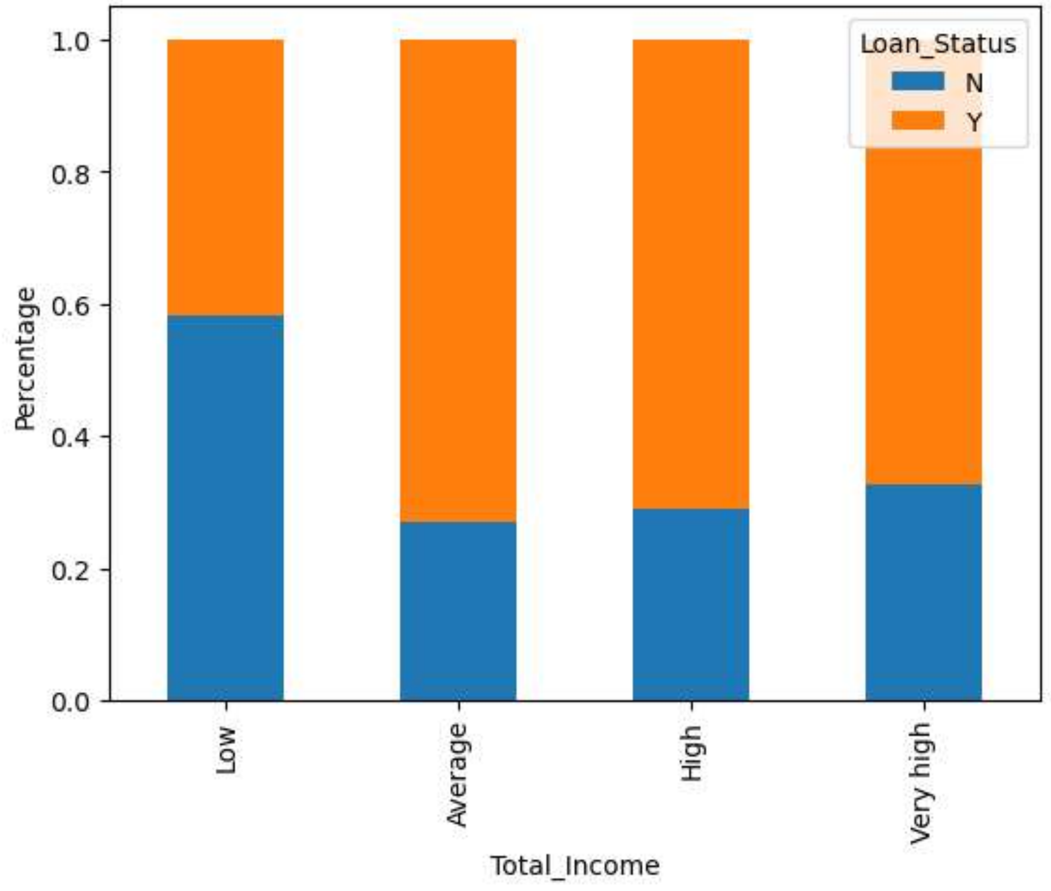




It shows that if co-applicants income is less the chances of loan approval are high.

The proportion of approved loans is higher for Low and Average Loan Amounts as compared to that of High Loan Amounts.

```python
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')
plt.show()
```

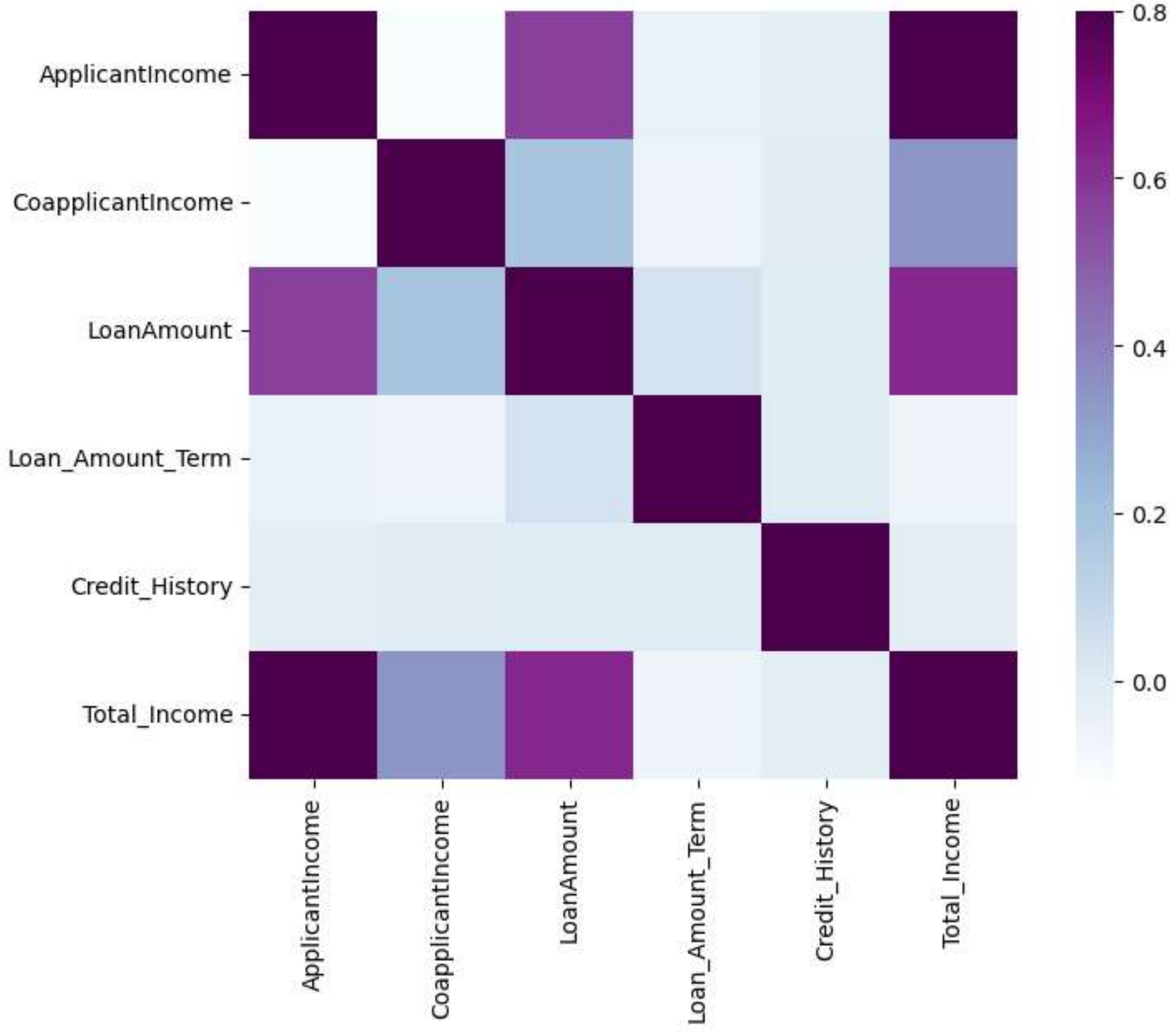


Feature engineering

It shows that if co-applicants income is less the chances of loan approval are high. But this does not look right. The possible reason behind this may be that most of the applicants don't have any co-applicant so the co-applicant income for such applicants is 0 and hence the loan approval is not dependent on it. So we can make a new variable in which we will combine the applicant's and co applicants' income to visualize the combined effect of income on loan approval.

Proportion of loans getting approved for applicants having low Total_Income is very less as compared to that of applicants with Average, High, and Very High Income.

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



Correlation between all the numerical variables using the heat map.
The variables with darker colors mean their correlation is more.

Missing Value and Outlier Treatment

```
#Missing Value Treatment
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
Income_bin	0
Coapplicant_Income_bin	273
LoanAmount_bin	22
Total_Income	0
Total_Income_bin	0

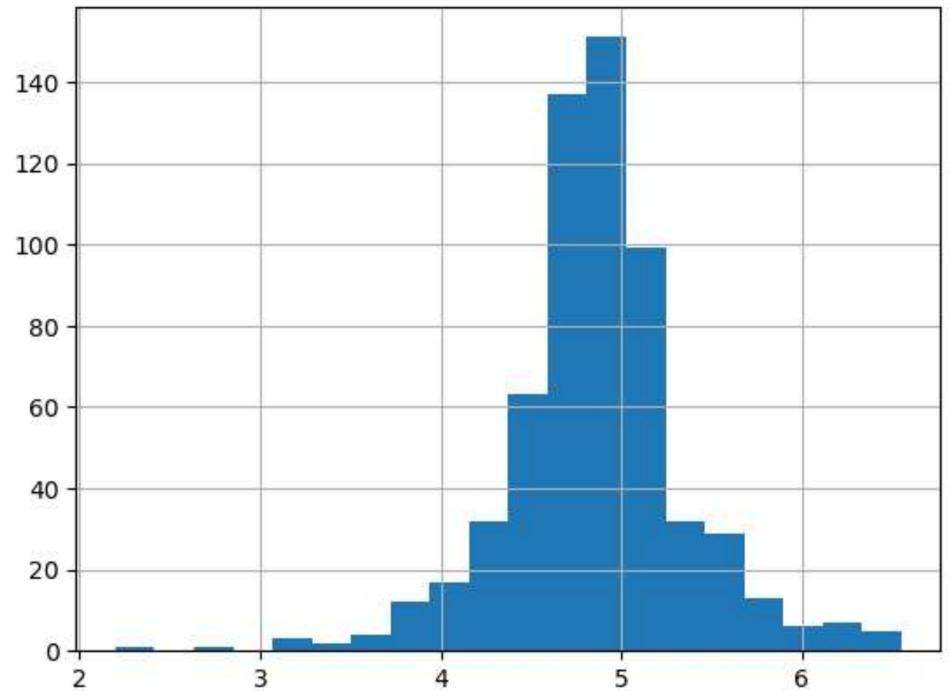
dtype: int64

```
#For categorical variables: imputation using mode
#For numerical variables: imputation using mean or median
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'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
```

- For numerical variables: imputation using mean or median.
- For categorical variables: imputation using mode

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

<Axes: >



Doing the log transformation To remove the right skewness. As we take the log transformation, it does not affect the smaller values much but reduces the larger values. So, we get a distribution similar to the normal distribution.

Model Building

Logistic Regression

```
#Logistic Regression
```

```
train=train.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)
x_train, x_cv, y_train, y_cv = train_test_split(X,y, test_size =0.3)
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
from sklearn.metrics import confusion_matrix, classification_report
model = LogisticRegression()
model.fit(x_train, y_train)
```

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Confusion Matrix
cm = confusion_matrix(y_cv, pred_cv)
print("Confusion Matrix:")
print(cm)
# Classification Report
print("\nClassification Report:")
print(classification_report(y_cv, pred_cv))
```

```
Confusion Matrix:
[[ 30  23]
 [  6 126]]

Classification Report:
              precision    recall  f1-score   support

           N       0.83      0.57      0.67        53
           Y       0.85      0.95      0.90       132

    accuracy                           0.84       185
   macro avg       0.84      0.76      0.79       185
weighted avg       0.84      0.84      0.83       185
```

```
from sklearn.metrics import confusion_matrix, classification_report
# Confusion Matrix
cm = confusion_matrix(y_cv, pred_cv)
print("Confusion Matrix:")
print(cm)
# Parse Classification Report
report = classification_report(y_cv, pred_cv, output_dict=True)
# Extracting Metrics
accuracy = report['accuracy']
f1_score = report['weighted avg']['f1-score']
precision = report['weighted avg']['precision']
recall = report['weighted avg']['recall']
# Printing Metrics
print("\nAccuracy:", accuracy)
print("F1 Score:", f1_score)
print("Recall:", recall)
print("Precision:", precision)
```

```
Confusion Matrix:
[[ 30  23]
 [  6 126]]

Accuracy: 0.8432432432432433
F1 Score: 0.8330138447456215
Recall: 0.8432432432432433
Precision: 0.8421125823810388
```

Result

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
# Create and train the Random Forest classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train, y_train)
# Predict on the validation set
pred_cv_rf = rf_classifier.predict(x_cv)
# Confusion Matrix
cm_rf = confusion_matrix(y_cv, pred_cv_rf)
print("Confusion Matrix (Random Forest):")
print(cm_rf)
# Classification Report
print("\nClassification Report (Random Forest):")
print(classification_report(y_cv, pred_cv_rf))
```

```
Confusion Matrix (Random Forest):
[[ 30  23]
 [ 15 117]]

Classification Report (Random Forest):
              precision    recall  f1-score   support

           N       0.67      0.57      0.61        53
           Y       0.84      0.89      0.86       132

    accuracy                           0.79       185
   macro avg       0.75      0.73      0.74       185
weighted avg       0.79      0.79      0.79       185
```

```
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score
# Confusion Matrix
print("Confusion Matrix (Random Forest):")
print(cm_rf)
# Accuracy
accuracy_rf = accuracy_score(y_cv, pred_cv_rf)
# F1 Score
f1_score_rf = f1_score(y_cv, pred_cv_rf, average='weighted')
# Recall
recall_rf = recall_score(y_cv, pred_cv_rf, average='weighted')
# Precision
precision_rf = precision_score(y_cv, pred_cv_rf, average='weighted')
# Printing Metrics
print("\nAccuracy (Random Forest):", accuracy_rf)
print("F1 Score (Random Forest):", f1_score_rf)
print("Recall (Random Forest):", recall_rf)
print("Precision (Random Forest):", precision_rf)
```

```
Confusion Matrix (Random Forest):
[[ 30  23]
 [ 15 117]]

Accuracy (Random Forest): 0.7945945945945946
F1 Score (Random Forest): 0.7892313682229648
Recall (Random Forest): 0.7945945945945946
Precision (Random Forest): 0.7872844272844274
```

Result

Conclusion

In conclusion, the loan prediction project showcased the power of data science in extracting actionable insights from complex datasets. By leveraging techniques like exploratory data analysis, feature engineering, and model evaluation, data scientists can uncover patterns, make accurate predictions, and drive informed decision-making. This project underscores the role of data science in tackling real-world challenges and highlights its potential to revolutionize industries through data-driven solutions.

Thank you!!