A task on

PREDICTING LOAN APPROVALS

By

Pratik Lohar

Abstract

In the financial industry, loan eligibility is determined by a set of criteria that assess an individual's capability to repay the loan. This assessment typically involves collecting and analyzing various factors such as loan amount, educational status, marital status, gender, credit history, and assets. However, this traditional approach to evaluating loan eligibility is time-consuming and resource-intensive. To streamline this process, there is a growing interest in leveraging machine learning algorithms to develop predictive models that can automatically assess loan eligibility based on applicant information.

This study explores the application of machine learning techniques to predict loan approvals using a comprehensive dataset containing relevant features collected from loan applications. Initially, exploratory data analysis is conducted to understand the characteristics of the dataset and identify patterns or correlations among the features. Subsequently, supervised learning algorithms including logistic regression, decision trees, random forests, and gradient boosting models are employed to build predictive models for loan approval.

The performance of these models is evaluated using various metrics such as accuracy, precision, recall, and F1-score to assess their effectiveness in classifying loan applications as approved or denied. Additionally, feature importance analysis is conducted to identify the most influential factors contributing to loan approval decisions. This analysis provides valuable insights into the factors considered by lending institutions when assessing the creditworthiness of loan applicants.

The results demonstrate the efficacy of machine learning algorithms in accurately predicting loan approvals, with certain models outperforming others based on specific evaluation metrics. By leveraging advanced analytical techniques, financial institutions can enhance the efficiency and accuracy of loan approval processes, thereby improving customer experience and reducing operational costs. This study underscores the potential benefits of incorporating machine learning models into existing loan approval systems to automate decision-making processes and expedite loan processing while ensuring prudent risk management practices.

Aims and Objectives

1. Automated Loan Eligibility Prediction:

Aim: Develop a predictive model using machine learning algorithms to automate the assessment of loan eligibility for retail banks.

Objectives:

Implement various machine learning algorithms, including Logistic Regression, SVM, Decision Trees, and Random Forest, to predict whether a loan applicant is eligible for approval.

Train and evaluate each model to determine its accuracy and effectiveness in predicting loan eligibility.

2. Exploratory Data Analysis (EDA) for Insights:

Aim: Gain insights into the dataset and understand feature relationships through exploratory data analysis (EDA).

Objectives:

Analyze the distribution of key features such as applicant income, coapplicant income, loan amount, gender, marital status, education, self-employment status, credit history, and property area.

Investigate relationships between categorical variables (e.g., gender, education, property area) and loan approval status to identify influential factors.

3. Data Preprocessing and Feature Engineering:

Aim: Prepare the dataset for model training by handling missing values and optimizing feature representation.

Objectives:

Handle missing values in the dataset using strategies like imputation and median replacement for continuous features (e.g., loan amount).

Perform feature engineering techniques such as label encoding for categorical variables (e.g., gender, education, property area) to convert them into numeric format suitable for machine learning models.

4. Model Building and Evaluation:

Aim: Build machine learning models to predict loan eligibility and evaluate their performance.

Objectives:

Train machine learning models, including Logistic Regression, SVM, Decision Trees, and Random Forest, on the preprocessed dataset to predict loan eligibility based on applicant information.

Evaluate the performance of each model using metrics like accuracy, precision, recall, and confusion matrices to determine the most effective model for loan eligibility prediction.

5. Comparison of Model Performance:

Aim: Compare the performance of different machine learning models in predicting loan eligibility.

Objectives:

Evaluate and compare the accuracy scores of Logistic Regression, SVM, Decision Trees, and Random Forest models on both training and testing datasets.

Analyze the confusion matrices to understand the classification performance of each model and identify any potential trade-offs between accuracy and other metrics.

6. Selection of Optimal Model:

Aim: Select the most accurate and efficient machine learning model for loan eligibility prediction.

Objectives:

Compare the performance metrics (accuracy scores) of all models to identify the most optimal model for predicting loan eligibility.

Consider additional factors such as computational complexity and interpretability when selecting the final model for deployment in real-world scenarios.

7. Deployment and Application:

Aim: Deploy the selected machine learning model for automated loan eligibility prediction in retail banks.

Objectives:

Deploy the selected model, which demonstrates the highest accuracy and efficiency, into the retail bank's loan approval system for automated loan eligibility prediction.

Monitor the model's performance in real-world scenarios and iterate as necessary to ensure continuous improvement and accuracy in loan eligibility prediction.

8. Overall Project Evaluation:

Aim: Evaluate the overall effectiveness and impact of the project in automating loan eligibility prediction for retail banks.

Objectives:

Assess the project's success based on the accuracy and efficiency of the deployed machine learning model in predicting loan eligibility.

Gather feedback from stakeholders, including retail banks and loan applicants, to measure the project's impact on streamlining loan approval processes and improving customer satisfaction.

These aims and objectives outline the key steps and goals of the project, from data analysis and preprocessing to model building, evaluation, deployment, and overall project evaluation.

Contents

Chapter 1

Introduction

Loans represent the cornerstone of banking operations, constituting a significant portion of a bank's revenue through interest payments. Despite thorough verification and validation processes, banks often lack complete assurance regarding an applicant's ability to repay a loan without encountering difficulties. This uncertainty underscores the importance of loan prediction, a critical real-life problem that retail banks face at least once in their lifetime. If executed correctly, loan prediction has the potential to streamline operations and save considerable manpower for retail banks.

The primary objective of this project is to design a predictive model capable of accurately determining loan eligibility, thereby reducing the time-consuming verification process. This entails leveraging machine learning algorithms to assess an applicant's creditworthiness and likelihood of repayment based on various factors such as credit history, income level, employment status, and assets. By accurately predicting loan eligibility, banks can mitigate risks associated with defaults and make informed lending decisions, ultimately improving customer satisfaction and financial performance.

To achieve this goal, we begin by establishing a foundational understanding of key machine learning concepts. Subsequently, we delve into the exploration of machine learning algorithms such as Logistic Regression, Decision Trees, and Random Forests. Through a comparative analysis of these algorithms, we aim to identify the most suitable model for predicting loan eligibility based on performance metrics such as accuracy, precision, recall, and F1-score.

By leveraging machine learning algorithms to predict loan eligibility, this project seeks to optimize the loan approval process and enhance operational efficiency for retail banks. The resulting predictive model has the potential to revolutionize the way banks assess loan applicants, offering a data-driven approach to lending decisions that can improve overall financial outcomes while simultaneously providing a seamless experience for customers.

Chapter 2

Preliminaries

This chapter contains the basic definitions and results which we referred from other research papers or books.

2.1 Basic Definitions and Results

Definition 2.1.1 (Machine Learning (ML))

Machine Learning is a method of data analysis based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Definition 2.1.2 (Algorithm)

It is the set of rules used to learn patterns from the data. OR A set of rules or techniques used to make predictions from the given data.

Definition 2.1.3 (Model)

A component which is trained by Machine Learning algorithm and used to recognize the patterns in the data.

Definition 2.1.4 (Predictor Variable (X))

It is an independent variable of the data that can be used to predict the output.

Definition 2.1.5 (Target Variable)

It is the output variable that can be predicted using response variables.

Definition 2.1.6 (Training data)

A set of data used to build the model.

Definition 2.1.7 (Testing data)

A set of data used to evaluate the performance of the model.

2.2 Machine Learning Process

Machine learning process includes designing a predictive model that helps a system to solve the problem with minimal human intervention. The process follows 7 steps as

Problem Statement

In this step we define and understand the problem for which we have to find the solution.

Data Collection

It involves collection of reliable data so that our Machine Learning model can find the correct patterns.

Preparing Data

This step involves dealing with cleaning the raw data. Data cleaning deals with the wrong formats, missing values, outliers, erroneous and inconsistent values, etc.

Data Exploration

Visualization of the data and exploration of categorical and continuous variables to detect the relationships between the class variables.

Building a Model

Developing a model using suitable algorithm.

Model Evaluation

Testing the performance of the model on previously unseen data.

Predictions

Use the model on unseen data to make predictions accurately.

2.3 Libraries Used

1. Pandas

Pandas is a python library used for working with data sets. It has functions for analyzing, cleaning, exploring and manipulating data. The name 'Pandas' has a reference to both 'Panel Data' and 'Python Data Analysis'. Pandas allows us to analyse big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science. Powerful, fast and flexible library for Data analysis, manipulation and filtering.

2. NumPy

NumPy stands for Numerical Python and is a Python library used for working with arrays. It has functions for working in domain of linear algebra, fourier transform and matrices. It aims to provide an array object that is up to 50 times faster than traditional Python lists. It is popular machine learning library that supports large matrices and multidimensional data. It consists of in-built mathematical functions for easy computations.

3. Matplotlib

It is a plotting library in python. It is amazing visualization library used to create interactive graphs, charts and maps. It has a module named pyplot which makes things easy for plotting and supports wide variety of graphs and plots namely histogram, bar charts, etc.

4. Scikit-learn

Scikit-learn is a famous Python library to work with complex data. It is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with NumPy and SciPy.

Chapter 3

Dataset Information

3.1 About Prediction Loan Approval

First understand what the Prediction Loan Approval is, Loan's are one of the core business of banks as a lot of profit gain through interest. Usually loan eligibility is decided after long and intensive process of document verification and validation of set of criteria which takes huge amount of time. This time consuming process can be avoided by developing a system that can predict whether a customer is eligible for the loan based on the information given by the customer. We will use concepts of Data science and Machine Learning algorithm to develop a model which will predict the eligibility.

3.2 Dataset

A classification problem where we have to predict whether a loan would be approved or not. Below is the dataset attributes with 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 a loan in months
- Credit_History ----------→ Credit history meets guidelines
- Property_Area ----------→ Urban/ Semi-Urban/ Rural
- Loan_Status ----------→ Loan approved (Y/N)

Chapter 4

Python Code

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
[2] df = pd.read_csv("/content/drive/MyDrive/filehandling/train.csv")
    df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0

In the script above we use the read_csv()method of the Pandas library to read the "train.csv" file. Next we call the head() method from the dataframe object returned by the read_csv() function, which will display the first five rows of the dataset.

```
df.shape
```

```
(614, 13)
```

We have 614 rows and 13 columns in the train dataset.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

We called info() method for checking any null entry in the dataset. As you can see some values are missing in few of the variables that present to be here. You can also see data types of each of the variables mentioned over here.

```
[5] df.columns

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

We have 12 independent variables and 1 target variable i.e. Loan_Status in the train dataset.

```
df.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
```

We can see there are three formats of data types:

object: Object format means variables are categorical. Categorical variables in our dataset are: Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Property_Area, Loan_Status.

int64: It represents the integer variables. ApplicantIncome is of this format.

float64: It represents the variable which have some decimal values involed. They are also numerical.

Data Exploration

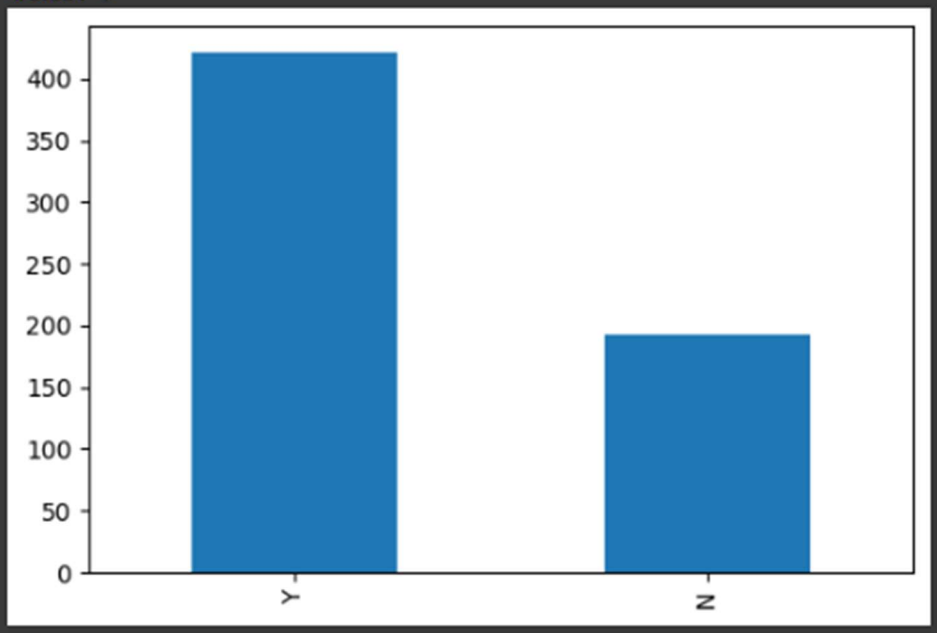
```
[7] df['Loan_Status'].value_counts()

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

Normalize can be set to True to print proportions instead of number.

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

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

```
[9] df['Loan_Status'].value_counts().plot.bar(figsize=(6,4))
```

```
<Axes: >
```



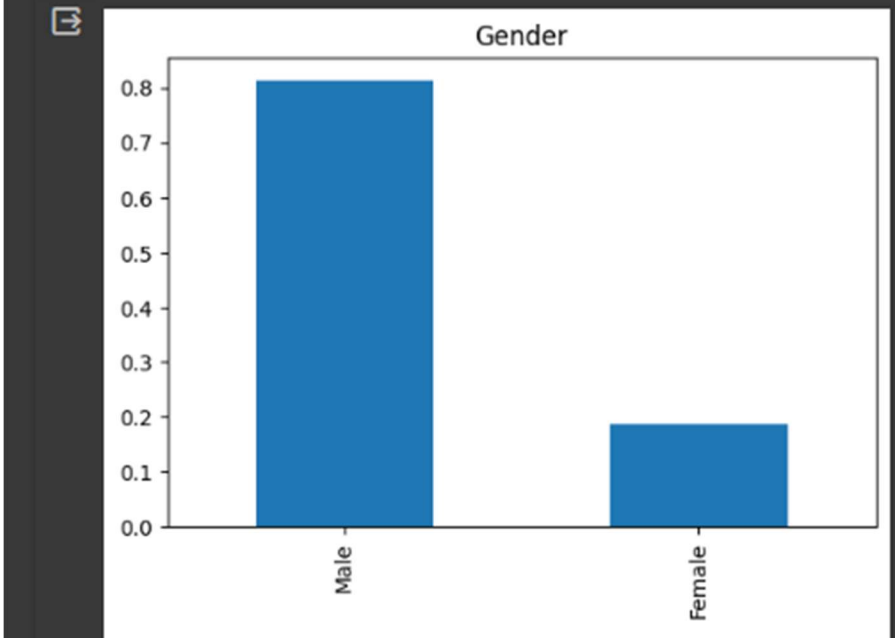
The loan of 422(around 69%) people out of 614 approved.

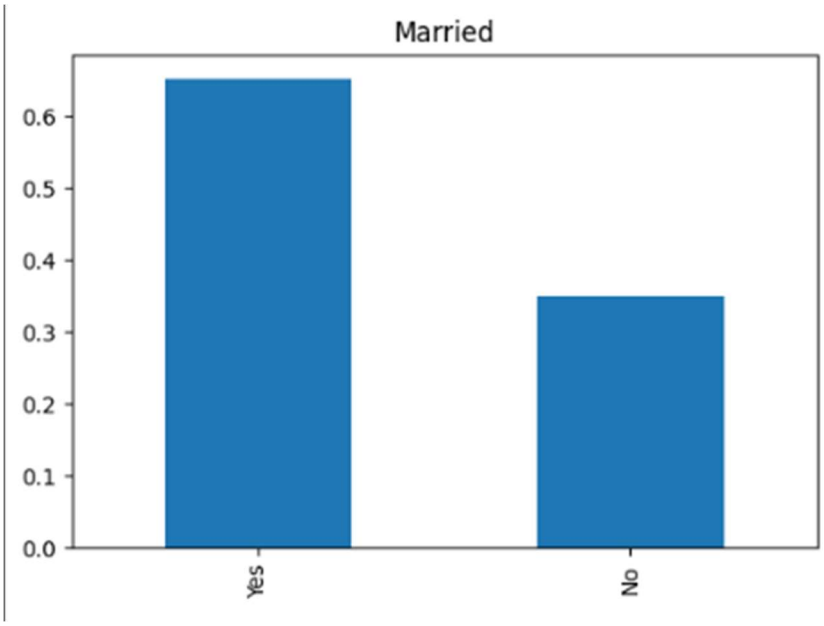
Now, let's visualize each variable separately. Different types of variables are categorical, ordinal and numerical.

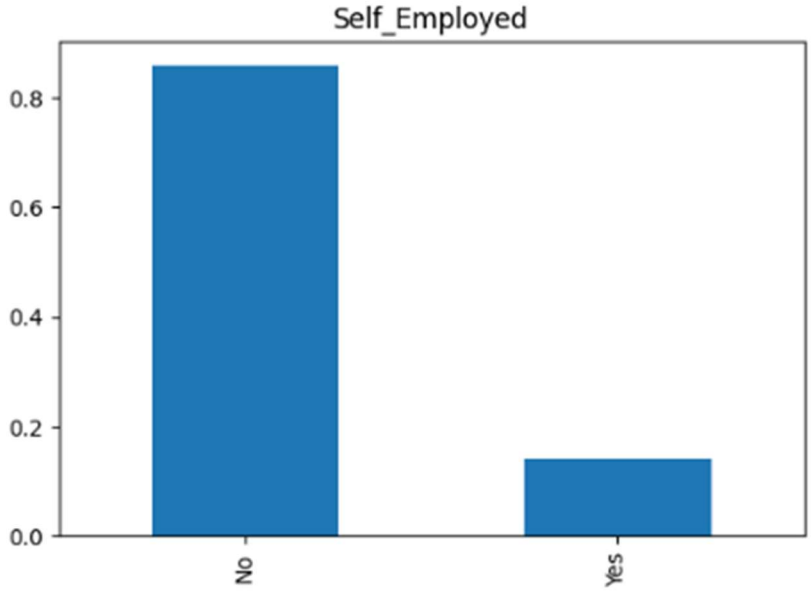
Categorical features: These features have categories (Gender, Married, Self_Employed, Credit_History, Loan_Staus) Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property_Area) Numerical features: These features have numerical values (ApplicantIncome, CoapplicatIncome, LoanAmount, Loan_Amount_Term)

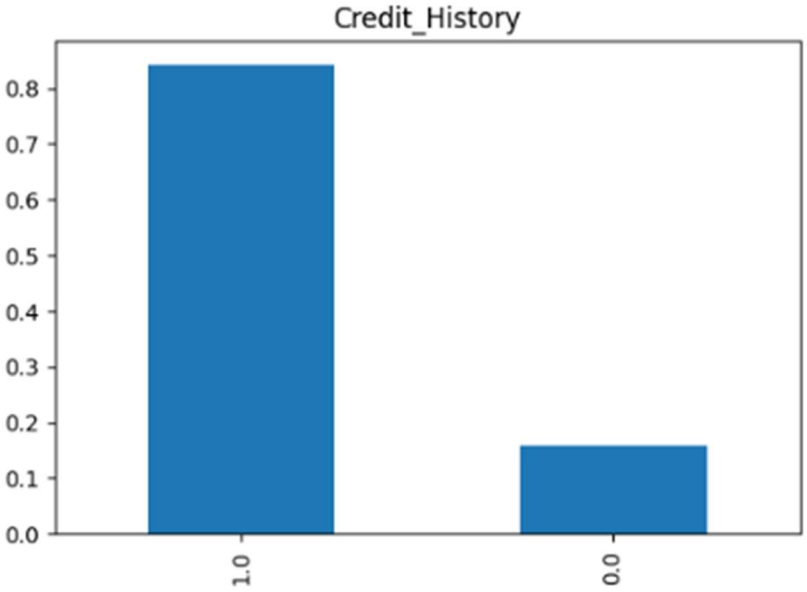
Independent Variable (Categorical)

```
df['Gender'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Gender")
plt.show()
df['Married'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Married")
plt.show()
df['Self_Employed'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Self_Employed")
plt.show()
df['Credit_History'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Credit_History")
plt.show()
```









It can be inferred from the above bar plots that:

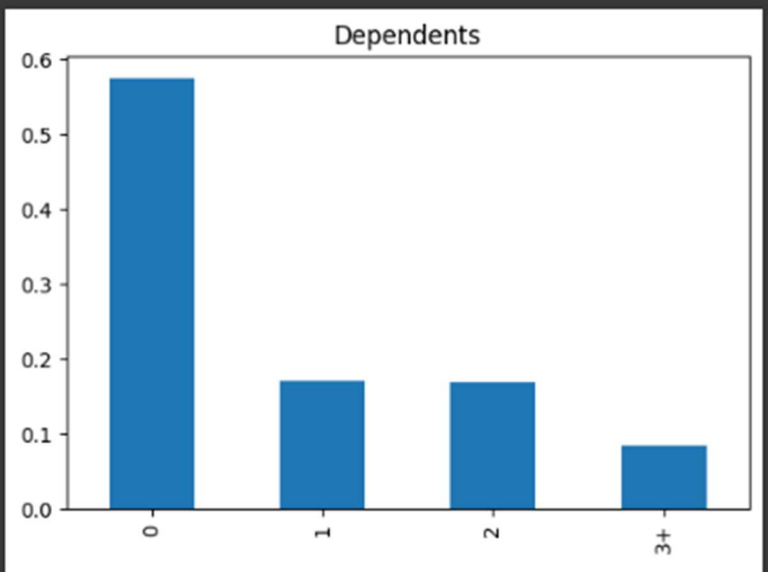
80% applicants in the dataset are male.

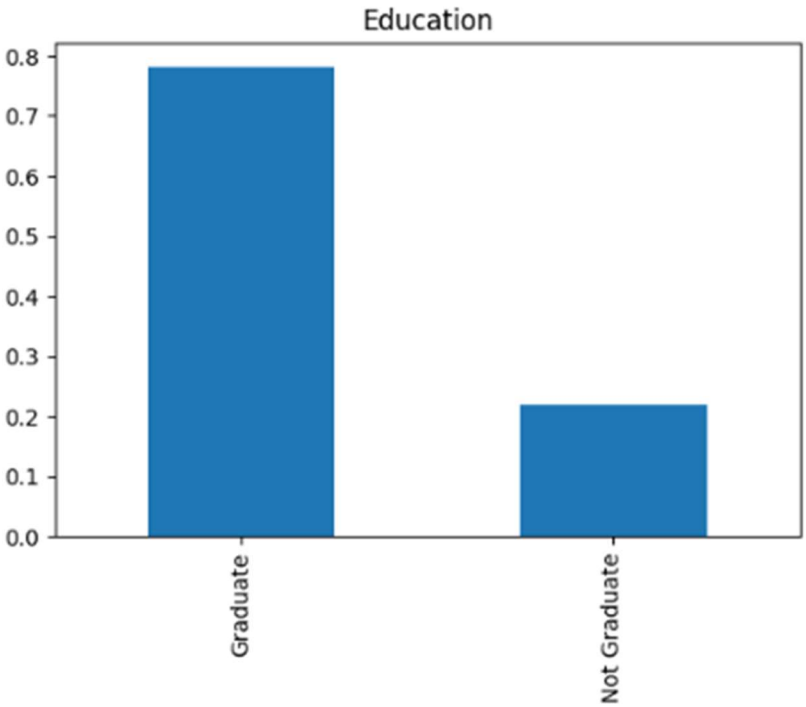
Around 65% of the applicants in the dataset are married.

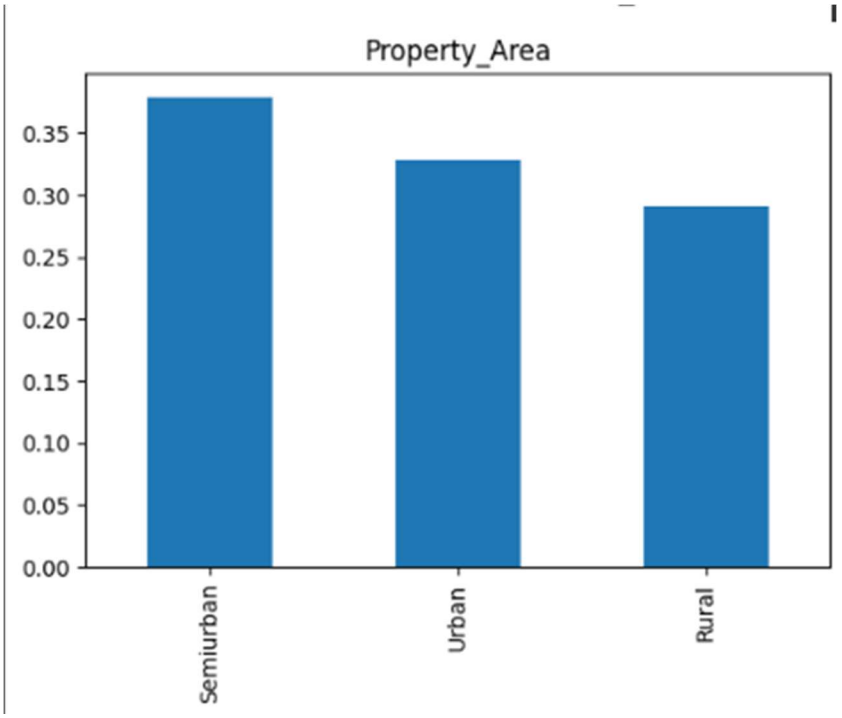
Around 15% applicants in the dataset are self employed. Around 85% applicants have repaid their doubts.

Independent Variable (Ordinal)

```
df['Dependents'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Dependents")
plt.show()
df['Education'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Education")
plt.show()
df['Property_Area'].value_counts(normalize=True).plot.bar(figsize=(6,4),title="Property_Area")
plt.show()
```







Following inferences can be made frm the above bar plots:

Most of the applicants don't have any dependents.

Around 80% of the applicants are Graduate.

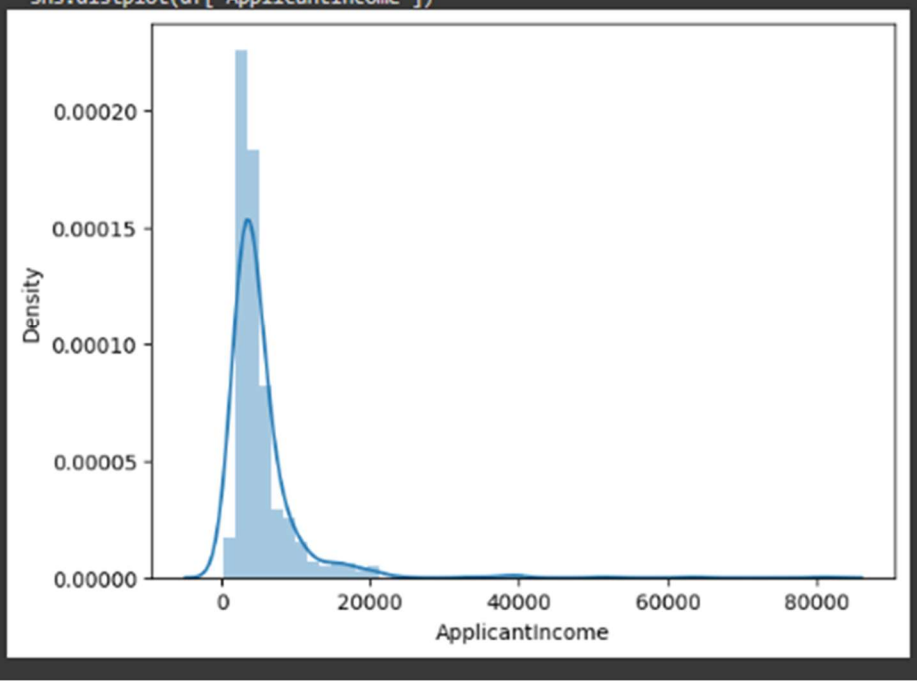
Most of the applicants are from SemiUrban Area.

Independent Variable (Numerical)

Till now we have seen the categorical and ordinal variables and now lets visualize the numerical variables. Lets look at the distributaion of Applicamt Income first.

```
[69] sns.distplot(df['ApplicantIncome'])
     plt.show()
```

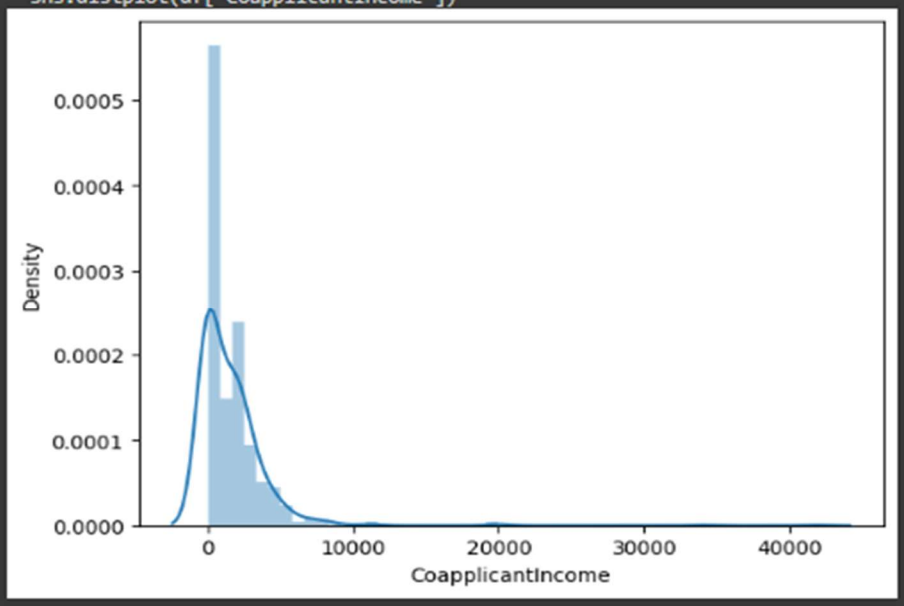
```
<ipython-input-69-103a24546b03>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['ApplicantIncome'])
```



It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed. We will try to make it normal in later sections as algorithms works better if the data is normally distributed.

```
sns.distplot(df['CoapplicantIncome'])
plt.show()
```

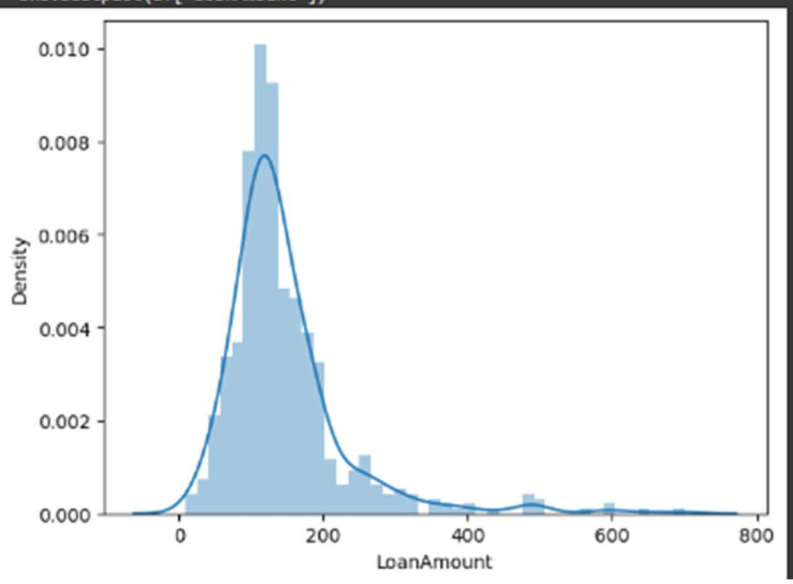
```
<ipython-input-70-1a98ee5a07a6>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['CoapplicantIncome'])
```



We see a similar distribution as that of the applicant's income. The majority of co-applicants income ranges from 0 to 5000.

```
sns.distplot(df['LoanAmount'])
plt.show()
df['LoanAmount'].plot.box(figsize=(7,4))
plt.show()
```

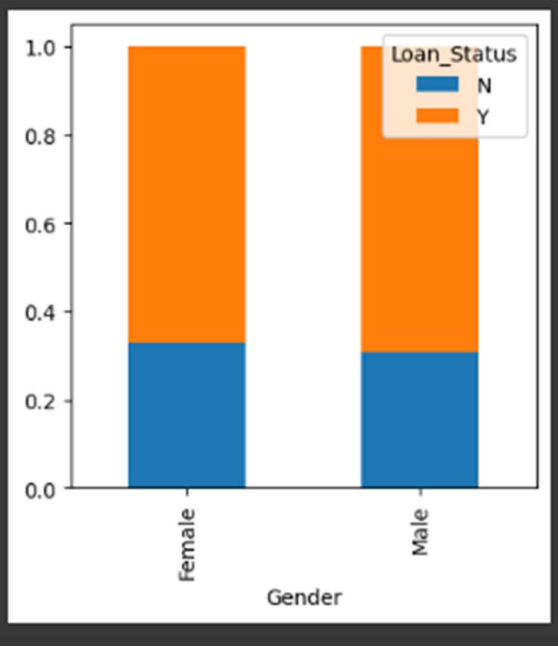
```
<ipython-input-15-116e09a4b146>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

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



Categorical Independent Variable vs Target Variable

First of all, we will find the relation between the 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.

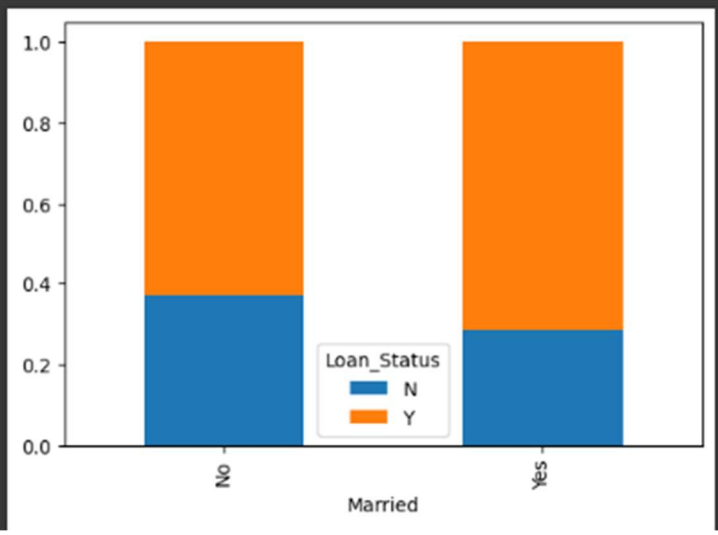
```
Gender= pd.crosstab(df['Gender'], df['Loan_Status'])
Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(4,4))
plt.show()
```

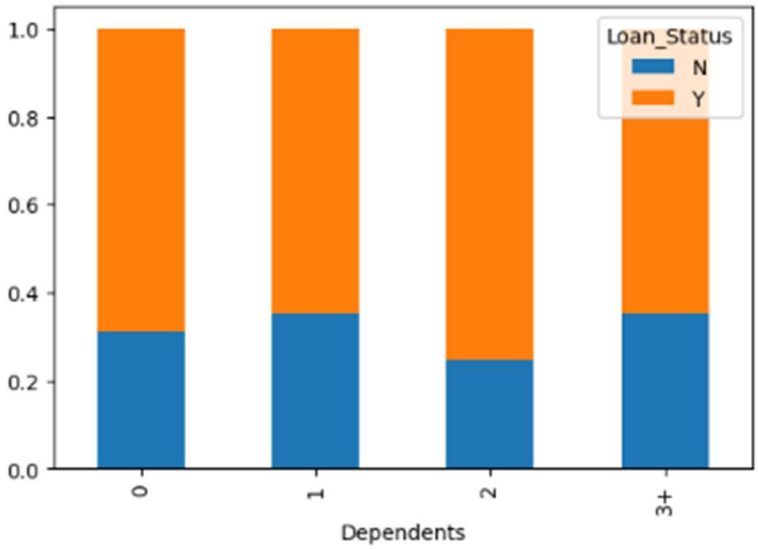


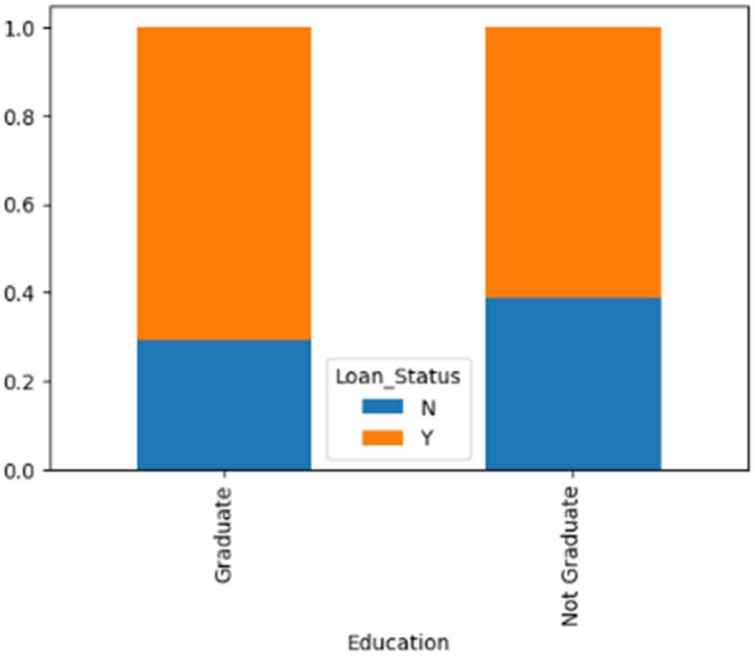
It can be inferred that the proportion of male and female applicants is more or less the same for both approved and unapproved loans.

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

```
Married= pd.crosstab(df['Married'], df['Loan_Status'])
Married.div(Married.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
Dependents= pd.crosstab(df['Dependents'], df['Loan_Status'])
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
Education= pd.crosstab(df['Education'], df['Loan_Status'])
Education.div(Education.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
Self_Employed= pd.crosstab(df['Self_Employed'], df['Loan_Status'])
Self_Employed.div(Self_Employed.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
```



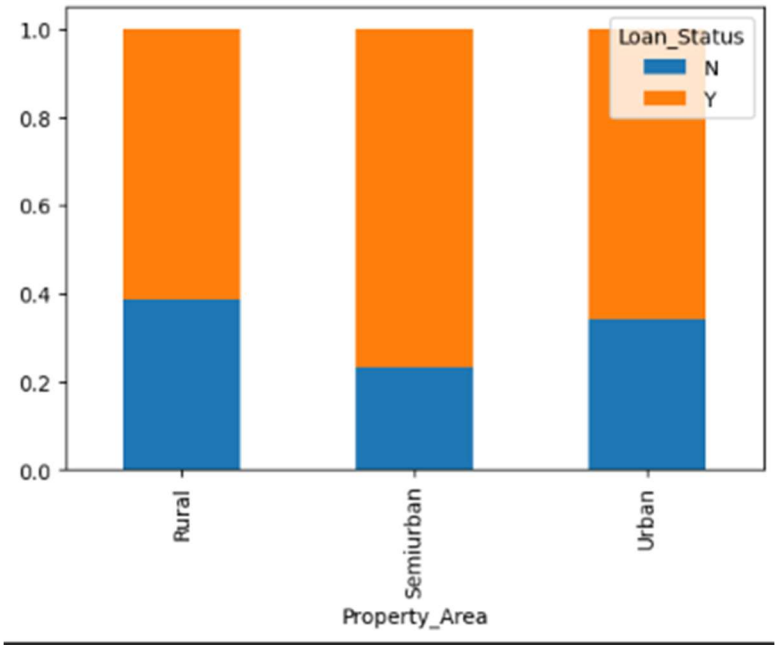




The proportion of married applicants is higher for approved loans. Distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status. There is nothing significant we can infer from Self_Employed vs Loan_Status plot.

Now we will look at the relationship between remaining categorical independent variables and Loan_Status.

```
Credit_History= pd.crosstab(df['Credit_History'], df['Loan_Status'])
Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
Property_Area= pd.crosstab(df['Property_Area'], df['Loan_Status'])
Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind='bar',stacked=True, figsize=(6,4))
plt.show()
```

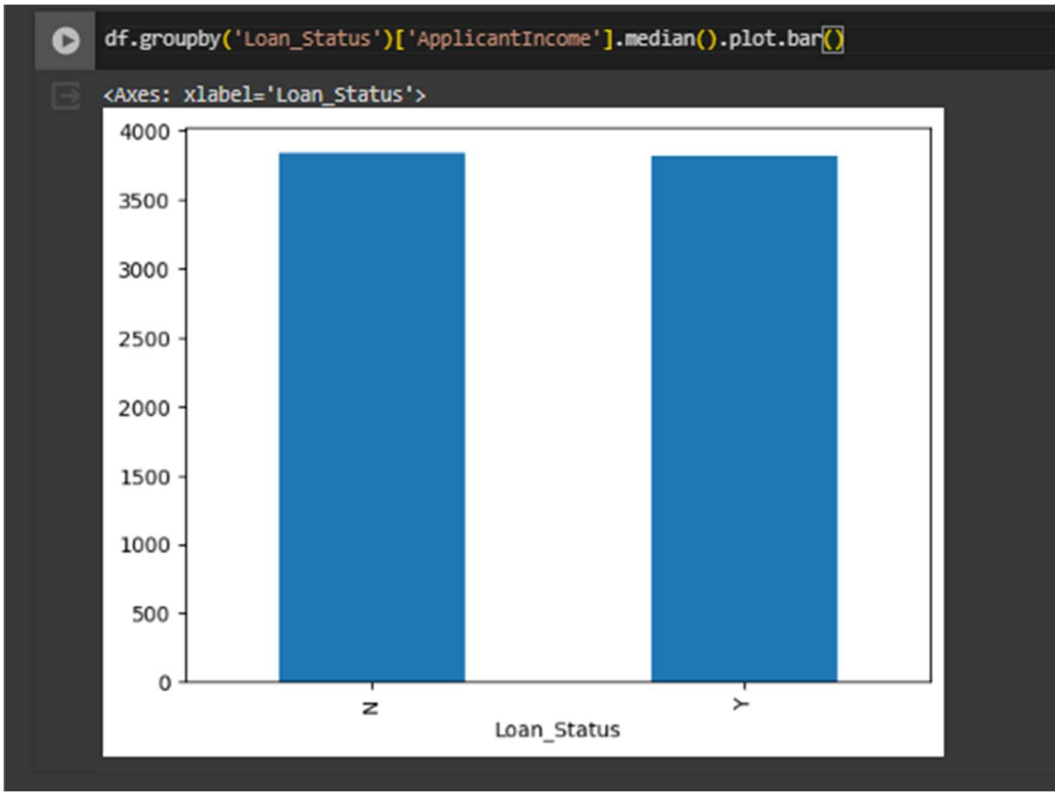


It seems people with credit history as 1 are more likely to get their loans approved. The proportion of loans getting approved in the semi-urban area is higher as compared to that in rural or urban areas.

Now let's visualize numerical independent variables with respect to the target variable.

Numerical Independent Variable vs Target Variable

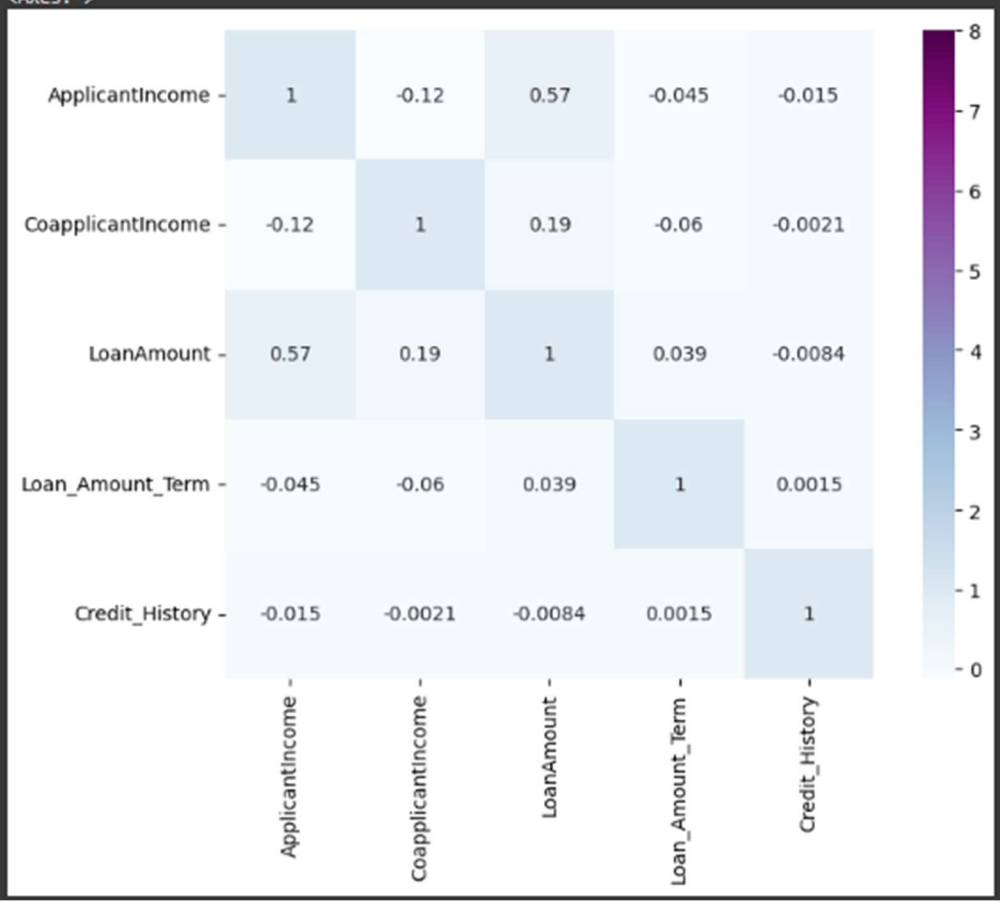
We will try to find the mean or median income of people for which the loan has been approved vs the mean or median income of people for which the loan has not been approved.



Here the y-axis represents the mean applicant income. We don't see any change in the mean income. So let's make bins for the applicant income variable based on the values in it and analyze the corresponding loan status for each bin.

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

```
<ipython-input-20-eee6e80b79bf>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is de
  matrix = df.corr()
<Axes: >
```



We see that the most correlated variables are (ApplicantIncome - LoanAmount) and (Credit_History - Loan_Status). LoanAmount is also correlated with CoapplicantIncome.

Missing value imputation

```
df.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
```

There are missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History features.

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

We can consider these methods to fill the missing values.

For numerical variables imputation using mean or median.

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.

```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

Now 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.

```
df["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
```

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.

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

Now we will see the LoanAmount variable. As it is a numerical variable , we can use mean or median to impute the missing values. We will use the median to fill the null values as earlier we saw that the loan amount has outliers so the mean will not be the proper approach as it is highly affected by the presence of outliers.

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

Now lets check whether all the missing values are filled in the dataset.

```
[26] df.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
```

All the null values replace. So we don't need to worry about null values any more this is basic preprocessing. We have the null values using mean, median or mode operation.

```
print(df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             614 non-null    object
 2   Married            614 non-null    object
 3   Dependents         614 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      614 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    float64
 9   Loan_Amount_Term   614 non-null    float64
 10  Credit_History     614 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
None
```

Categorical Converted into Numerical Attribute

```
[23] # Label Encoding
from sklearn.preprocessing import LabelEncoder
cols = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area', 'Loan_Status']
le = LabelEncoder()
for col in cols:
  df[col]=le.fit_transform(df[col])

[ ] df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	1	0	0	0	0	5849	0.0	128.0	360.0	1.0	2	1
1	LP001003	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	0	0
2	LP001005	1	1	0	0	1	3000	0.0	66.0	360.0	1.0	2	1
3	LP001006	1	1	0	1	0	2583	2358.0	120.0	360.0	1.0	2	1
4	LP001008	1	0	0	0	0	6000	0.0	141.0	360.0	1.0	2	1

Now you can see the all attribute in the data set in the numerical format so we can train our model easily. Here only we need to noticed about Loan Status for Yes(1) and no(0)

Let's drop the Loan_ID variable as it do not have any effect on the loan status. We will do the same changes to the test dataset which we did for the training dataset.

```
[30] cols = ['Loan_ID']
     df = df.drop(columns = cols, axis=1)

     # Replace '3+' with '3' in the 'Dependents' column
     df['Dependents'] = df['Dependents'].replace('3+', '3')

     # Convert the 'Dependents' column to numeric dtype
     df['Dependents'] = pd.to_numeric(df['Dependents'])
```

Train - Test Split

```
[32] # specify input and output attributes
     x = df.drop(columns =['Loan_Status'], axis=1)
     y = df['Loan_Status']

     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.25,random_state=15)
```

Split the ratio of 75-25. 75 percente affect to training and 25 percente affect on testing. We are going to set therandom state=15 because we don't know want to a result change the every cycle. It will give a same accuracy or the same result every time the you run the code again.

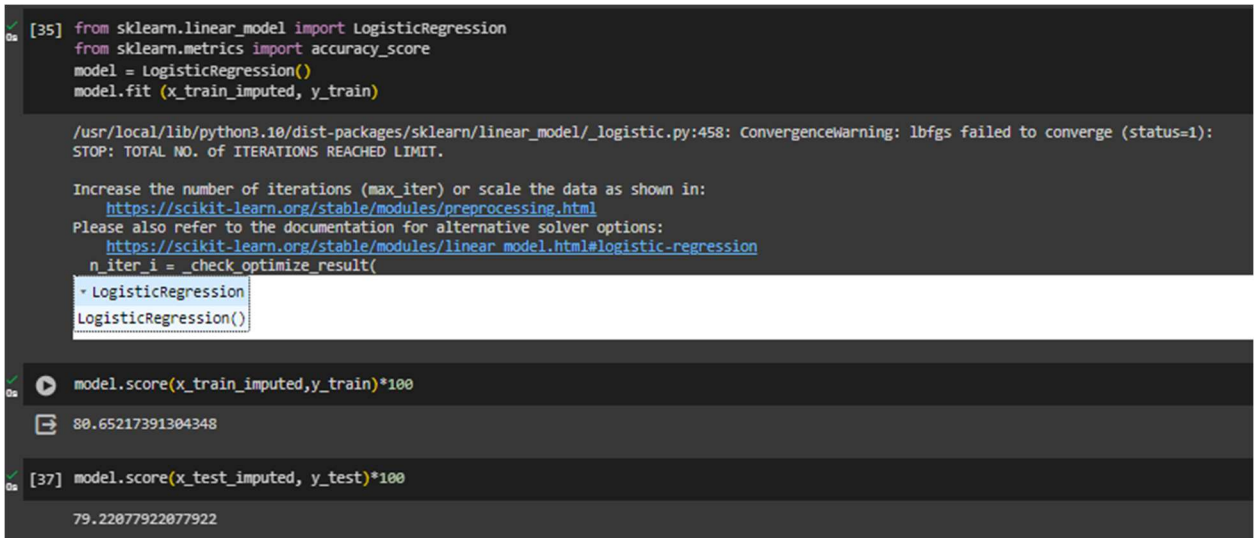
```
from sklearn.impute import SimpleImputer

# Initialize SimpleImputer
imputer = SimpleImputer(strategy='mean')

# Fit imputer on x_train and transform x_train and x_test
x_train_imputed = imputer.fit_transform(x_train)
x_test_imputed = imputer.transform(x_test)
```

Model Building

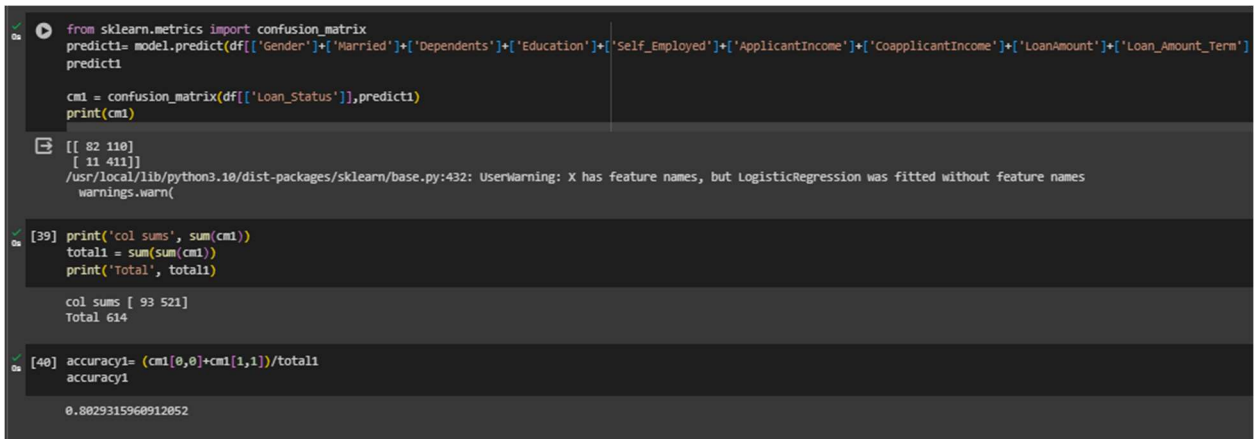
Let us make our first model predict the target variable. We will start with Logistic Regression which is used for predicting binary outcome. Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. 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.

Logistics Regression Logistic regression analysis studies the association between a categorical dependent variable and a set of independent

(explanatory) variables. The name logistic regression is used when the dependent variable has only two values, such as 0 and 1 or Yes and No. In logistic regression, a categorical dependent variable Y having n (usually n = 2) unique values is regressed on a set of p independent variables X1, X2, . . . , Xp. The logistic regression model is given by the G equations ln pn p1 = ln pn p1 + βn1X1 + βn2X2 + . . . + βnpXp Here, pn is the probability that an individual with values X1, X2, . . . , Xp is in outcome n. That is, pn = Pr(Y = n | X) The quantities p1, p2, . . . , pn represent the prior probabilities of outcome membership. If these prior probabilities are assumed equal, then the term ln (pn/p) becomes zero and drops out. If the priors are not assumed equal, they change the values of the intercepts in the logistic regression equation.



```
[35] from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score
     model = LogisticRegression()
     model.fit (x_train_imputed, y_train)

     /usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
     Please also refer to the documentation for alternative solver options:
         https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
       n_iter_i = _check_optimize_result(
     LogisticRegression()

     model.score(x_train_imputed,y_train)*100

     80.65217391304348

[37] model.score(x_test_imputed, y_test)*100

     79.22077922077922
```



```
from sklearn.metrics import confusion_matrix
predict1= model.predict(df[['Gender']+['Married']+['Dependents']+['Education']+['Self_Employed']+['ApplicantIncome']+['CoapplicantIncome']+['LoanAmount']+['Loan_Amount_Term']
predict1

cm1 = confusion_matrix(df[['Loan_Status']],predict1)
print(cm1)

[[ 82 110]
 [ 11 411]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(

[39] print('col sums', sum(cm1))
     total1 = sum(sum(cm1))
     print('Total', total1)

     col sums [ 93 521]
     Total 614

[40] accuracy1= (cm1[0,0]+cm1[1,1])/total1
     accuracy1

     0.8029315960912052
```

Multicollinearity

```
[41] import statsmodels.formula.api as sm

     def vif_cal(input_data, dependent_col):
       x_vars=input_data.drop([dependent_col], axis=1)
       xvar_names=x_vars.columns
       for i in range(0,xvar_names.shape[0]):
         y=x_vars[xvar_names[i]]
         x=x_vars[xvar_names.drop(xvar_names[i])]
         rsq=sm.ols(formula="y~x", data=x_vars).fit().rsquared
         vif=round(1/(1-rsq),2)
         print(xvar_names[i], " VIF = ", vif)

[42] vif_cal(input_data=df, dependent_col ='Loan_Status')

     Gender  VIF =  1.17
     Married  VIF =  1.28
     Dependents  VIF =  1.17
     Education  VIF =  1.06
     Self_Employed  VIF =  1.02
     ApplicantIncome  VIF =  1.63
     CoapplicantIncome  VIF =  1.14
     LoanAmount  VIF =  1.71
     Loan_Amount_Term  VIF =  1.05
     Credit_History  VIF =  1.01
     Property_Area  VIF =  1.02
```

Multicollinearity is use on features not a target variable ’Less than 5’ independent then keep such features in sequence. ’Greater than 5’ dependent then drop down such features in sequence.

Support Vector Machine

Support Vector Machine or SVM is one of the most popular supervised learning algorithms, which is used for classification as well as regression problems. However, primarily it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

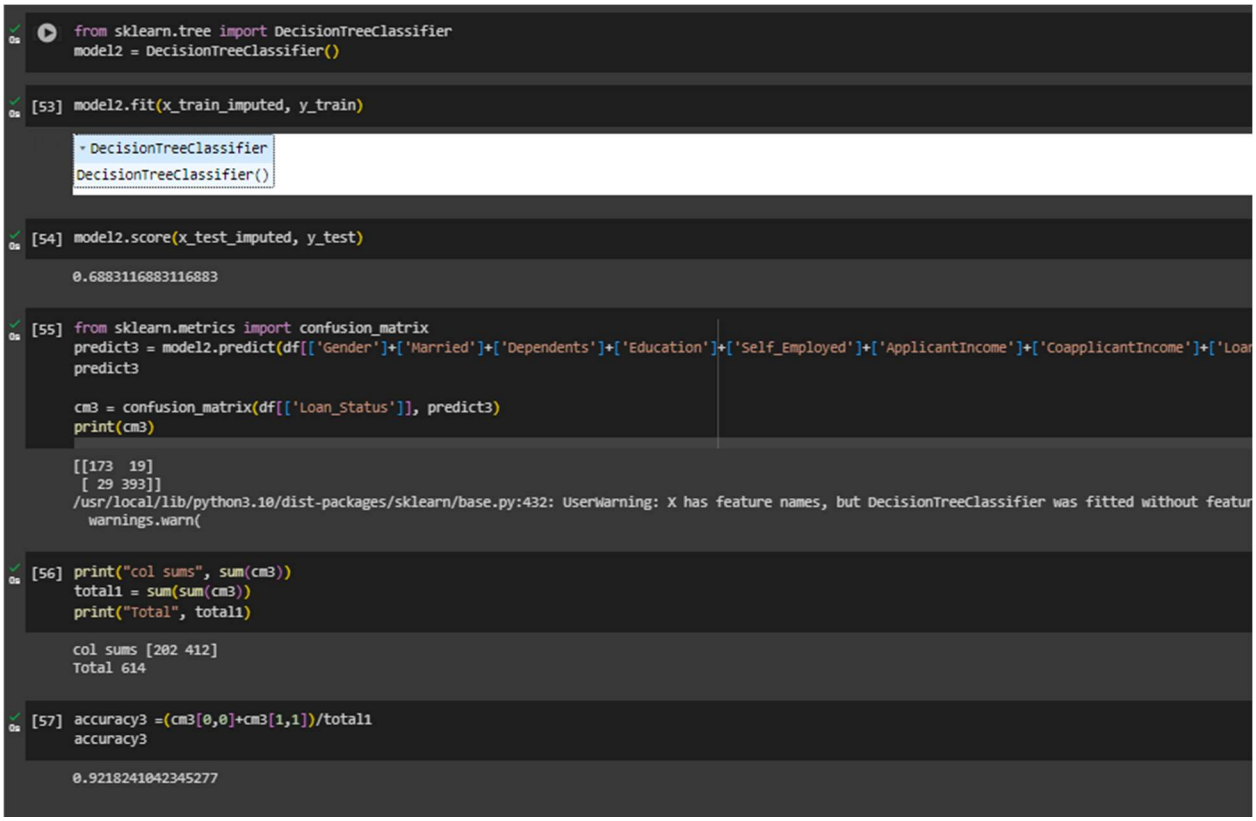
```
[43] from sklearn import svm
     classifier = svm.SVC(kernel="linear")

[44] # Training the support vector machine model
     classifier.fit(x_train_imputed,y_train)

     SVC
     SVC(kernel='linear')

[45] # Model Evaluation
     # Accuracy score on training data
     x_train_prediction = classifier.predict(x_train_imputed)
     training_data_accuracy = accuracy_score(x_train_prediction,y_train)

[46] print("Accuracy on training data:", training_data_accuracy)

     Accuracy on training data: 0.7739130434782608

[47] # Accuracy score on testing data
     x_test_prediction = classifier.predict(x_test_imputed)
     test_data_accuracy = accuracy_score(x_test_prediction, y_test)

[48] print(" Accuracy on testing data:", test_data_accuracy)

     Accuracy on testing data: 0.7662337662337663
```

```
from sklearn.metrics import confusion_matrix
predict2 = classifier.predict((df[['Gender']+['Married']+['Dependents']+['Education']+['Self_Employed']+['ApplicantIncome']+['CoapplicantIncome']+['LoanA
predict2

cm2 = confusion_matrix(df[["Loan_Status"]], predict2)
print(cm2)

[[ 77 115]
 [ 25 397]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SVC was fitted without feature names
  warnings.warn(

[50] print("col sums", sum(cm2))
     total1 = sum(sum(cm2))
     print("Total", total1)

     col sums [102 512]
     Total 614

[51] accuracy2 = (cm2[0,0]+cm2[1,1])/total1
     accuracy2

     0.7719869706840391
```

Decision Tree

Decision Tree is the type of Supervised Machine Learning where the data is continuously split according to a certain parameters. The tree can be explained by two entities viz. decision nodes and leaves. The leaves are the decisions or the final outcomes and the decision nodes are where the data is split.

Classification and regression trees are the two types of decision trees. Tree models where the target variable can take a discrete set of values are classification trees. If the target variable contains continuous values then it is regression tree.



Random Forest

Random Forest is the most used Supervised Machine Learning algorithm for classification and regression. The number of decision trees model together forms a random forest. It is constructed by using multiple decision trees and the final decision is obtained by majority votes of these decision trees. Random Forest works in two-phase first is to create the random forest by combining N decision trees and second is to make predictions for each tree created in the first phase.

```
[58] from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
     model3 = RandomForestClassifier()

[59] model3.fit(x_train_imputed,y_train)

     RandomForestClassifier
     RandomForestClassifier()

[60] model3.score(x_train_imputed,y_train)

     1.0

[61] model3.score(x_test_imputed,y_test)

     0.7922077922077922
```

```
from sklearn.metrics import confusion_matrix
predict4 = model3.predict(df[['Gender']+['Married']+['Dependents']+['Education']+['Self_Employed']+['ApplicantIncome']+['CoapplicantIncome']+['LoanAmount']+['L
predict4

cm4 = confusion_matrix(df[['Loan_Status']], predict4)
print(cm4)
```

```
[[165  27]
 [  5 417]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```
[63] print("col sums", sum(cm4))
     total1 = sum(sum(cm4))
     print("Total", total1)

     col sums [170 444]
     Total 614

[64] accuracy4 =(cm4[0,0]+cm4[1,1])/total1
     accuracy4

     0.9478827361563518
```

Hyperparameter Tunning

```
[65] model = RandomForestClassifier(n_estimators=50,max_depth=3,max_features=11)
     model.fit(x_train_imputed, y_train)

     RandomForestClassifier
     RandomForestClassifier(max_depth=3, max_features=11, n_estimators=50)

[66] model.score(x_train_imputed, y_train)

     0.8195652173913044

[67] model.score(x_test_imputed, y_test)

     0.7987012987012987
```

Experimental Result

```
[68] models = pd.DataFrame({'model': ['Logistic Regression','SVC','Decision Tree','Random Forest'],
                            'Score': [accuracy1,accuracy2,accuracy3,accuracy4]})
     models.sort_values(by ='Score', ascending =False)
```

	model	Score
3	Random Forest	0.947883
2	Decision Tree	0.921824
0	Logistic Regression	0.802932
1	SVC	0.771987

Chapter 5
Conclusion

In this project, we learned how to create models to predict the target variable, i.e. if the applicant will be able to repay the loan or not.

In this study we compare four machine learning prediction models viz. logistic Regression, Support Vector Machine (SVM), Decision Tree and Random Forest.

Out of this Random Forest gives higher accuracy i.e. 94 percentage so we are going to finalize it for future prediction.

Bibliography

[1] https://www.kaggle.com/code/sazid28/home-loan-prediction/data

[2] Machine Learning and Deep Learning Using Python and TensorFlow 1st Edition by Venkata Reddy Konasani, Shailendra Kadre

[3] https://www.youtube.com/watch?v=wNeDJLy0xa4 [4] https://www.youtube.com/watch?v=boC4jKIP V o

■✠■✠■