CHAPTER – 1

1.1 - Abstract

The recent significant increase in loan default has generated interest in understanding the key factors predicting the non-performance of these loans. However, despite the large size of the loan market, existing analyses have been limited by data. This report is based on model created to predicts the repayment of loans by candidates in USA. The detailed analysis is carried out on given data set (614 observations) and each observation in the dataset represents a United States institution of higher education and a specific year. This report presents an analysis of data concerning student loan delinquency. The analysis is based on 6391 observations of automobile data, each containing specific characteristics of the candidtes who are borrowing the loan. After exploring the data by calculating summary and descriptive statistics, and by creating visualizations of the data, several potential relationships between candidate characteristics and loan default were identified. After exploring the data, a predictive model is created to find out the repayment rate for loan. Later, a regression model is created to predict repayment rate of the candidate loan from its feature. The efficiency of the model is increased by identifying the most valuable and contributing features among the given features in the data set. For Minimum Error, we have taken only 13 Feature out of 16 features so that machine learning model created would have a precise prediction for the loan repayment. Thus, this report basically includes the steps followed during the implementation of machine learning model for the student loan repayment prediction and presents the different descriptive statistics and content rich visualization.

1.2 - Problem Statement :

A Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

1.3 - Project Background and Description

The recent significant increase in loan default has generated interest in understanding the key factors predicting the non-performance of these loans. However, despite the large size of the loan market, existing analyses have been limited by data. This report is based on model created to predicts the repayment of loans by potential candidates in USA.

The detailed analysis is carried out on given data set (614 observations) and each observation in the dataset represents a United States institution of higher education and a specific year. This report presents an analysis of data concerning candidate loan delinquency. The analysis is based on 614

observations of automobile data, each containing specific characteristics of the students who are borrowing the loan.

After exploring the data by calculating summary and descriptive statistics, and by creating visualizations of the data, several potential relationships between candidate characteristics and loan default were identified. After exploring the data, a predictive model is created to find out the repayment rate for candidate loan. Later, a regression model is created to predict repayment rate of the student loan from its feature. The efficiency of the model is increased by identifying the most valuable and contributing features among the given features in the data set. For Minimum Error, we have taken only 65 Feature out of 444 features so that machine learning model created would have a precise prediction for the loan repayment.

Thus, this report basically includes the steps followed during the implementation of machine learning model for the student loan repayment prediction and presents the different descriptive statistics and content rich visualization.

1.4 - Importance of Machine learning in financial sector

Using data science in banking industry is more than a trend, it has become a necessity to keep up with the competition. Banks have to realize that big data technologies can help them focus their resources efficiently, make smarter decisions, and improve performance.

Here is a list of data science use cases in banking area which we have combined to give an idea how can you work with significant amounts of data and how to use it effectively.

- Fraud detection
- Managing customer data
- Risk modelling for investment banks
- Personalized marketing

CHAPTER – 2

2.1 - Technology used :

1. Python
2. Numpy
3. Pandas
4. Matplotlib, seaborn
5. Sciket_learn
6. Django
7. HTML, CSS

2.2 - Introduction to Machine learning

“A computer program is said to learn from experience E with some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.”

-Tom M. Mitchell

Consider playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game.

2.2.1 - Examples of Machine Learning

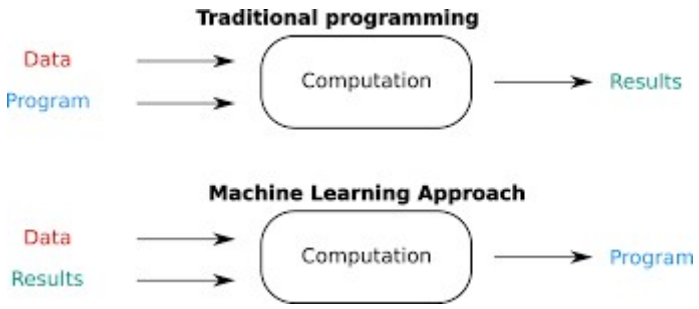
There are many examples of machine learning. Here are a few examples of classification problems where the goal is to categorize objects into a fixed set of categories.

Face detection: Identify faces in images (or indicate if a face is present).

Email filtering: Classify emails into spam and not-spam.

Medical diagnosis: Diagnose a patient as a sufferer or non-sufferer of some disease.

Weather prediction: Predict, for instance, whether or not it will rain tomorrow.



2.3 - Need of Machine Learning

Machine Learning is a field which is raised out of Artificial Intelligence (AI). Applying AI, we wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges.There was a realisation that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realise it while using it.

Finding patterns in data on planet earth is possible only for human brains. The data being very massive, the time taken to compute is increased, and this is where Machine Learning comes into action, to help people with large data in minimum time.

If big data and cloud computing are gaining importance for their contributions, machine learning as technology helps analyse those big chunks of data, easing the task of data scientists in an automated process and gaining equal importance and recognition.

The techniques we use for data mining have been around for many years, but they were not effective as they did not have the competitive power to run the algorithms. If you run deep learning with access to better data, the output we get will lead to dramatic breakthroughs which is machine learning.

2.4 - Kinds of Machine Learning

There are three kinds of Machine Learning Algorithms.

a. Supervised Learning

b. Unsupervised Learning

c. Reinforcement Learning

2.4.1 - Supervised Learning

A majority of practical machine learning uses supervised learning.

In supervised learning, the system tries to learn from the previous examples that are given. (On the other hand, in unsupervised learning, the system attempts to find the patterns directly from the example given.)

Speaking mathematically, supervised learning is where you have both input variables (x) and output variables(Y) and can use an algorithm to derive the mapping function from the input to the output.

Supervised learning problems can be further divided into two parts, namely classification, and regression.

Classification: A classification problem is when the output variable is a category or a group, such as "black" or "white" or "spam" and "no spam".

Regression: A regression problem is when the output variable is a real value, such as "Rupees" or "height."

2.4.2 - Unsupervised Learning

In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data.

Mathematically, unsupervised learning is when you only have input data (X) and no corresponding output variables.

This is called unsupervised learning because unlike supervised learning above, there are no given correct answers and the machine itself finds the answers.

Unsupervised learning problems can be further divided into association and clustering problems.

Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as "people that buy X also tend to buy Y".

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.

2.4.3 - Reinforcement Learning

A computer program will interact with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it continuously trains itself using trial and error method.

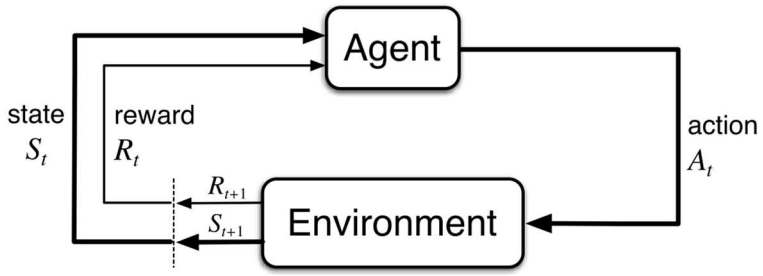


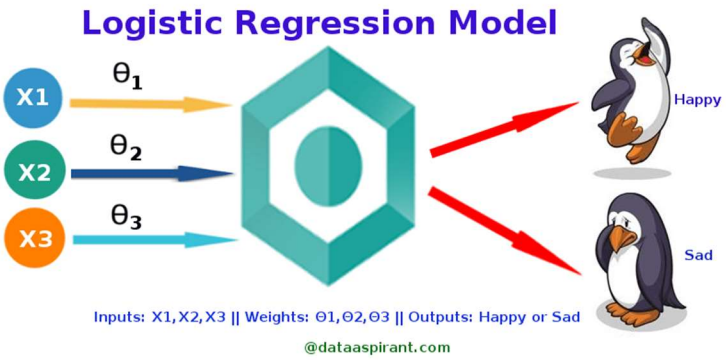
Figure 3.1: The agent–environment interaction in a Markov decision process.

2.5 – Logistic regression

Logistic Regression was used in the biological sciences in early twentieth century. It was then used in many social science applications. Logistic Regression is used when the dependent variable(target) is categorical.

For example,

- To predict whether an email is spam (1) or (0)
- Whether the tumor is malignant (1) or not (0)



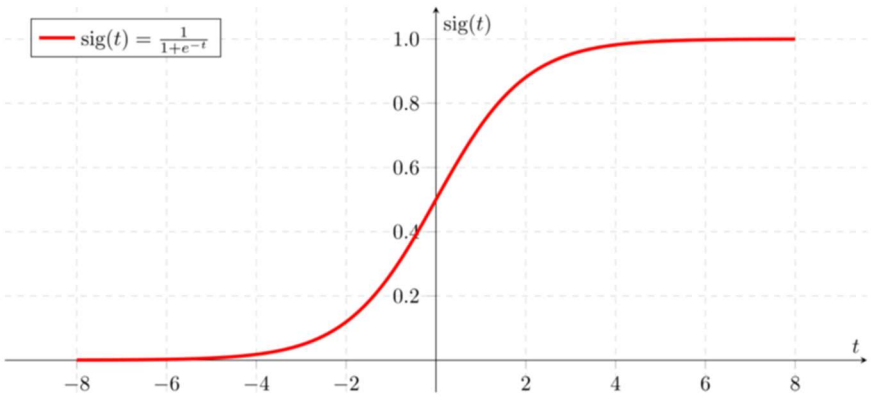
Model

Output = 0 or 1

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

2.5.1 - Sigmoid Function



If 'Z' goes to infinity, Y(predicted) will become 1 and if 'Z' goes to negative infinity, Y(predicted) will become 0.

2.5.2 - Types of Logistic Regression

1. Binary Logistic Regression

The categorical response has only two 2 possible outcomes. Example: Spam or Not

2. Multinomial Logistic Regression

Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

3. Ordinal Logistic Regression

Three or more categories with ordering. Example: Movie rating from 1 to 5

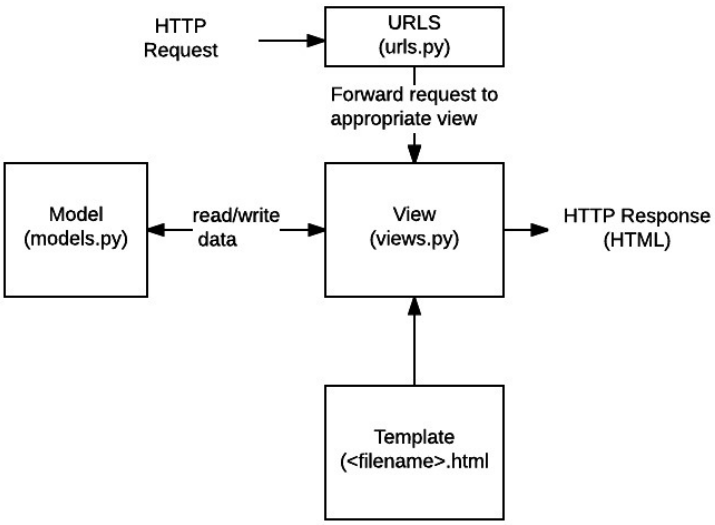
2.6 – Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

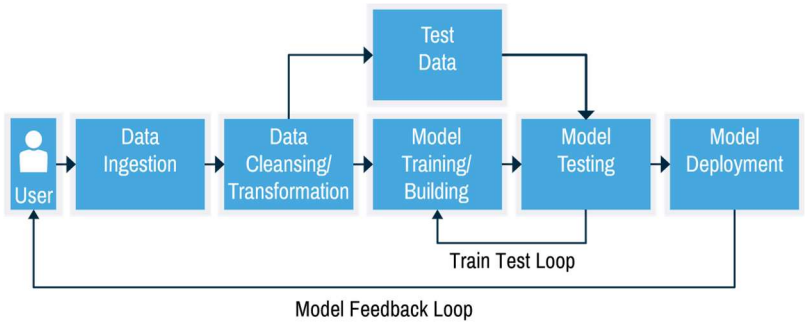
- Complete
- Versatile
- Secure
- Scalable
- Portable

Django web applications typically group the code that handles each of these steps into separate files:



CHAPTER – 3

3.1 – Work flow



From the detection of skin cancer to detecting escalators in need of repairs, or to its evolution today, machine learning has granted computer systems new abilities that we could have never thought of. But what is machine learning and how does it really work under the hood? What actually are the steps to machine Learning?

Machine learning is a field of computer science that gives computers the ability to learn without being programmed explicitly. The power of machine learning is that you can determine how to differentiate using models, rather than using human judgment. The basic steps that lead to machine learning and will teach you how it works are described below in a big picture:

1. Gathering data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyper parameter tuning
7. Prediction.

3.1.1. Gathering Data:

Once you know exactly what you want and the equipments are in hand, it takes you to the first real step of machine learning- Gathering Data. This step is very crucial as the quality and quantity of data gathered will directly determine how good the predictive model will turn out to be. The data collected is then tabulated and called as Training Data.

3.1.2. Data Preparation:

After the training data is gathered, you move on to the next step of machine learning: Data preparation, where the data is loaded into a suitable place and then prepared for use in machine learning training. Here, the data is first put all together and then the order is randomized as the order of data should not affect what is learned.

This is also a good enough time to do any visualizations of the data, as that will help you see if there are any relevant relationships between the different variables, how you can take their advantage and as well as show you if there are any data imbalances present. Also, the data now has to be split into two parts. The first part that is used in training our model, will be the majority of the dataset and the second will be used for the evaluation of the trained model's performance. The other forms of adjusting and manipulation like normalization, error correction, and more take place at this step.

3.1.3. Choosing a model:

The next step that follows in the workflow is choosing a model among the many that researchers and data scientists have created over the years. Make the choice of the right one that should get the job done.

3.1.4. Training:

After the before steps are completed, you then move onto what is often considered the bulk of machine learning called training where the data is used to incrementally improve the model's ability to predict.

The training process involves initializing some random values for say A and B of our model, predict the output with those values, then compare it with the model's prediction and then adjust the values so that they match the predictions that were made previously.

This process then repeats and each cycle of updating is called one training step.

3.1.5. Evaluation:

Once training is complete, you now check if it is good enough using this step. This is where that dataset you set aside earlier comes into play. Evaluation allows the testing of the model against data that has never been seen and used for training and is meant to be representative of how the model might perform when in the real world.

3.1.6. Parameter Tuning:

Once the evaluation is over, any further improvement in your training can be possible by tuning the parameters. There were a few parameters that were implicitly assumed when the training was done. Another parameter included is the learning rate that defines how far the line is shifted during each step, based on the information from the previous training step. These values all play a role in the accuracy of the training model, and how long the training will take.

For models that are more complex, initial conditions play a significant role in the determination of the outcome of training. Differences can be seen depending on whether a model starts off training with values initialized to zeroes versus some distribution of values, which then leads to the question of which distribution is to be used. Since there are many considerations at this phase of training, it's important that you define what makes a model good. These parameters are referred to as Hyper

parameters. The adjustment or tuning of these parameters depends on the dataset, model, and the training process. Once you are done with these parameters and are satisfied you can move on to the last step.

3.1.7. Prediction:

Machine learning is basically using data to answer questions. So this is the final step where you get to answer few questions. This is the point where the value of machine learning is realized. Here you can Finally use your model to predict the outcome of what you want.

CHAPTER – 4

4.1 - Data Analysis :

Firstly, we check the features present in our data and then we will look at their data types.

```
In [2]: #Reading data from respective data files
        train = pd.read_csv("Train.csv")
        test = pd.read_csv("Test.csv")

In [3]: #Making copies, not to lose the originals
        train_original=train.copy()
        test_original=test.copy()
```

Firstly, we check the features present in our data and then we will look at their data types.

```
In [4]: train.columns

Out[4]: 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. Let's also have a look at the columns of test dataset.

```
In [5]: test.columns

Out[5]: Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
               'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
               'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
              dtype='object')
```

We have similar features in the test dataset as the train dataset except the Loan_Status. We will predict the Loan_Status using the model built using the train data.

Program 4.1

We have similar features in the test dataset as the train dataset except the Loan_Status. We will predict the Loan_Status using the model built using the train data.

```
In [6]: #Printing data types for each variable
        train.dtypes

Out[6]: 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
```

Program 4.2

We can see there are three format 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 involved. They are also

numerical variables. Numerical variables in our dataset are: CoapplicantIncome, LoanAmount, Loan_Amount_Term, and Credit_History

Let's look at the shape of the dataset.

```
In [8]: #Returns number of rows and columns
        train.shape , test.shape

Out[8]: ((614, 13), (367, 12))
```

Program 4.3

4.1.1 Univariate Analysis

Target Variable: We will first look at the target variable, i.e., Loan_Status. As it is a categorical variable, let us look at its frequency table, percentage distribution and bar plot.

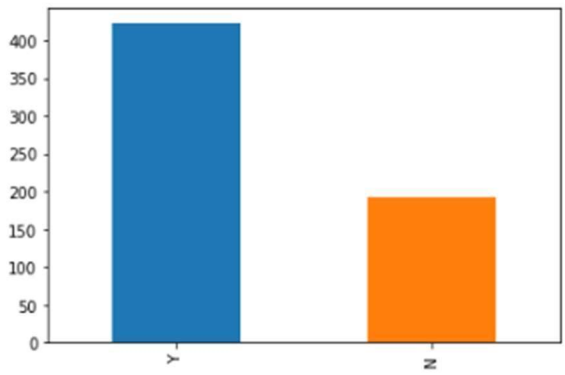
Frequency table of a variable gives the count of each category in that variable.

```
In [9]: train['Loan_Status'].value_counts()

Out[9]: Y    422
        N    192
        Name: Loan_Status, dtype: int64

In [10]: # Normalize can be set to True to print proportions instead of number
         train['Loan_Status'].value_counts(normalize=True)

Out[10]: Y    0.687296
         N    0.312704
         Name: Loan_Status, dtype: float64
```

Program 3.4

Now we will do some data visualization

```
In [11]: train['Loan_Status'].value_counts().plot.bar()

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1e7c2105208>
```



Graph 4.1

The loan of 422(around 69%) people out of 614 was approved. Now visualizing each variable separately. Different types of variables are Categorical, ordinal and numerical.

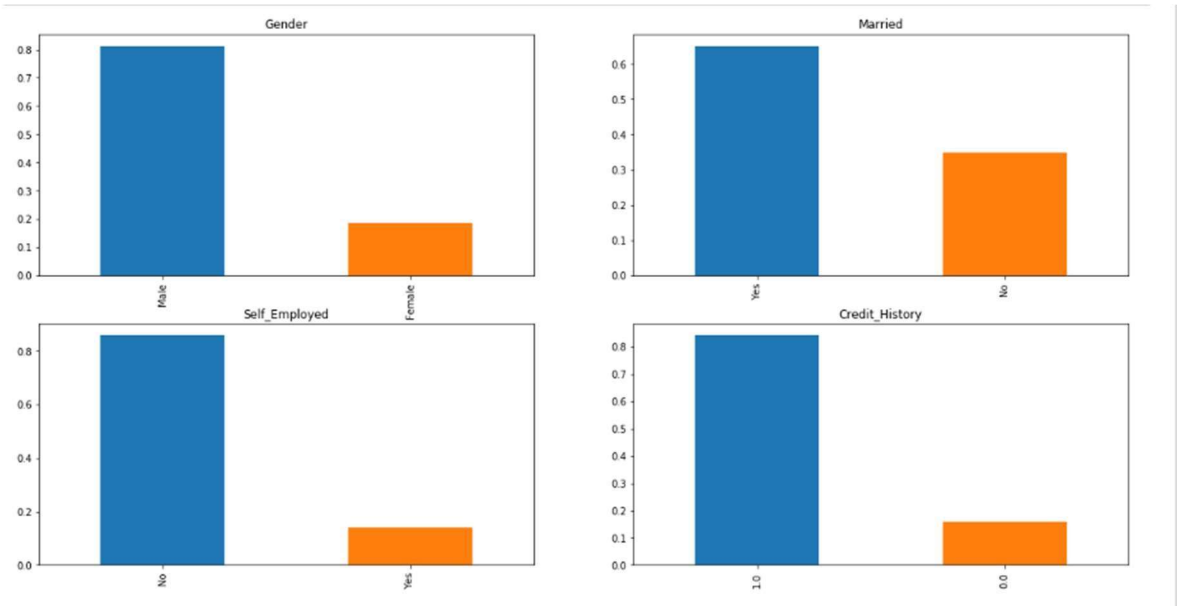
***Categorical features:** These features have categories (Gender, Married, Self_Employed, Credit_History, Loan_Status)

***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)

4.1.2 Independent Variable (Categorical)

```
In [12]: 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.show()
```



Graph 4.2

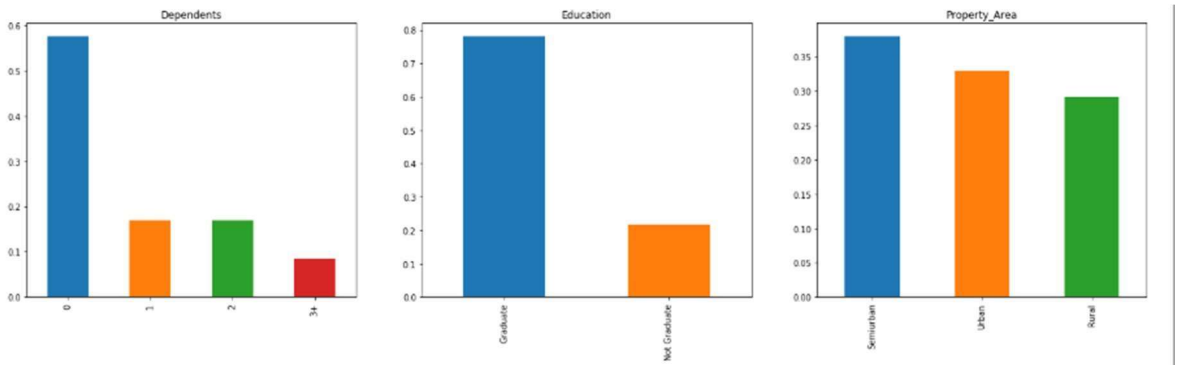
It can be inferred from the above bar plots that:

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 debts.

4.1.3 Independent Variable (Ordinal)

```
In [13]: 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(title= 'Education')

         plt.subplot(133)
         train['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Property_Area')

         plt.show()
```

Program 4.5



Graph 4.3

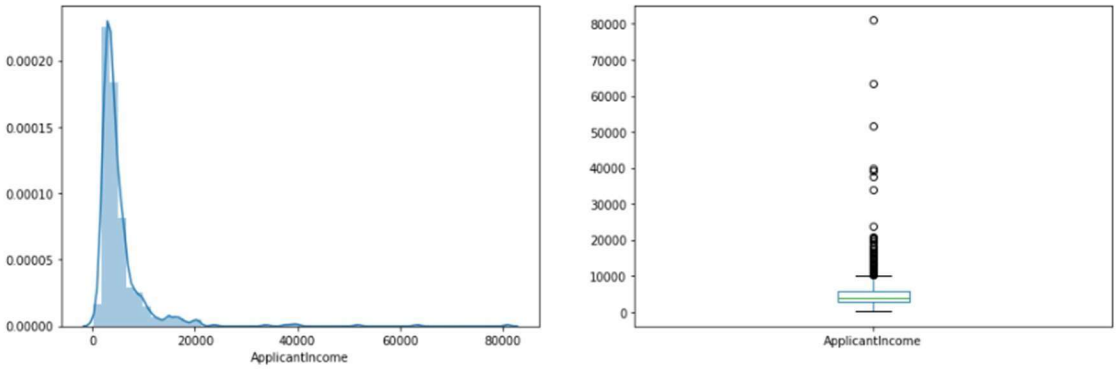
Following inferences can be made from the above bar plots:

Most of the applicants don't have any dependents. Around 80% of the applicants are Graduate. Most of the applicants are from Semiurban area.

4.1.4 Independent Variable (Numerical)

```
In [14]: #Distributution of Applicant Income
         plt.figure(1)
         plt.subplot(121)
         sns.distplot(train['ApplicantIncome']);

         plt.subplot(122)
         train['ApplicantIncome'].plot.box(figsize=(16,5))

         plt.show()
```

Program 4.6



Graph 4.4

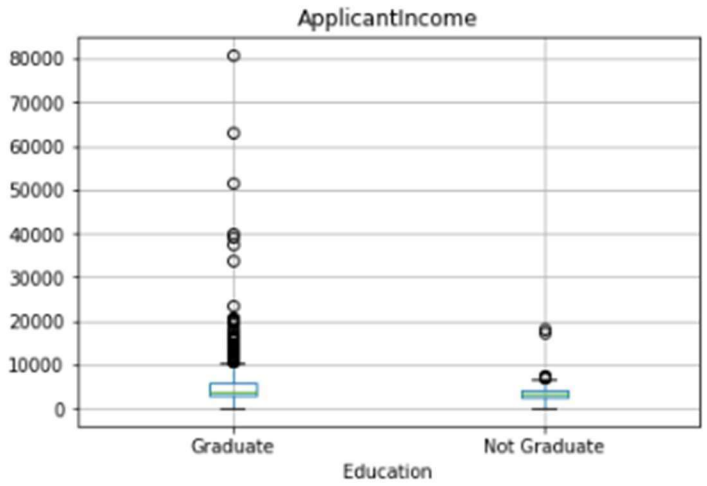
It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed. We should make it normal as algorithms works better if the data is normally distributed.

The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society. Part of this can be driven by the fact that we are looking at people with different education levels. So segregating them by Education:

```
In [15]: train.boxplot(column='ApplicantIncome', by = 'Education')
         plt.suptitle("")

Out[15]: Text(0.5,0.98,'')
```

Program 4.7

```
In [15]: train.boxplot(column='ApplicantIncome', by = 'Education')
         plt.suptitle("")

Out[15]: Text(0.5,0.98,'')
```

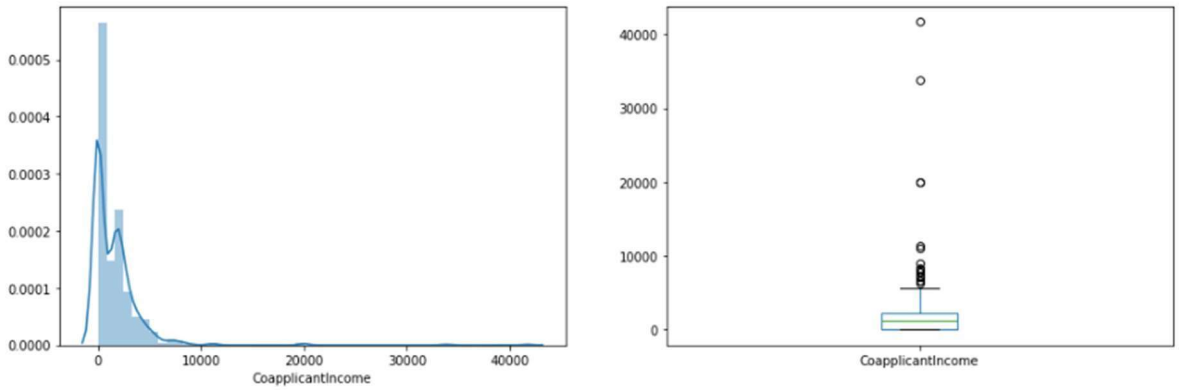


Graph 4.5

It is clear that there are a higher number of graduates with very high incomes, which are appearing to be the outliers.

Looking at the Coapplicant income distribution:

```
In [16]: plt.figure(1)
         plt.subplot(121)
         sns.distplot(train['CoapplicantIncome']);

         plt.subplot(122)
         train['CoapplicantIncome'].plot.box(figsize=(16,5))

         plt.show()
```

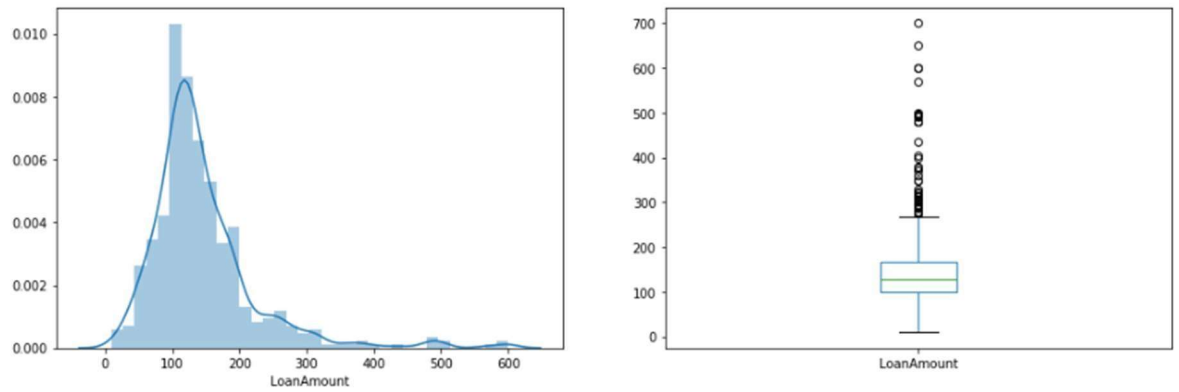


Graph 4.6

Similar distribution is seen as that of the applicant income. Majority of coapplicant's income ranges from 0 to 5000. Lot of outliers can also be seen in the coapplicant income and it is not normally distributed.

Now looking at the distribution of LoanAmount variable:

```
In [17]: plt.figure(1)
         plt.subplot(121)
         df=train.dropna()
         sns.distplot(df['LoanAmount']);

         plt.subplot(122)
         train['LoanAmount'].plot.box(figsize=(16,5))

         plt.show()
```



Graph 4.7

We see a lot of outliers in this variable and the distribution is fairly normal. These outliers are to be treated.

Now we would like to know how well each feature correlate with Loan Status. So, in the next section we will look at bivariate analysis.

4.1.5 Bivariate Analysis

Lets recall some of the hypothesis that I generated earlier:

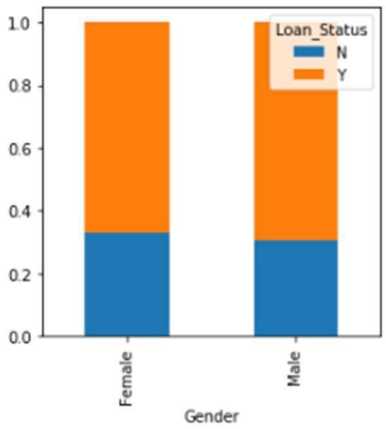
Applicants with high income should have more chances of loan approval. Applicants who have repaid their previous debts should have higher chances of loan approval. Loan approval should also depend on the loan amount. If the loan amount is less, chances of loan approval should be high. Lesser the amount to be paid monthly to repay the loan, higher the chances of loan approval. Now testing the above mentioned hypotheses using bivariate analysis

After looking at every variable individually in univariate analysis,now exploring them again with respect to the target variable.

4.1.6 Categorical Independent Variable vs Target Variable

First of all relation between target variable and categorical independent variables is to be find. Now looking at the stacked bar plot now which will give us the proportion of approved and unapproved loans.

```
In [18]: 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))
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1e7c23197b8>
```

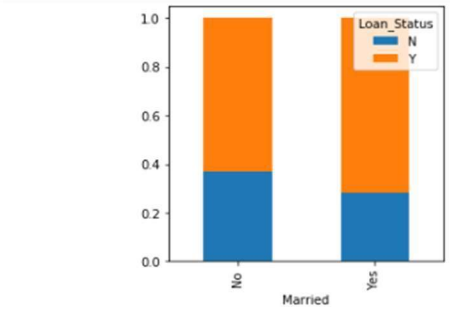


Graph 4.8

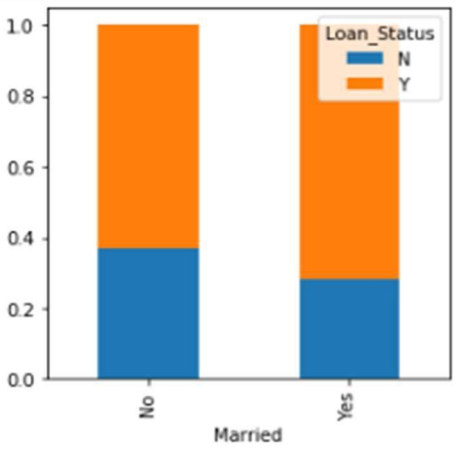
It can be inferred that the proportion of male and female applicants is more or less same for both approved and unapproved loans.

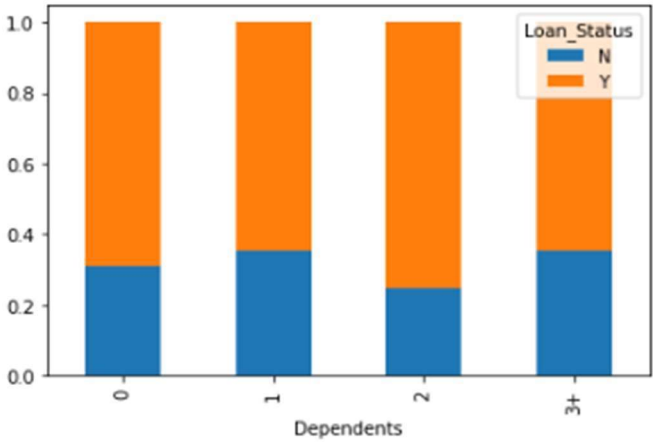
Now visualizing the remaining categorical variables vs target variable.

```
In [19]: 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))
         plt.show()

         Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
         plt.show()

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

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



Graph 4.9





Graph 4.10

Proportion of married applicants is higher for the approved loans. Distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status. There is nothing significant inferred from Self_Employed vs Loan_Status plot. Now looking at the relationship between remaining categorical independent variables and Loan_Status.

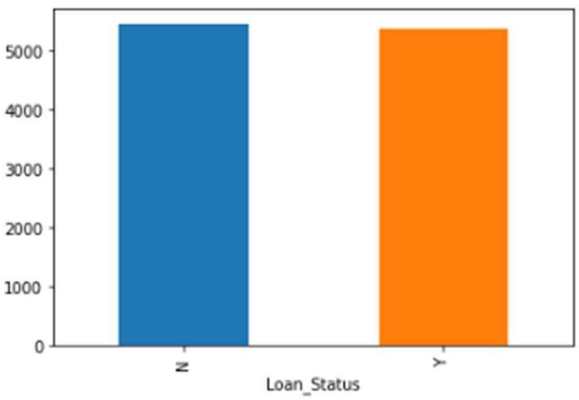
```
In [20]: Credit_History=pd.crosstab(train['Credit_History'],train['Loan_Status'])
         Property_Area=pd.crosstab(train['Property_Area'],train['Loan_Status'])

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

         Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
         plt.show()
```

Graph 4.11

It seems people with credit history as 1 are more likely to get their loans approved. Proportion of loans getting approved in semiurban area is higher as compared to that in rural or urban areas. Now visualize numerical independent variables with respect to target variable.

Numerical Independent Variable vs Target Variable Now finding the mean income of people for which the loan has been approved vs the mean income of people for which the loan has not been approved.

```
In [21]: train.groupby('Loan_Status')['ApplicantIncome'].mean().plot.bar()
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1e7c2338d68>
```

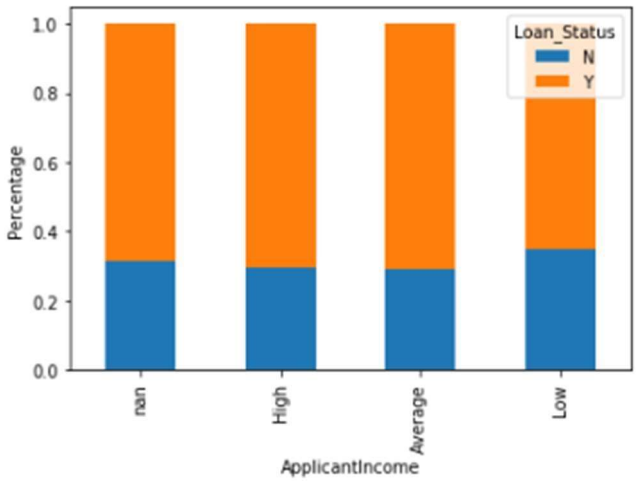


Graph 4.12

Here the y-axis represents the mean applicant income. There is no significant change in the mean income. So, bins are to be made for the applicant income variable based on the values in it and analyze the corresponding loan status for each bin.

```
In [22]: bins=[0,2500,4000,6000,81000]
         group=['Low','Average','High', 'Very high']
         train['Income_bin']=pd.cut(df['ApplicantIncome'],bins,labels=group)
```

```
In [23]: 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')
```

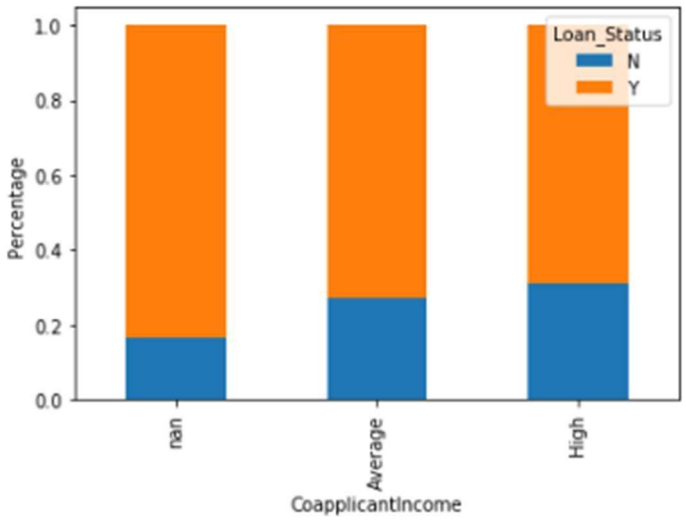


Graph 4.13

It can be inferred that Applicant income does not affect the chances of loan approval which contradicts our hypothesis in which it was assumed that if the applicant income is high the chances of loan approval will also be high.

Now analyzing the coapplicant income and loan amount variable in similar manner.

```
In [24]: bins=[0,1000,3000,42000]
         group=['Low','Average','High']
         train['Coapplicant_Income_bin']=pd.cut(df['CoapplicantIncome'],bins,labels=group)

In [25]: 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')
```



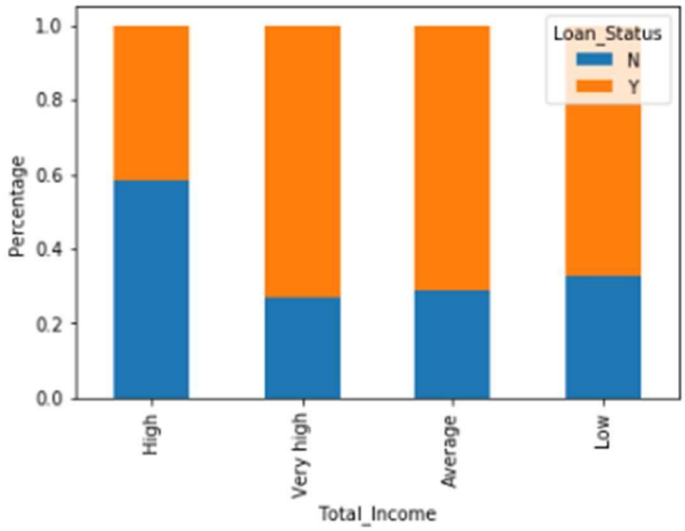
Graph 4.14

It shows that if coapplicant's 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 coapplicant so the coapplicant income for such applicants is 0 and hence the loan approval is not dependent on it. So it is good to make a new variable in which we will combine the applicant's and coapplicant's income to visualize the combined effect of income on loan approval.

Now combining the Applicant Income and Coapplicant Income and see the combined effect of Total Income on the Loan_Status.

```
In [26]: train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']

In [27]: bins=[0,2500,4000,6000,81000]
         group=['Low','Average','High', 'Very high']
         train['Total_Income_bin']=pd.cut(train['Total_Income'],bins,labels=group)

In [28]: 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')
```

Program 4.8



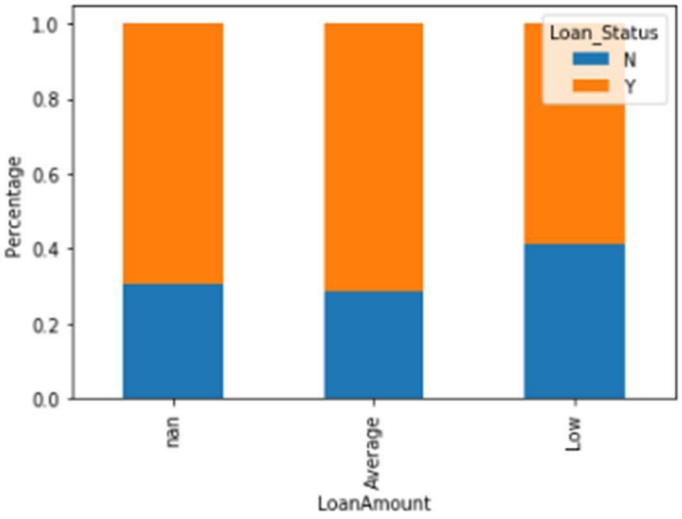
Graph 4.15

We can see that 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.

Let's visualize the Loan amount variable

```
In [29]: bins=[0,100,200,700]
         group=['Low','Average','High']
         train['LoanAmount_bin']=pd.cut(df['LoanAmount'],bins,labels=group)

In [30]: 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')
```

Program 3.9



Graph 4.16

It can be seen that the proportion of approved loans is higher for Low and Average Loan Amount as compared to that of High Loan Amount which supports our hypothesis in which it was considered that the chances of loan approval will be high when the loan amount is less.

Let's drop the bins which we created for the exploration part. We will change the 3+ in dependents variable to 3 to make it a numerical variable.We will also convert the target variable's categories into 0 and 1 so that we can find its correlation with numerical variables. One more reason to do so is few models like logistic regression takes only numeric values as input. We will replace N with 0 and Y with 1.

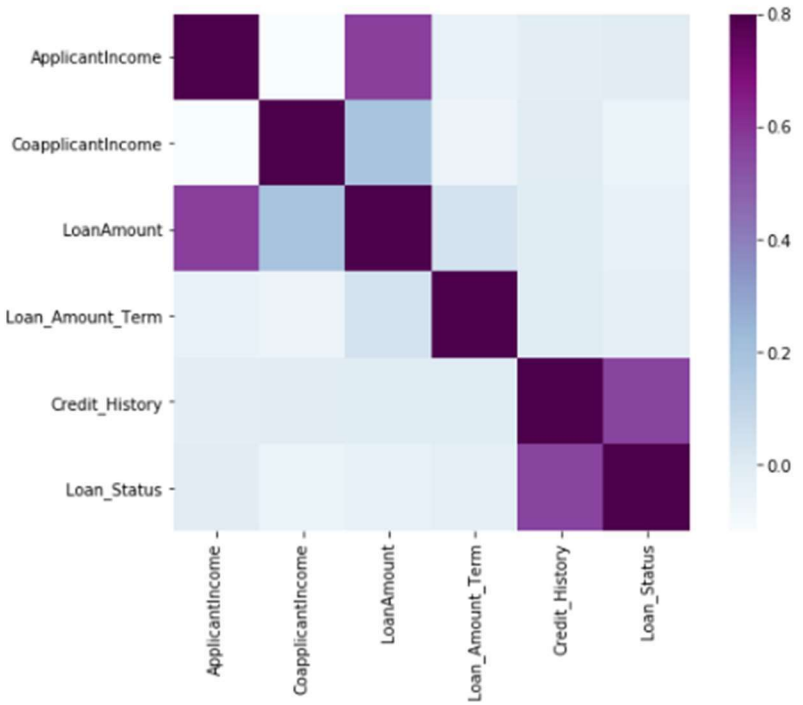
```
In [31]: train=train.drop(['Income_bin', 'Coapplicant_Income_bin', 'LoanAmount_bin', 'Total_Income_bin', 'Total_Income'], axis=1)

In [32]: 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)
```

Program 4.10

Now going to look at the correlation between all the numerical variables. We will use the heat map to visualize the correlation. Heatmaps visualize data through variations in coloring. The variables with darker color means their correlation is more.

```
In [33]: matrix = train.corr()
         f, ax = plt.subplots(figsize=(9, 6))
         sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu");
```

Program 4.11



Graph 4.17

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

4.2 Missing Value Treatment

After exploring all the variables in our data, we can now impute the missing values and treat the outliers because missing data and outliers can have adverse effect on the model performance.

Missing value imputation Let's list out feature-wise count of missing values.

```
In [34]: train.isnull().sum()

Out[34]: 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
```

Program 4.12

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 the 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.

```
In [35]: 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)
```

```
Out[36]: 360.0    512
         180.0     44
         480.0     15
         300.0     13
         84.0       4
         240.0      4
         120.0      3
         36.0       2
         60.0       2
         12.0       1
         Name: Loan_Amount_Term, dtype: int64
```

Program 4.13

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.

this variable.

```
In [37]: train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=True)
```

Program 4.14

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 median to fill the null values as earlier we saw that loan amount have outliers so the mean will not be the proper approach as it is highly affected by the presence of outliers.

```
In [40]: 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)
```

Program 4.15

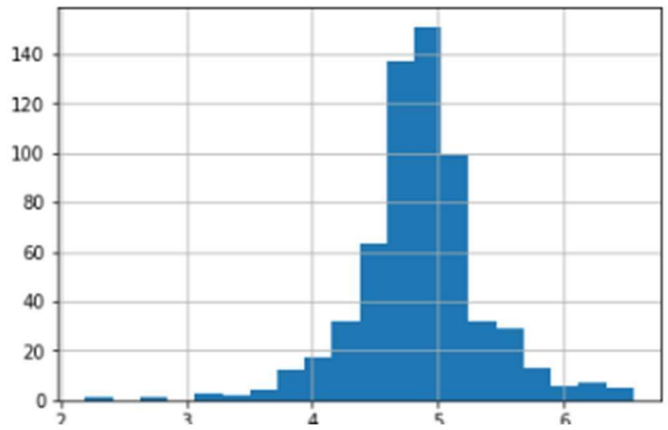
4.3 Outlier Treatment :

We must take steps to remove outliers from our data sets.

Due to these outliers bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skewness. One way to remove the skewness is by doing the log transformation. 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 normal distribution.

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

```
In [41]: train['LoanAmount_log'] = np.log(train['LoanAmount'])
         train['LoanAmount_log'].hist(bins=20)
         test['LoanAmount_log'] = np.log(test['LoanAmount'])
```



Program 4.16

Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided. Let's build a logistic regression model and make predictions for the test dataset

4.4 Model Building

Now making the model to predict the target variable 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. Logistic regression is an estimation of Logit function. Logit function is simply a log of odds in favor of the event. This function creates a s-shaped curve with the probability estimate, which is very similar to the required step wise function
Lets 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.

```
In [42]: train=train.drop('Loan_ID',axis=1)
         test=test.drop('Loan_ID',axis=1)
```

Program 4.17

We will use scikit-learn (sklearn) for making different models which is an open source library for Python. It is one of the most efficient tool which contains many inbuilt functions that can be used for modeling in Python.
Sklearn requires the target variable in a separate dataset. So, we will drop our target variable from the train dataset and save it in another dataset.

Now we will make dummy variables for the categorical variables. Dummy variable turns categorical variables into a series of 0 and 1, making them lot easier to quantify and compare. Let us understand the process of dummies first:

Consider the “Gender” variable. It has two classes, Male and Female. As logistic regression takes only the numerical values as input, we have to change male and female into numerical value. Once we apply dummies to this variable, it will convert the “Gender” variable into two variables(Gender_Male and Gender_Female), one for each class, i.e. Male and Female. Gender_Male will have a value of 0 if the gender is Female and a value of 1 if the gender is Male.

```
In [58]: X=pd.get_dummies(X)
         train=pd.get_dummies(train)
         test=pd.get_dummies(test)
```

Program 4.18

Now we will train the model on training dataset and make predictions for the test dataset. But can we validate these predictions? One way of doing this is we can divide our train dataset into two parts: train and validation. We can train the model on this train part and using that make predictions for the validation part. In this way we can validate our predictions as we have the true predictions for the validation part (which we do not have for the test dataset).

We will use the train_test_split function from sklearn to divide our train dataset. So, first let us import train_test_split.

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.

```
In [51]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score

In [52]: model = LogisticRegression()
         model.fit(x_train, y_train)

Out[52]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm start=False)
```

Program 4.19

Here the C parameter represents inverse of regularization strength. Regularization is applying a penalty to increasing the magnitude of parameter values in order to reduce overfitting. Smaller values of C specify stronger regularization.
Let's predict the Loan_Status for validation set and calculate its accuracy.

```
In [53]: pred_cv = model.predict(x_cv)

In [54]: pred_cv

Out[54]: array([1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1,
                1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

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

```
In [55]: accuracy_score(y_cv,pred_cv)

Out[55]: 0.8
```

Program 4.20

So the predictions are almost 80% accurate, i.e. 80% of the loan status are identified correctly.

```
In [57]: pred_test

Out[57]: array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
                1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
                1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

Program 3.21

4.5 Model deployment

This trained model is deployed using Django which is a python web framework. following figure is the entire directory of Django project. In loanML is our main directory. Under this we have loanML(folder), templates(folder), manage.py(python file) and Train.csv(Data file).
We can run our server using manage.py file. In views.py we have our machine learning model and some other functions. Urls.py file is to navigate the urls and settings.py have all the settings we are using in our Django project.

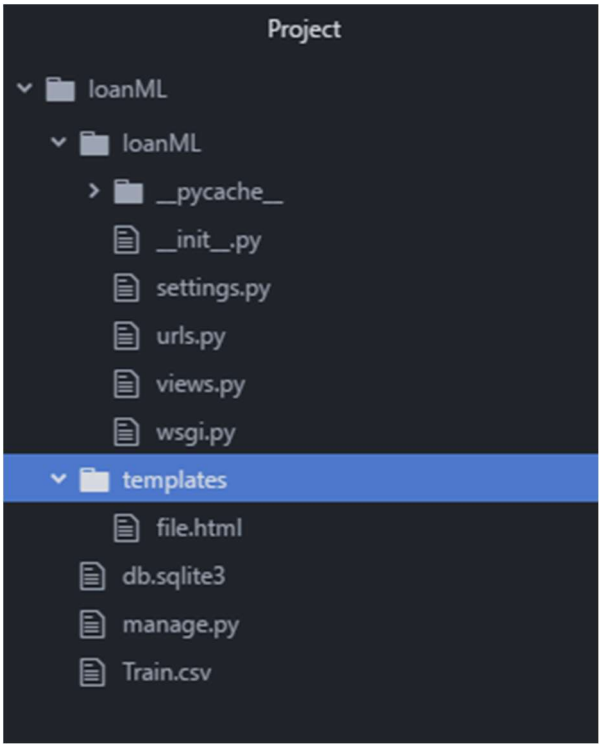


Figure 1

We runserver using **python manage.py runserver** command in command prompt.

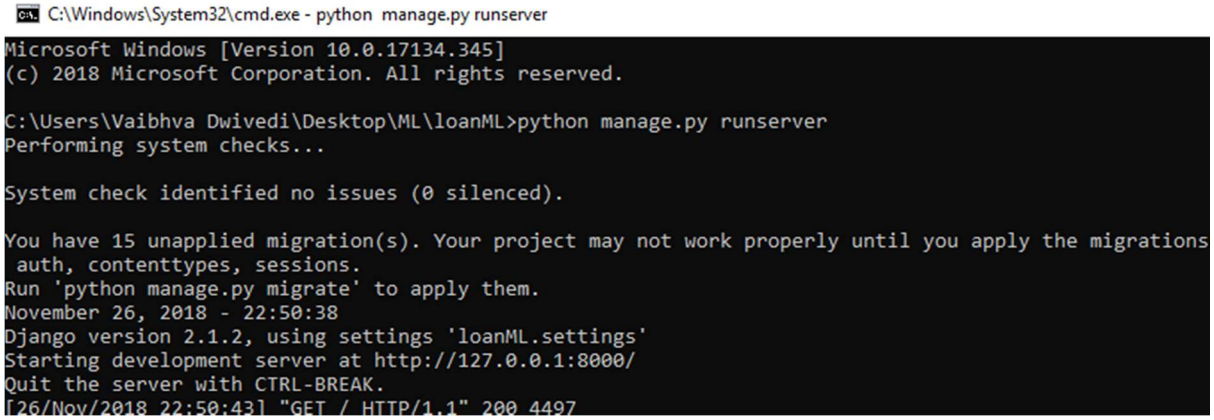


Figure 2

After running the sever this is how our main page looks like.

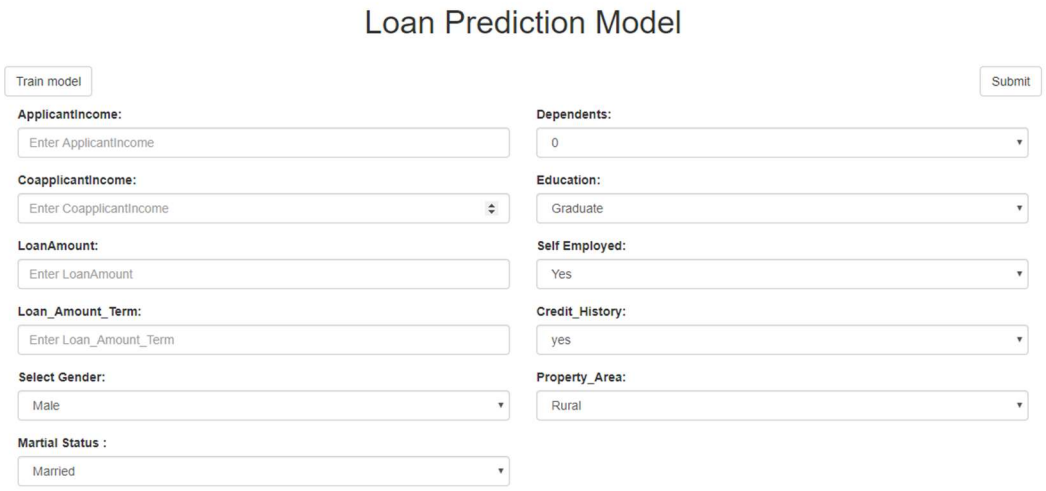


Figure 3

When we input all these entries we get a pop-up massage whether the applicant is potential or not for loan

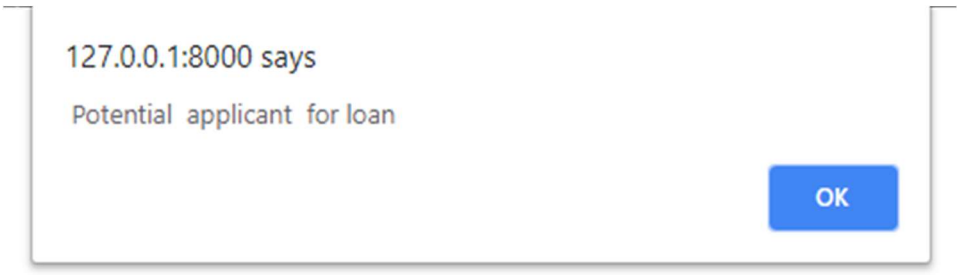


Figure 4

CHAPTER – 5

5.1 Conclusion

Using data science in the banking industry is more than a trend, it has become a necessity to keep up with the competition. Banks have to realize that big data technologies can help them focus their resources efficiently, make smarter decisions, and improve performance.
And this loan prediction model help banks to know whether the applicant is good for loan approval or not basis of previous data records. This helps financial corporation to reach potential customers to get more profit and risk free investment.

5.2 References

1. Python for Data analysis
2. Building machine learning systems using python
3. Think States 2

5.3 Tutorials

1. W3shcools [https:\\www.w3chools.com]
2. Data science website kaggle [https:\\www.kaggle.com]
3. Django documentation [https:\\docs.djangoproject.com\en\2.1]