# Loan Eligibility Prediction


## Team Mentor:

Sanjay Basumatary

## Team Members:

1. Sahil Jethva (**Team Leader**)
2. B.Sriharsha
3. Heeta Parmar
4. Thandava Krishna

# Abstract

In the banking system, banks have a variety of products to provide, but credit lines are their primary source of revenue. As a result, they will profit from the interest earned on the loans they make. Loans, or whether customers repay or default on their loans, affect a bank's profit or loss. The bank's Non-Performing Assets will be reduced by forecasting loan defaulters. As a result, further investigation into this occurrence is essential. Because precise forecasts are essential for benefit maximization, it's crucial to analyze and compare the various methodologies. The logistic regression model is an important predictiveanalytics tool for detecting loan defaulters. The two most pressing issues in the banking sector are: 1) How risky is the borrower? 2) Should we lend to the borrower given the risk. Banks make loans to customers in exchange for the guarantee of repayment. Some would default on their debts, unable to repay them for a number of reasons. The bank retains insurance to minimize the possibility of failure in the case of a default. The insured sum can cover the whole loan amount or just a portion of it. At that time, making a decision would take a long time. As a result, the loan prediction machine learning model can be used to assess a customer's loan status and build strategies. This model extracts and introduces the essential features of a borrower that influence the customer's loan status. Finally, it produces the planned performance (loan status). These reports make a bank manager's job simpler and quicker.


## 1. Problem Statement

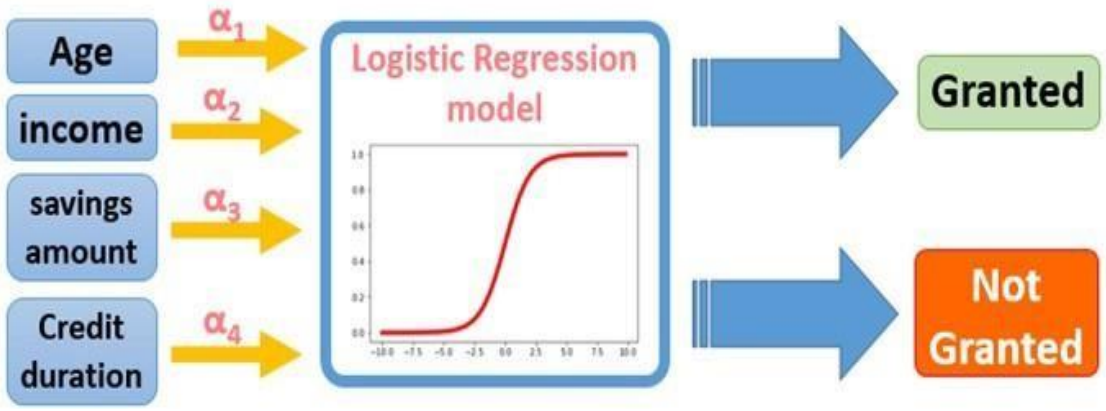
Loans are the core business of banks. The main profit comes directly from the loan’s interest. The loan companies grant a loan after an intensive process ofverification and validation. Even after this process, the banks are not assured ofloan recovery. They face a huge loss as the borrower delays or completely ignores to pay the loan taken from the bank.

This may arise due to many reasons some of them would be misinterpretation or favoritism with the borrower or collectively can be considered as human error. To overcome this problem, we can develop a model which predicts whether the borrower would pay the loan on time using Machine Learning.

When seen from the borrower’s side even, there are many instances where the borrower is capable of repaying the loan promptly but the loan gets rejected due to some orthodox decisions where the loan providing is decided on some single factor.

## 2. Business Need Assessment

As mentioned above, Loans are the primary income for the banks as their main profit comes directly from the loan's interest. If the Machine Learning model is able to predict the extent of the loan recovery from the individual efficiently, Then the banks would have a better profit as they would assess the data of the borrower and check the prediction of the model, decide whether to provide loan to the borrower or not.




## 3. Target Specification and Characterization

The target audience is the banks that provide the loan. Most of the banks would be willing to provide as much as loans as possible which would increase their profit but recovery is always the biggest fear which makes them a bit more conscious in providing the loan.

Factors considered in this model are-

- Gender (M or F)
- Marital status (Yes or No)
- Dependents (number of persons dependent)
- Education (education of the applicant)
- Self-employed (Yes or No)
- Co-applicant income (income of the-applicant)
- Loan amount (required amount of the borrower)
- Loan amount term (term duration)
- Credit history (ability to repay the loan)
- Property area(urban, semi-urban or rural)

# 4. External Search

The model uses a dataset from a repository on GitHub which consists of 614 rows of data that shows on what basis the loan was provided to the applicant. Basedon this data the model would predict whether the applicant (new data) is eligible for the loan or not.

```
In [60]: train = pd.read_csv(r"C:/Users/SWAYAM/Downloads/wf/train_data.csv")
         train.head()
```

Out[60]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | 360.0 | 1.0 |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | 360.0 | 1.0 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | 360.0 | 1.0 |

```
In [61]: test = pd.read_csv(r'C:\Users\SWAYAM\Downloads\wf\test.csv')
         test.head()
```

Out[61]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001015 | Male | Yes | 0 | Graduate | No | 5720 | 0 | 110.0 | 360.0 | 1.0 |
| 1 | LP001022 | Male | Yes | 1 | Graduate | No | 3076 | 1500 | 126.0 | 360.0 | 1.0 |
| 2 | LP001031 | Male | Yes | 2 | Graduate | No | 5000 | 1800 | 208.0 | 360.0 | 1.0 |
| 3 | LP001035 | Male | Yes | 2 | Graduate | No | 2340 | 2546 | 100.0 | 360.0 | NaN |
| 4 | LP001051 | Male | No | 0 | Not Graduate | No | 3276 | 0 | 78.0 | 360.0 | 1.0 |

# 5. Benchmarking alternate products

There are many independent loan eligibilities predicting firms and banks also maintain their own system to predict whether an applicant is eligible to receive the loan or not. One such independent Organisation is “allcloud.in”. It provides various functionalities in which the “Loan Originating system” is such program which gives streamlines processes across various channels and products thus enabling banks to issue loans and manage underwriting services in a few minutes, absolutely paperless.

# 6. Applicable Patents

The organization “allcloud.in” is open-source software and it is free to integrate with any other platforms.

## 7. Applicable Constraints

- All necessary libraries in python like pandas, NumPy, sci-kit learn, and matplotlib to execute the code cell is required
- A cloud space to store, update and retrieve the datasets
- An AI/ML expert body to monitor the working of the model and improviseit timely for better performance
- An easy-to-use UI for the client
- Continuous data collection and maintenance.
- Product usage can be found not so reassuring as models have low accuracy, Optimization is required in this case.
- Marketing the product to a Training center or a Software development company.
- The product developed should be responsive such that customers who are not educated will be able to easily use the product.

## 8. Business Opportunity

If a company provides a better prediction of loan eligibility than the banks then most of them would prefer using the services from the company as their profits increases as the loan defaulters are kept as minimum as possible.

This would increase the company's engagement with the banks and hence a good revenue is generated as the banks trust the predictions of the company.

## 9. Concept Generation

The term "Loan" is both a give and take thing, Banks need to keep providing loans to increase their income of it and the applicant would require a loan for his/her personal reasons. Hence an easy way to help both banks and applicants in providing and getting loans respectively would be by using this machine learning model.

## 10. Concept Development

Firstly, we collect datasets of all types of applicants, remove all the unwanted data and train the machine learning model with that refined data. Once the model is trained, we can take the values from the user (applicant details are entered by the bank) and give an output of whether the applicant is eligible for the loan or not.

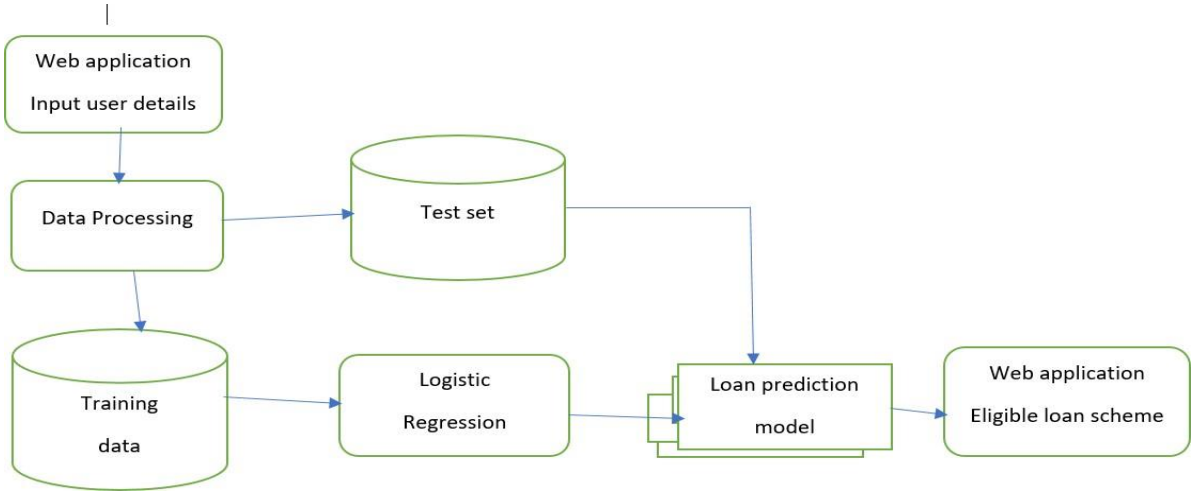
We can divide our project into two phases

First phase: In this phase, our model is more helpful to banks than the applicants as it only predicts whether the applicant is eligible for getting the loan or not.

Second phase: In this phase, the model takes the input, gives the output (whether the applicant is eligible for the loan or not), and also gives the possible ways for the applicant to get the loan (by decreasing the loan amount or increasing the interest, etc).

## 11. Final Product Prototype

**Schematic Diagram:** The schematic diagram for the above model can be



## Product Details

- **Working**

  The machine learning model works using logistic regression which classifies the details entered by the user (applicant details) into whether the applicant can receive the loan or not.

- **Data Sources**
  We use datasets for training and testing the model from Kaggle
- **Algorithms**

  We use the Logistic Regression algorithm as it gives the best accuracy among allother algorithms when tested with the same test data set (83%)

- **Team required to develop**

  To develop this project, we need a team that consists of members from the Machine Learning field and banking field where the majority can be membersfrom the Machine Learning field.

## Business Model:

For the Business model, we can implement any of the three business models given below

- Direct sales Bm
- Consulting bm
- Freemium bm can be implemented.

In direct sales, we give the customer the product like we give access to the website where he gives inputs and can be sure whether his application for the loan will get approved or not. Consulting is also similar but here he gives input to us and we use the model to give him our opinions on how he improves his situation to get loan approval from the bank. Freemium: we give some access (i.e., some no of inputs accessed out of the total no of inputs our model uses) to our final product based on that he will / not decides to buy

our product. This is from a customer point of view; we can sell it to banks which then use it to predict whether a customer pays or not.

## Financial Modelling:

Financial modeling is a technique that builds models that help to forecast a business's financial performance in the future. The forecast is based on assumptions about the future, and the company's historical performance, and requires the preparation of financial statements, the company's expenses, and earnings and may include their supporting schedules. Financial models are tools that are helpful in making business decisions. For: in IPO, whether or not to raise money. This type of financial model involves looking at comparable company analysis in conjunction with an assumption about how much the investors would be willing to pay for the company. The valuation in an IPO model includes "an IPO discount" to make sure the stock trades well in the secondary market. Often used by Investment bankers and corporate development professionals.

We are expecting a linear increase in sales for the next 3-4 years based on the situation that our sales might increase exponentially.

# CODE IMPLEMENTATION:

Initially, we imported all the necessary libraries required for the code implementation and read the CSV file containing the dataset.

### Importing Necessary Libraries

- NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- Pandas is a library written for the Python programming language for data manipulation and analysis
- Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays.
- Seaborn is an open-source Python library built on top of matplotlib. It is used for data visualization and exploratory data analysis.
- Warnings are provided to warn the developer of situations that aren't necessarily exceptions

In [3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

# Understanding the data

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

In [60]:
```
train = pd.read_csv(r"C:/Users/SWAYAM/Downloads/wf/train_data.csv")
train.head()
```

Out[60]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | 360.0 | 1.0 |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | 360.0 | 1.0 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | 360.0 | 1.0 |

In [61]:
```
test = pd.read_csv(r'C:\Users\SWAYAM\Downloads\wf\test.csv')
test.head()
```

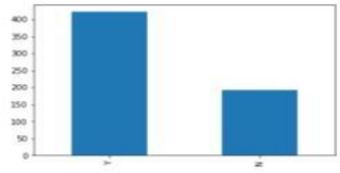
Out[61]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001015 | Male | Yes | 0 | Graduate | No | 5720 | 0 | 110.0 | 360.0 | 1.0 |
| 1 | LP001022 | Male | Yes | 1 | Graduate | No | 3076 | 1500 | 126.0 | 360.0 | 1.0 |
| 2 | LP001031 | Male | Yes | 2 | Graduate | No | 5000 | 1800 | 208.0 | 360.0 | 1.0 |
| 3 | LP001035 | Male | Yes | 2 | Graduate | No | 2340 | 2546 | 100.0 | 360.0 | NaN |
| 4 | LP001051 | Male | No | 0 | Not Graduate | No | 3276 | 0 | 78.0 | 360.0 | 1.0 |

## Missing value imputation

Let’s list out feature-wise count of missing values.

```
In [14]: df['Loan_Status'].value_counts(normalize=True)
Out[14]: Y    0.687296
         N    0.312704
         Name: Loan_Status, dtype: float64

In [15]: df['Loan_Status'].value_counts().plot.bar()
Out[15]: <AxesSubplot:>
```



```
In [16]: df.isnull().sum()
Out[16]: 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 can consider these methods to fill the missing values:

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

```
In [17]: 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)

In [18]: df['Loan_Amount_Term'].value_counts()
Out[18]: 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

In [20]: df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)
         df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)

In [21]: df.isnull().sum()
Out[21]: 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
```

# Project Building

We can remove ‘loan_ID’ as it does not have any effect on the prediction, and in fact, eliminating all the unwanted data would give a better and more optimised solution.

```
In [21]: df.isnull().sum()

Out[21]: 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

In [23]: df=df.drop('Loan_ID',axis=1)
```

We used scikit-learn (sklearn) for making different models which is an open-source library for Python. It is one of the most efficient tools which contains many inbuilt functions that can be used for modeling in Python. 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 a numerical value.

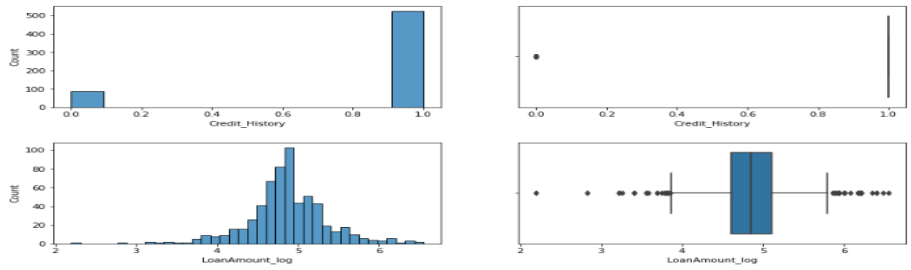
Here, we are visualizing the data of all the factors affecting Loan Eligibility.

```
In [69]: numerical=train.select_dtypes(exclude=["object"])
         numerical.head()
```

Out[69]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| 0 | 5849 | 0.0 | NaN | 360.0 | 1.0 |
| 1 | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 |
| 2 | 3000 | 0.0 | 66.0 | 360.0 | 1.0 |
| 3 | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 |
| 4 | 6000 | 0.0 | 141.0 | 360.0 | 1.0 |

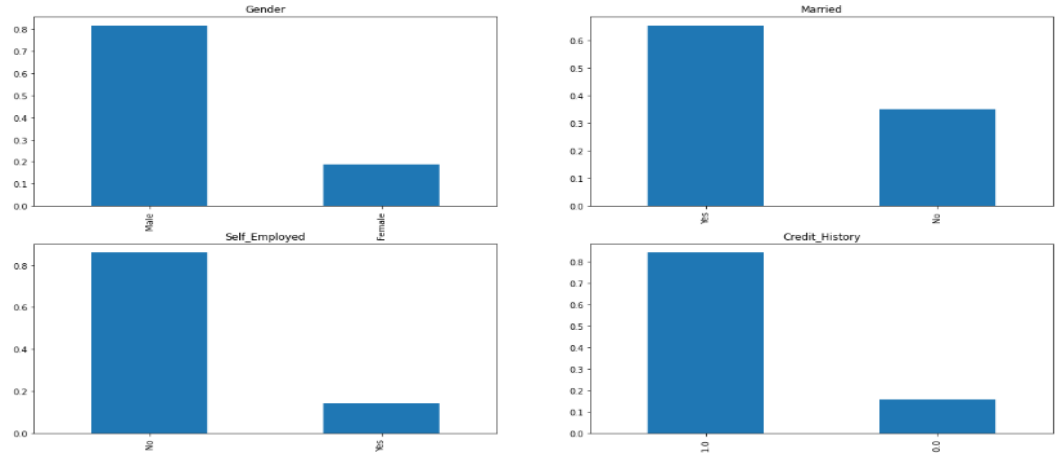
```
In [50]: def visualize_numeric_feature(data_frame,col_name):
             fig,ax=plt.subplots(1,2,figsize=(14,3))
             sns.histplot(data=data_frame,x=col_name,ax=ax[0])
             sns.boxplot(data=data_frame,x=col_name,ax=ax[1])
         plt.show()

In [53]: for i in numerical:
             visualize_numeric_feature(train,i)
```



Now, the factors of the data are dependent on each other. We have compared some of the factors and visualized them for better understanding.

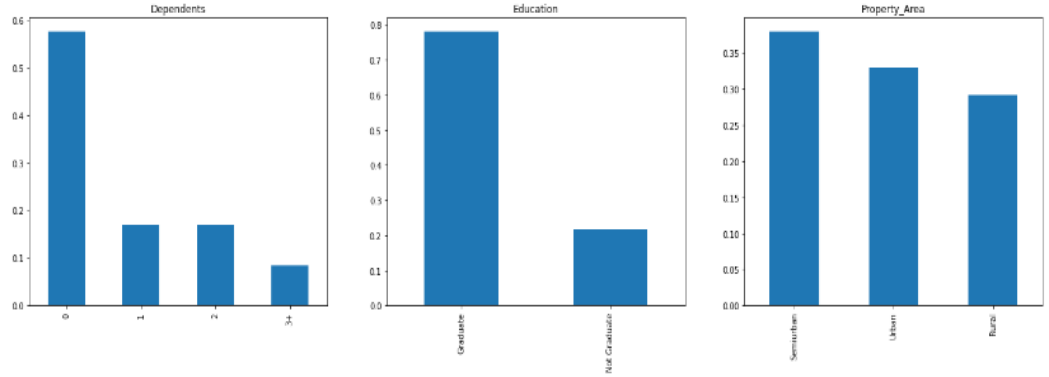
Here the factors affecting are Gender, Married, Self-employed, and Credit History.

```
In [62]: 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()
```




Here, we have visualized Dependents, Education, and Property Area factors.

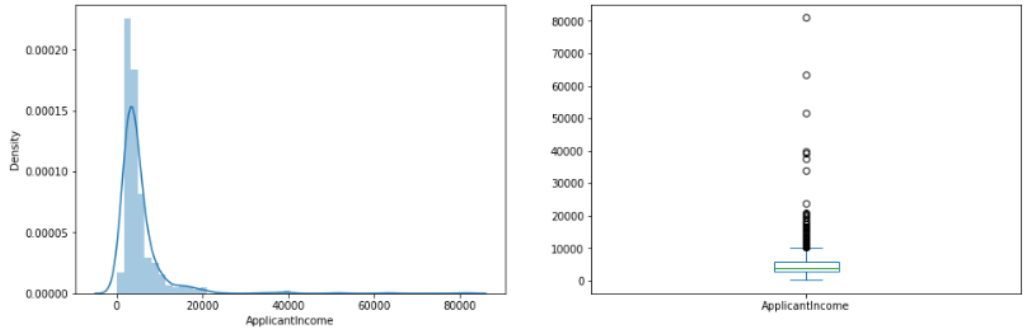
```
In [63]: plt.figure(1)
         plt.subplot(131)
         train['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6), title= 'Dependents')

         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()
```



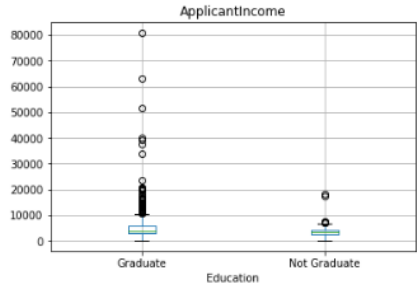
## Independent Variable (Numerical)

Till now we have seen the categorical and ordinal variables and now let's visualize the numerical variables. Let's look at the distribution of Applicant income first.

```
In [65]: plt.figure(1)
         plt.subplot(121)
         sns.distplot(train['ApplicantIncome']);
         plt.subplot(122)
         train['ApplicantIncome'].plot.box(figsize=(16,5))
         plt.show()
```




```
In [66]: train.boxplot(column='ApplicantIncome', by = 'Education')
         plt.suptitle("")
Out[66]: Text(0.5, 0.98, '')
```



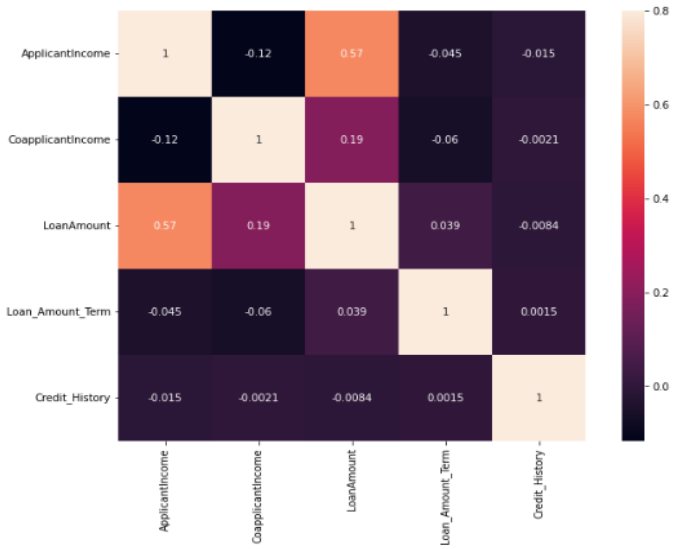

It can be inferred that most of the data in the distribution of applicant income are towards the left which means it is not normally distributed. We made it normal in later sections as algorithms work better if the data is normally distributed.

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

Now let’s look at the correlation between all the numerical variables. We used the heat map to visualize the correlation. Heatmaps visualize data through variations in coloring. The variables with darker colors mean their correlation is more.

```
In [67]: matrix = train.corr()
         f, ax = plt.subplots(figsize=(12,8))
         sns.heatmap(matrix,vmax=.8,square=True, annot = True)
```

```
Out[67]: <AxesSubplot:>
```

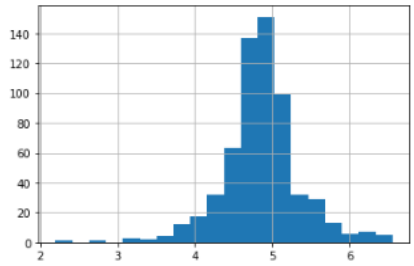



Due to these outliers,' the bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skewness. One way to remove the skewness is by doing the log transformation. As we took log transformation, it does not affect the smaller values much but reduces the larger values. So, we get a distribution similar to the normal distribution.

Let's visualize the effect of log transformation. We did similar changes to the text file simultaneously.

```
In [12]: test['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
         test['Married'].fillna(train['Married'].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)
```

```
In [13]: train['LoanAmount_log']=np.log(train['LoanAmount'])
         train['LoanAmount_log'].hist(bins=20)
         test['LoanAmount_log']=np.log(test['LoanAmount'])
```




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

# Model Building:

## Logistic Regression

Let us make our first model predict the target variable. We started with Logistic Regression which is used for predicting binary outcomes. The dataset has been divided into training and validation parts. Let us import Logistic Regression and fit the logistic regression model.

Let's drop the Loan_ID variable as it does not have any effect on the loan status. We did the same changes to the test dataset which we did for the training dataset. we made dummy variables for the categorical variables. The dummy variable turns categorical variables into a series of 0 and 1, making them a lot easier to quantify and compare.

```
In [14]: train=train.drop('Loan_ID',axis=1)
         test=test.drop('Loan_ID',axis=1)

In [15]: X = train.drop('Loan_Status',1)
         y = train.Loan_Status

In [16]: X = pd.get_dummies(X)
         train=pd.get_dummies(train)
         test=pd.get_dummies(test)
```

Now we trained the model on the training dataset and make predictions for the test dataset. One way of doing this can divide our train dataset into two parts: train and validation. We trained the model on this training part and used that to make predictions for the validation part. In this way, we can validate our predictions as we have the true predictions for the validation part.

We used the train_test_split function from sklearn to divide our train dataset.

```
In [17]: from sklearn.model_selection import train_test_split
         x_train, x_cv, y_train, y_cv = train_test_split(X,y, test_size=0.3)

In [18]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score
         model = LogisticRegression()
         model.fit(x_train, y_train)

Out[18]: LogisticRegression()

In [19]: pred_cv = model.predict(x_cv)
         accuracy_score(y_cv,pred_cv)

Out[19]: 0.745945945945946
```

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

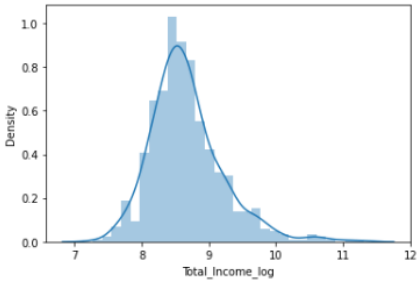
## Logistic Regression using stratified k-folds cross-validation

To check how robust our model is to unseen data, we used Validation. It is a technique that involves reserving a particular sample of a dataset on which we do not train the model. Later, we tested the model on this sample before finalizing it.

```
In [26]: from sklearn.model_selection import StratifiedKFold

In [27]: i=1
         mean = 0
         kf = StratifiedKFold(n_splits=5)
         for train_index,test_index in kf.split(X,y):
             print ('\n{} of kfold {} '.format(i,kf.n_splits))
             xtr,xvl = X.loc[train_index],X.loc[test_index]
             ytr,yvl = y[train_index],y[test_index]
             model = LogisticRegression(random_state=1)
             model.fit(xtr,ytr)
             pred_test=model.predict(xvl)
             score=accuracy_score(yvl,pred_test)
             mean += score
             print ('accuracy_score',score)
             i+=1
             pred_test = model.predict(test)
             pred = model.predict_proba(xvl)[:,1]
         print ('\n Mean Validation Accuracy',mean/(i-1))

         1 of kfold 5
         accuracy_score 0.8048780487804879

         2 of kfold 5
         accuracy_score 0.7642276422764228

         3 of kfold 5
         accuracy_score 0.7804878048780488

         4 of kfold 5
         accuracy_score 0.8455284552845529

         5 of kfold 5
         accuracy_score 0.8032786885245902

          Mean Validation Accuracy 0.7996801279488205
```

The mean validation accuracy for this model turns out to be 0.80.

```
In [28]: submission['Loan_Status']=pred_test
         submission['Loan_ID']=test_original['Loan_ID']

In [29]: submission['Loan_Status'].replace(0, 'N', inplace=True)
         submission['Loan_Status'].replace(1, 'Y', inplace=True)

In [30]: pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('Log1.csv')

In [31]: train['Total_Income']=train['ApplicantIncome']+train['CoapplicantIncome']
         test['Total_Income']=test['ApplicantIncome']+test['CoapplicantIncome']

In [32]: train['Total_Income_log'] = np.log(train['Total_Income'])
         sns.distplot(train['Total_Income_log'])
         test['Total_Income_log'] = np.log(test['Total_Income'])
```




The distribution looks much closer to normal and the effect of extreme values has significantly subsided.

## Decision Tree

The decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on the most significant splitter/differentiator in input variables.

Let's fit the decision tree model with 5 folds of cross-validation.

```
In [40]: from sklearn import tree
         i=1
         mean = 0
         kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
         for train_index,test_index in kf.split(X,y):
             print ('\n{} of kfold {} '.format(i,kf.n_splits))
             xtr,xvl = X.loc[train_index],X.loc[test_index]
             ytr,yvl = y[train_index],y[test_index]
             model = tree.DecisionTreeClassifier(random_state=1)
             model.fit(xtr,ytr)
             pred_test=model.predict(xvl)
             score=accuracy_score(yvl,pred_test)
             mean += score
             print ('accuracy_score',score)
             i+=1
             pred_test = model.predict(test)
             pred = model.predict_proba(xvl)[:,1]
         print ('\n Mean Validation Accuracy',mean/(i-1))

1 of kfold 5
accuracy_score 0.7073170731707317

2 of kfold 5
accuracy_score 0.7235772357723578

3 of kfold 5
accuracy_score 0.7073170731707317

4 of kfold 5
accuracy_score 0.7154471544715447

5 of kfold 5
accuracy_score 0.6885245901639344

 Mean Validation Accuracy 0.70843662534986
```

## Random Forest

Random Forest is a tree-based bootstrapping algorithm wherein a certain no. of weak learners (decision trees) is combined to make a powerful prediction model.

In [44]:
```
from sklearn.ensemble import RandomForestClassifier
i=1
mean = 0
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print ('\n{} of kfold {} '.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = RandomForestClassifier(random_state=1, max_depth=10)
    model.fit(xtr,ytr)
    pred_test=model.predict(xvl)
    score=accuracy_score(yvl,pred_test)
    mean += score
    print ('accuracy_score',score)
    i+=1
    pred_test = model.predict(test)
    pred = model.predict_proba(xvl)[:,1]
print ('\n Mean Validation Accuracy',mean/(i-1))
```

```
1 of kfold 5
accuracy_score 0.8130081300813008

2 of kfold 5
accuracy_score 0.8373983739837398

3 of kfold 5
accuracy_score 0.7804878048780488

4 of kfold 5
accuracy_score 0.8130081300813008

5 of kfold 5
accuracy_score 0.7622950819672131

 Mean Validation Accuracy 0.8012395041983206
```

We try to improve the accuracy by tuning the hyperparameters for this model. We used a grid search to get the optimized values of hyper parameters. Grid-search is a way to select the best of a family of hyper parameters, parametrized by a grid of parameters.

We tuned the max_depth and n_estimators parameters. max_depth decides the maximum depth of the tree and n_estimators decide the number of trees that will be used in the random forest model.

In [45]:
```
from sklearn.model_selection import GridSearchCV
paramgrid = {'max_depth': list(range(1,20,2)), 'n_estimators': list(range(1,200,20))}
grid_search=GridSearchCV(RandomForestClassifier(random_state=1),paramgrid)
```

In [46]:
```
from sklearn.model_selection import train_test_split
x_train, x_cv, y_train, y_cv = train_test_split(X,y, test_size=0.15, random_state=1)
grid_search.fit(x_train,y_train)
```

Out[46]:
```
GridSearchCV(estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
                         'n_estimators': [1, 21, 41, 61, 81, 101, 121, 141, 161,
                                          181]})
```

In [47]:
```
grid_search.best_estimator_
```

Out[47]:
```
RandomForestClassifier(max_depth=15, n_estimators=21, random_state=1)
```

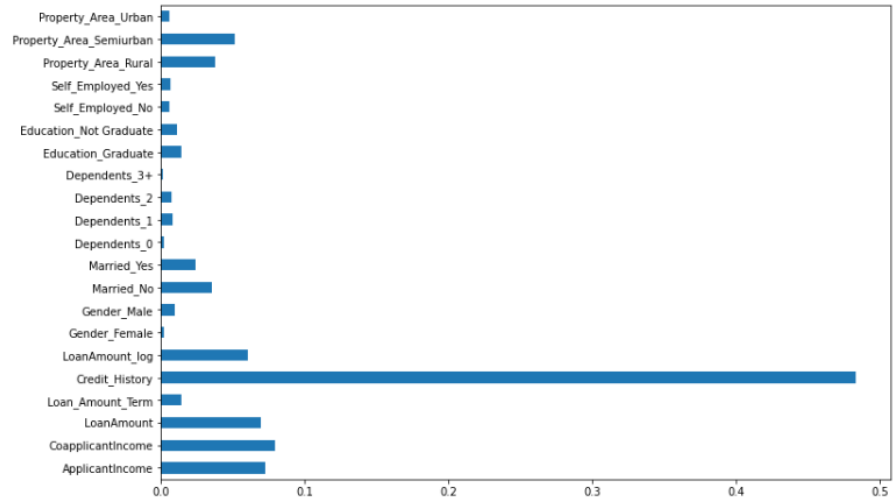
In [48]:
```
i=1
mean = 0
kf = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
for train_index,test_index in kf.split(X,y):
    print ('\n{} of kfold {} '.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = RandomForestClassifier(random_state=1, max_depth=3, n_estimators=41)
    model.fit(xtr,ytr)
    pred_test = model.predict(xvl)
    score = accuracy_score(yvl,pred_test)
    mean += score
    print ('accuracy_score',score)
    i+=1
    pred_test = model.predict(test)
    pred = model.predict_proba(xvl)[:,1]
print ('\n Mean Validation Accuracy',mean/(i-1))
```

```
1 of kfold 5
accuracy_score 0.8130081300813008

2 of kfold 5
accuracy_score 0.8211382113821138

3 of kfold 5
accuracy_score 0.7967479674796748

4 of kfold 5
accuracy_score 0.8130081300813008

5 of kfold 5
accuracy_score 0.7950819672131147

 Mean Validation Accuracy 0.807796881247501
```

## Comparing the Dependencies

To find the feature importance, i.e., which features are most important for this problem. We used the feature_importances_ attribute of sklearn to do so. We can see that Credit History is the most important feature followed by Balance Income, Total Income, and EMI. So, feature engineering helped us in predicting our target variable. Out of all the features, the feature 'Credit History' is the most important andthe prediction is more dependent on it.

```
In [96]: submission['Loan_Status']=pred_test
         submission['Loan_ID']=test_original['Loan_ID']

In [97]: submission['Loan_Status'].replace(0, 'N', inplace=True)
         submission['Loan_Status'].replace(1, 'Y', inplace=True)

In [98]: pd.DataFrame(submission, columns=['Loan_ID','Loan_Status']).to_csv('RandomForest.csv')

In [99]: importances=pd.Series(model.feature_importances_, index=X.columns)
         importances.plot(kind='barh', figsize=(12,8))

Out[99]: <AxesSubplot:>
```




## GitHub link for code:

https://github.com/sahil-2505/Feynn_Labs_Task-3.git

## References

https://towardsdatascience.com/predict-loan-eligibility-using-machine-learning-models-7a14ef904057

https://github.com/mridulrb/Predict-loan-eligibility-using-IBM-Watson-Studio

## Conclusion

This model is very useful in the banking sector for swift processing of loan eligibility for the applicants. After the Final Submission of test data, my accuracy score was 80%. Feature engineering helped me increase my accuracy. The predictive models based on Logistic Regression, Decision Tree, Random Forest, and Grid Search give the accuracy as 79.96%, 70.84%, 80.11%, and 80.77