Loan Application Status Prediction Using Machine Learning

Declaration

I declare that this dissertation entitled "Loan Approval Prediction Using Machine Learning Techniques" is the result of my own work and analysis.

Author - Deepak Kumar

Publish Date – April 2,2022

B-Tech in Information Technology

Email ID - swrrocky17@gmail.com

Contents:

1. Abstract

Loan is the essential product of banks and other financial institutions. As a big number of people go to banks to borrow money for different activities, the number of customers have increased and some banks expect to earn a lot of money as a result of interest paid on loans. However, loans are associated with risk of defaulting, i.e. the possibility that some borrowers may not be able to pay back their loans. Thus, high levels of non-performing loans can be a source of instability of the banking sector and lead to bankruptcy. One of the important steps for banks to decide if a loan has to be authorized is to ensure that the candidate to borrow has the capacity of paying back the loan in the proposed terms. The advancement of technology like machine learning, computer science and other science is playing an important role by supporting banks to predict the probability of defaulting for a given customer based on his past behavior. With past behavior, it means the historical record of the customers which banks keep with themselves.

In my research, I have taken the data of an unknown bank to predict the loan status of an applicant. I would recommend financial institutions to use machine learning techniques because it saves money and time for both sides. The finding shows that Credit History is the most important feature while checking the approval of Loan status. Credit History includes information on your present and previous credit accounts such as loans, mortgages, credit cards, payment history, account balances and the duration of each account being active or open. If the credit History is good, then it is 1 or else it is 0.

Acquiring loans for different purposes 2 such as the home loan, education loan, car loan, business loans etc., has become part of our day-to-day life from different financial institutions like credit unions and banks. However, many people are unable to determine the total amount of credit that they can afford to pay back. Analysis of creditworthiness is one of the most important for banks and other financial institutions to stay working in the highly competitive market and for their profitability. They must set clear and defined criteria for lending. These criteria must be sufficient and adequate to provide the required information about the structure of credit, borrowers, and mode of payment.

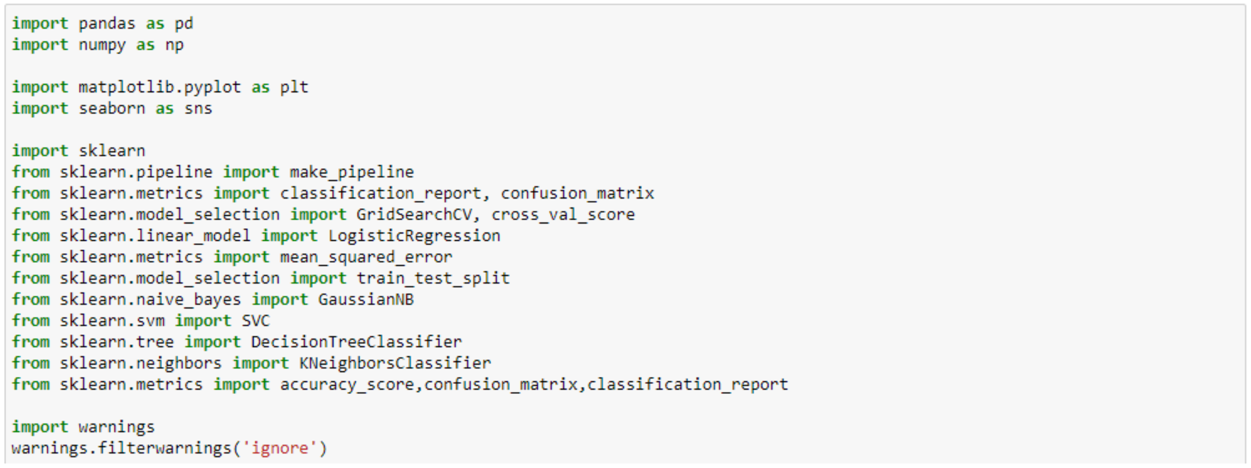
2. Data Description

The machine learning model is trained using the training data set. Every new applicant details filled at the time of application form acts as a test data set. On the basis of the training data sets, the model will predict whether a loan would be approved or not. We have 13 features in total out of which we have 12 independent variables and 1 dependent variable i.e. Loan_Status in train dataset and 12 independent variables in test dataset. The Loan_ID, Gender, Married, Education, Self_Employed, Property_Area, Loan_Status are all categorical. Below is the brief description of all data.

- **Gender** - It is nothing but the sex of the applicant
- **Married** - It gives us information if the applicant is married, single or divorced
- **Dependents** - While applying for the Loan, Banks check how many dependents are there with the applicant.
- **Education** - It describes how much educated is the applicant.
- **Self_Employed** - It gives us information if the applicant is self-employed or does any job.
- **Applicant Income** - It is the income of the applicant applying the loan.
- **Co-applicant Income** - It is the income of the co-applicant applying the loan. Generally, all bank checks the total income of the family applying the loan.
- **Loan Amount** - It is the amount of the loan sanctioned to the applicant.
- **Loan Amount Term** - It is the amount of the loan issues for the number of the months(In Short Loan period)
- **Credit History** - Credit History includes information on your present and previous credit accounts such as loans, mortgages, credit cards, payment history, account balances and the duration of each account being active or open. If the credit History is good, then it is 1 or else it is 0.
- **Property Area** - In which area does the property belongs, also is one of the factor for the bank to issue loan

3.Data Preparation

3.1 Importing modules



```python
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

import warnings
warnings.filterwarnings('ignore')
```

Python Libraries
The models are implemented using Python 3.7 with listed libraries:

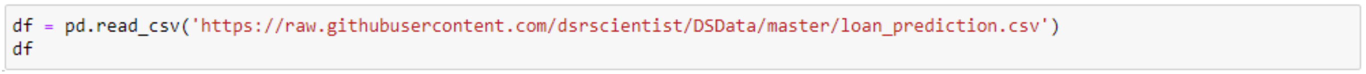
Pandas is a Python package to work with structured and time series data. The data from various file formats such as csv, json, sql etc can be imported using Pandas. It is a powerful open source tool used for data analysis and data manipulation operations such as data cleaning, merging, selecting as well wrangling, whereas the **NumPy** module works with the numerical data.

Matplotlib is well connected with Numpy and Pandas and acts as a graphics package for data visualization in python. Pyplot provides similar features and syntax as in MATLAB. Therefore, MATLAB users can easily study it.

Seaborn is a python library for building graphs to visualize data. It provides integration with pandas. This open source tool helps in defining the data by mapping the data on the informative and interactive plots. Each element of the plots gives meaningful information about the data

Sklearn python library is helpful for building machine learning and statistical models such as clustering, classification, regression etc. Though it can be used for reading, manipulating and summarizing the data as well, better libraries are there to perform these functions.

Import the data from **Github** location.



```python
df = pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/loan_prediction.csv')
df
```

The data in Github is in csv format, we are using Dataframe to read the csv file.

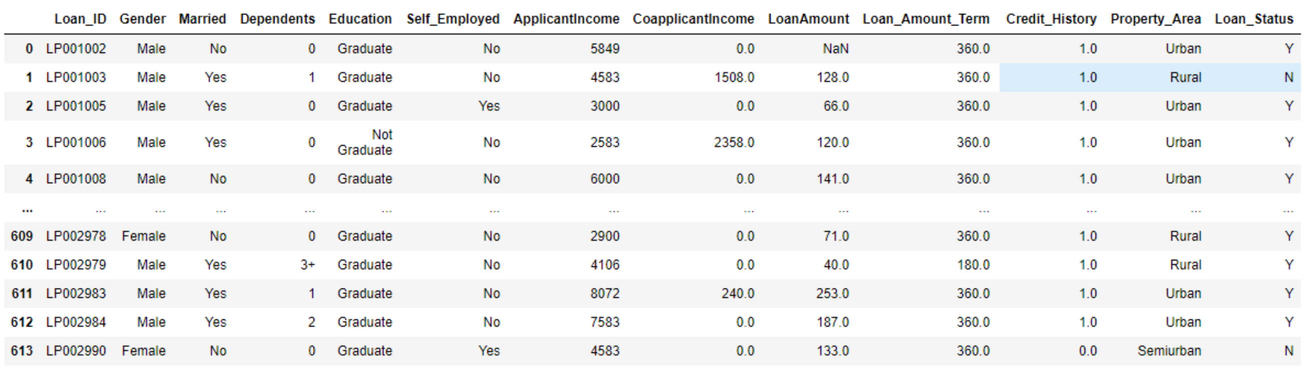
pandas.DataFrame

class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)[source]

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

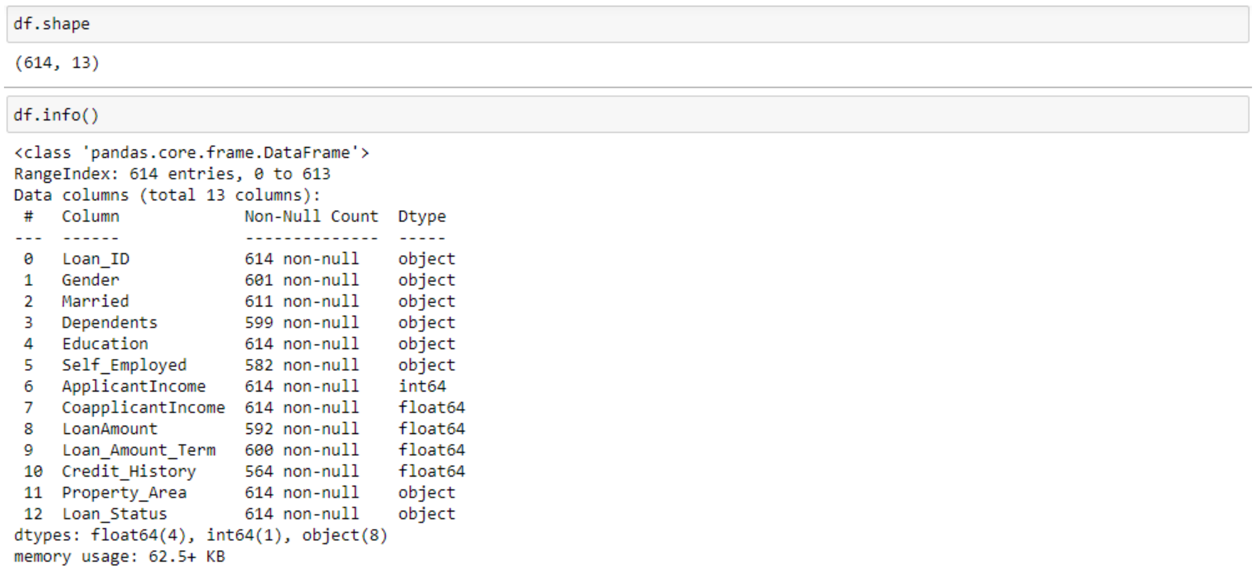
Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Output.



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

Shape and data info



```
df.shape
```

```
(614, 13)
```

```
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
```

3.2 Handle null values.

```
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
```

We see null values in Gender, Married, Dependents, Self_Employed, Loan_Amount, Loan_Amount_Term, Credit_History.

- We know that imputing missing data with **mean** values can only be done with **numerical** data.
- Another technique is **median** imputation in which the missing values are replaced with the median value of the entire feature column. When the data is skewed, it is good to consider using the median value for replacing the missing values. We must note that imputing missing data with median value can only be done with **numerical** data.
- Yet another technique is **mode** imputation in which the missing values are replaced with the mode value or most frequent value of the entire feature column. When the data is skewed, it is good to consider using mode values for replacing the missing values. We must note that imputing missing data with mode values can be done with **numerical** and **categorical** data.

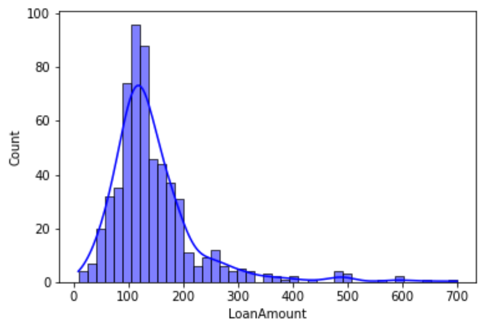
We see that Gender,Married,Dependents and Self_Employed are categorical data,hence, it is good to consider **mode** values for replacing the missing values

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

Now we have missing values in LoanAmount,Loan_Amount_Term and Credit_History.

```
sns.histplot(df.LoanAmount,color="blue",kde=True,stat="count")
```

```
<AxesSubplot:xlabel='LoanAmount', ylabel='Count'>
```

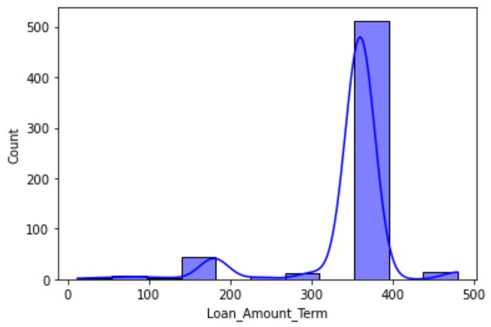


We see that it is skewed right and is not normally distributed and also there are not too much data which is missing, hence we will fill it using median

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

```
sns.histplot(df.Loan_Amount_Term,color="blue",kde=True,stat="count")
```

```
<AxesSubplot:xlabel='Loan_Amount_Term', ylabel='Count'>
```



The Loan Amount Term is left skewed and is not normally distributed, hence we will fill this also with median

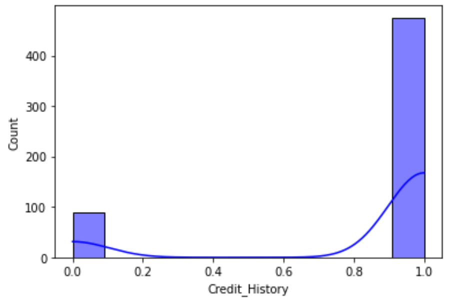
```
df['Loan_Amount_Term']=df['Loan_Amount_Term'].fillna(df['LoanAmount'].median())
```

In [18]:
```
print('Count of Credit History')
creditHistory_counts = df['Credit_History'].value_counts()
creditHistory_counts
```

```
Count of Credit History
```

Out[18]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

```
In [19]: sns.histplot(df.Credit_History,color="blue",kde=True,stat="count")
Out[19]: <AxesSubplot:xlabel='Credit_History', ylabel='Count'>
```



Now check for null values.

```
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
```

We don’t see any null values now.

3.3 Delete duplicates

In data cleaning we check for duplicate and also if few columns are not required, we will drop them.

Check for Duplicates

```
In [22]: duplicate = df[df.duplicated(keep = 'last')]
         print("Duplicate Rows :")

         # Print the resultant Dataframe
         duplicate

Duplicate Rows :
Out[22]:
```

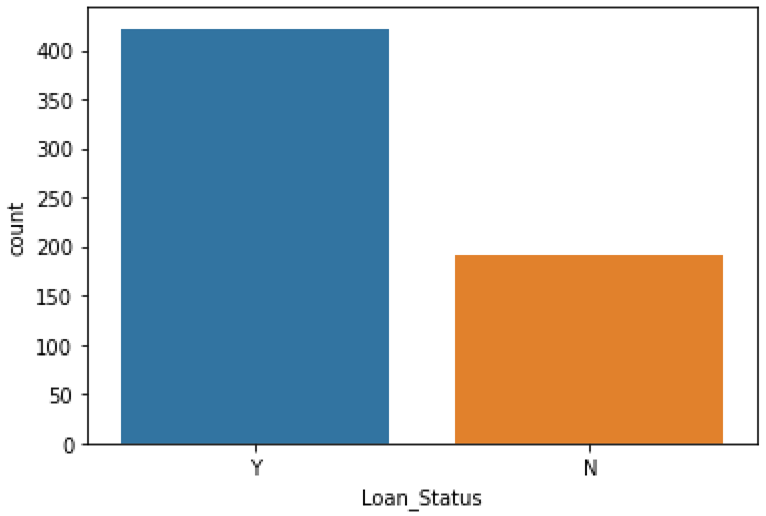
Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pr

There is not a single Duplicate row, hence we are good till this point

3.4 Visualization and Data cleaning.

Virtualization is nothing but graphic representation of data that is used to find useful information.

```
In [23]: sns.countplot(df['Loan_Status'])
Out[23]: <AxesSubplot:xlabel='Loan_Status', ylabel='count'>
```



```
In [24]: print('Loan Status')
         loanStatus = df['Loan_Status'].value_counts()
         loanStatus

         Loan Status
Out[24]: Y    422
         N    192
         Name: Loan_Status, dtype: int64
```

The dataset is a bit imbalanced, we will be using techniques like SMOTE to handle such datasets, just to increase the minority classes in the training dataset

```
In [26]: categorical_columns=[x for x in df.dtypes.index if df.dtypes[x] !='object']
         categorical_columns
Out[26]: ['ApplicantIncome',
          'CoapplicantIncome',
          'LoanAmount',
          'Loan_Amount_Term',
          'Credit_History']
```

LabelEncoder to convert the object Datatypes

```
In [27]: import re
         updated_LoanStatus = {"Y": 1, "N": 0}
         data = [df]

         for dataset in data:
             dataset['Loan_Status'] = dataset['Loan_Status'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).group())
             dataset['Loan_Status'] = dataset['Loan_Status'].map(updated_LoanStatus)
             dataset['Loan_Status'] = dataset['Loan_Status'].astype(int)
```

When we check, we see that Loan_status is converted into 1 and 0, where 1 means Y and 0 means N

```
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	360.0	1.0	Urban	1
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	1
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	1
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	1
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	1
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	1
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	0

614 rows × 13 columns

Lets create a new column LoanStatus_Rejected to anlayse the data

```
df['LoanStatus_Rejected'] = 1 - df['Loan_Status']
df.head(2)
```

	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	128.0	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0

```
df['Gender'].value_counts()
```

```
Male      502
Female    112
Name: Gender, dtype: int64
```

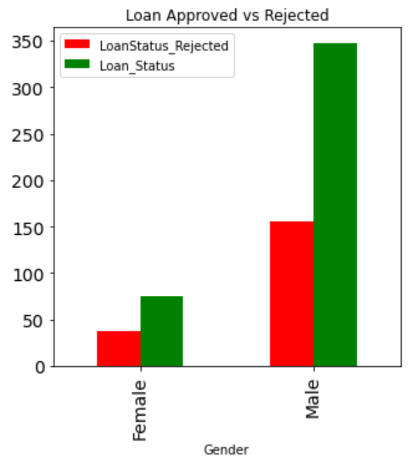
```
#Create a Total column to sum up rejected and passed.
df['Total']=df['LoanStatus_Rejected']+df['Loan_Status']
```

```
df.groupby('Gender').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

	LoanStatus_Rejected	Loan_Status	Total
Gender			
Female	37	75	112
Male	155	347	502

```
df.groupby('Gender').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",
                                                                           stacked=False,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Gender'>
```

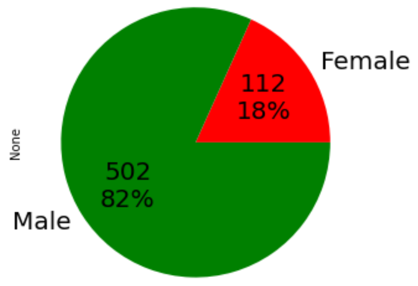


Let’s do the further analysis on Gender column using Pie chart. For that we defined a function which we will use at multiple areas.

```
def label_function(val):
    return f'{val / 100 * len(df):.0f}\n{val:.0f}%'
```

```
df.groupby('Gender').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},figsize=(5,5),
                                 colors=['red', 'green'])
```

```
<AxesSubplot:ylabel='None'>
```



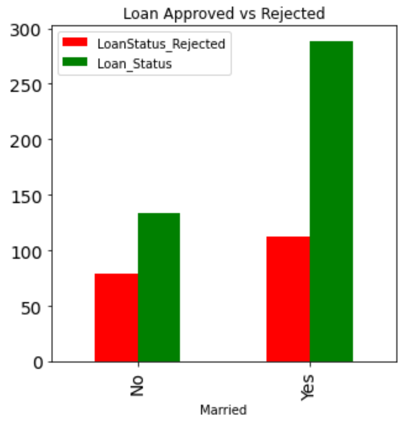
82% of the applicant are male and 18% are Female, and out of this 33% of the Loan got rejected for Female applicant and 30% of the Loan got rejected for Male applicant

```
df.groupby('Married').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

	LoanStatus_Rejected	Loan_Status	Total
Married			
No	79	134	213
Yes	113	288	401

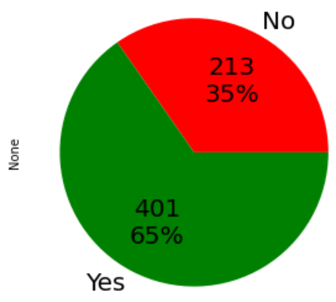
```
: df.groupby('Married').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",stacked=Fal
                                                                    ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
: <AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Married'>
```



```
df.groupby('Married').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},figsize=(5,5),
                                        colors=['red', 'green'])
```

```
<AxesSubplot:ylabel='None'>
```



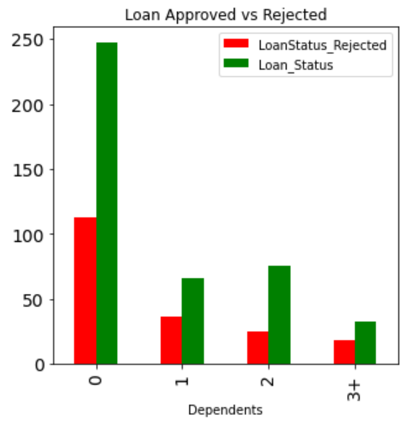
65% of married applicants have applied for the loan and 72% got their loan sanctioned successfully

```
df.groupby('Dependents').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

	LoanStatus_Rejected	Loan_Status	Total
Dependents			
0	113	247	360
1	36	66	102
2	25	76	101
3+	18	33	51

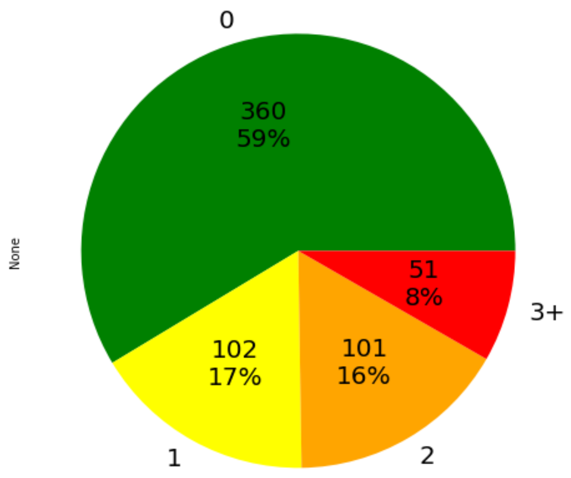
```
df.groupby('Dependents').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",stacked=
                                                    ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Dependents'>
```



```
df.groupby('Dependents').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},figsize=(8,8),
                               colors=['green', 'yellow','orange','red'])
```

```
<AxesSubplot:ylabel='None'>
```



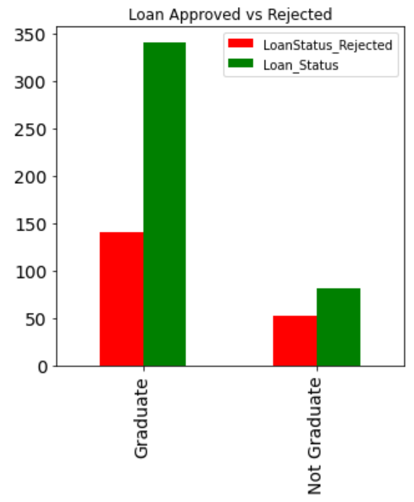
59% of the applicants have no dependents and out of this 59%,68.6% of the them got their Loan sanctioned.

```
df.groupby('Education').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

	LoanStatus_Rejected	Loan_Status	Total
Education			
Graduate	140	340	480
Not Graduate	52	82	134

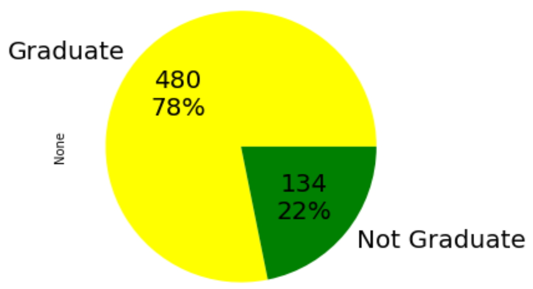
```
df.groupby('Education').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",stacked=F
                                                        ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Education'>
```



```
df.groupby('Education').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},figsize=(6, 5),
                                 colors=['yellow', 'green'])
```

```
<AxesSubplot:ylabel='None'>
```



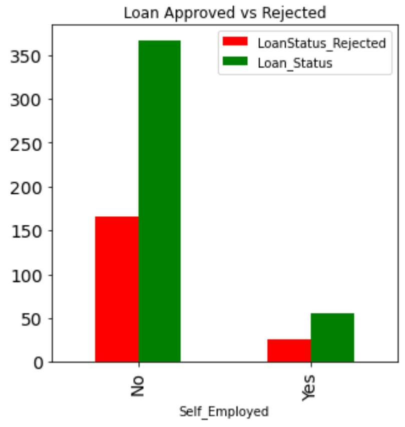
78% of the Loan got sanctioned for the applicants who were graduate and Loan got sanctioned for 71% of the Graduate applicants

```
df.groupby('Self_Employed').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']]
```

	LoanStatus_Rejected	Loan_Status
Self_Employed		
No	166	366
Yes	26	56

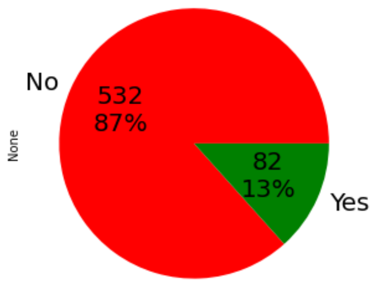
```
df.groupby('Self_Employed').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",stack
                                                        ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Self_Employed'>
```



```
df.groupby('Self_Employed').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},figsize=(5,5),
                                  colors=['red', 'green'])
```

```
<AxesSubplot:ylabel='None'>
```



Only 13% of the applicants are self- employed, and out this 13% applied for Loan, 54% Loan got sanctioned

```
df.groupby('ApplicantIncome').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

	LoanStatus_Rejected	Loan_Status	Total
ApplicantIncome			
150	1	0	1
210	0	1	1
416	1	0	1
645	0	1	1
674	0	1	1
...	...	...	...
39147	0	1	1
39999	0	1	1
51763	0	1	1
63337	0	1	1
81000	1	0	1

505 rows × 3 columns

Most of the Applicant Income lies between 2877 to 5795, and co-applicant income is between 0 to 2297, probably most of them are housewife's. There are few outliers are in the income of Applicant and co-applicant.

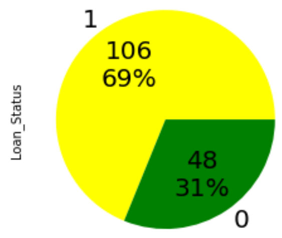
Now we are dividing the Applicant Income in the range of 150 to 2878 as low income, 2878 to 3813 as medium Income group, 3813 to 5795 as High Income Group and more than 5795 as Rich for further data analysis.

```
lowIncomeGroup_DF=df[(df['ApplicantIncome'] >=150) & (df['ApplicantIncome'] <2878)]
lowIncomeGroup_DF.shape
```

```
(154, 15)
```

```
def lowIncomeGroup(val):
    return f'{val / 100 * len(lowIncomeGroup_DF):.0f}\n{val:.0f}%'

lowIncomeGroup_DF['Loan_Status'].value_counts().plot(kind='pie', autopct=lowIncomeGroup, textprops={'fontsize': 20},
                                                     colors=['yellow', 'green'])
```

```
<AxesSubplot:ylabel='Loan_Status'>
```



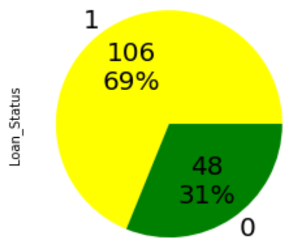
Out of 154, 106 got their Loan sanctioned and 48 applications were rejected

```
mediumIncomeGroup=df[(df['ApplicantIncome'] >=2878) & (df['ApplicantIncome'] <=3813)]
mediumIncomeGroup.shape
```

```
(154, 15)
```

```
def medIncomeGroup(val):
    return f'{val / 100 * len(mediumIncomeGroup):.0f}\n{val:.0f}%'

mediumIncomeGroup['Loan_Status'].value_counts().plot(kind='pie', autopct=medIncomeGroup, textprops={'fontsize': 20},
                                    colors=['yellow', 'green'])
```

```
<AxesSubplot:ylabel='Loan_Status'>
```



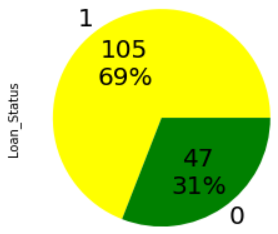
Out of 154 in medium earning range, 106 got their Loan sanctioned and 48 applications were rejected.

```
highGroup_DF=df[(df['ApplicantIncome'] >3813) & (df['ApplicantIncome'] <=5797)]
highGroup_DF.shape
```

```
(152, 15)
```

```
def highIncomeGroup(val):
    return f'{val / 100 * len(highGroup_DF):.0f}\n{val:.0f}%'

highGroup_DF['Loan_Status'].value_counts().plot(kind='pie', autopct=highIncomeGroup, textprops={'fontsize': 20},
                                colors=['yellow', 'green'])
```

```
<AxesSubplot:ylabel='Loan_Status'>
```



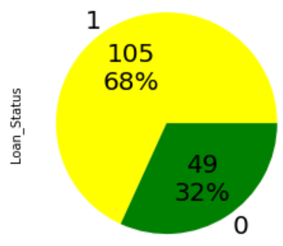
Out of 152 in High Income group, 105 got their Loan sanctioned and 47 applications were rejected

```
rich_DF=df[(df['ApplicantIncome'] >5795)]
rich_DF.shape
```

```
(154, 15)
```

```
def rich(val):
    return f'{val / 100 * len(rich_DF):.0f}\n{val:.0f}%'

rich_DF['Loan_Status'].value_counts().plot(kind='pie', autopct=rich, textprops={'fontsize': 20},
                                 colors=['yellow', 'green'])
```

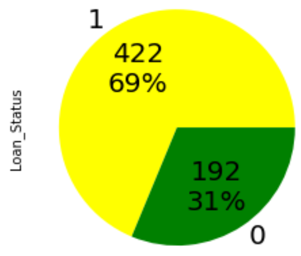
```
<AxesSubplot:ylabel='Loan_Status'>
```



Out of 154 in the Rich category, 105 got their Loan sanctioned and 49 applications were rejected

```
df['Loan_Status'].value_counts().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},
                                 colors=['yellow', 'green'])
```

```
<AxesSubplot:ylabel='Loan_Status'>
```



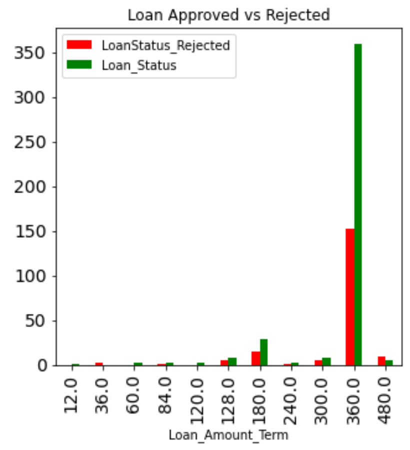
If we see from the above graphs, 69% of the people got their loan sanctioned and 31% applications were rejected.

```
df.groupby('Loan_Amount_Term').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

Loan_Amount_Term	LoanStatus_Rejected	Loan_Status	Total
12.0	0	1	1
36.0	2	0	2
60.0	0	2	2
84.0	1	3	4
120.0	0	3	3
128.0	6	8	14
180.0	15	29	44
240.0	1	3	4
300.0	5	8	13
360.0	153	359	512
480.0	9	6	15

```
df.groupby('Loan_Amount_Term').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",st
                                                         ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Loan_Amount_Term'>
```



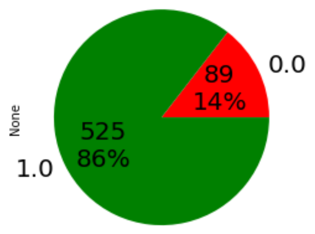
Maximum people applied for Loan term of 360 months i.e. 30 years

```
df.groupby('Credit_History').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

Credit_History	LoanStatus_Rejected	Loan_Status	Total
0.0	82	7	89
1.0	110	415	525

```
df.groupby('Credit_History').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},
                                        colors=['red', 'green'])
```

```
<AxesSubplot:ylabel='None'>
```



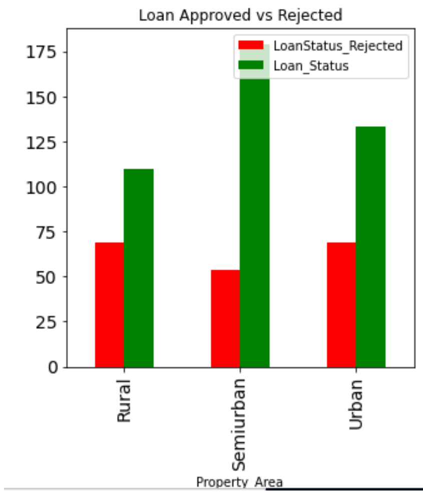
We see that 79% of the Loan got sanctioned for the applications whose Credit Histroy is 1 and 86% of the applications belongs to the those applicants whose Credit Histroy is 1

```
df.groupby('Property_Area').agg('sum')[['LoanStatus_Rejected', 'Loan_Status','Total']]
```

Property_Area	LoanStatus_Rejected	Loan_Status	Total
Rural	69	110	179
Semiurban	54	179	233
Urban	69	133	202

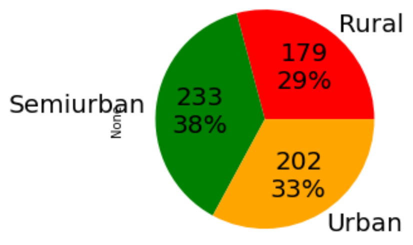
```
df.groupby('Property_Area').agg('sum')[['LoanStatus_Rejected', 'Loan_Status']].plot(kind='bar', figsize=(5,5),fontsize="14",stack
                                                                          ,color=['red', 'green'],title="Loan Approved vs Rejected")
```

```
<AxesSubplot:title={'center':'Loan Approved vs Rejected'}, xlabel='Property_Area'>
```



```
df.groupby('Property_Area').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},
                                       colors=['red', 'green','orange'])
```

```
<AxesSubplot:ylabel='None'>
```



The Property area is divided into three parts, Rural,Urban and Semiurban.

Semiurban had most of the loan sanctioned followed by Urban and Rural. 76% of the loan got sanctioned for semiurban area,65% of loan got sanctioned for urban area and 61% of the loan got sanctioned for Rural area

Drop Columns

Loan_ID is the unique ID, this is basically the Loan number of each applicant, whether rejected or accepted. This will not play any significant role in prediction, Hence it is safe column for us to delete.

Also, we had created LoanStatus_Rejected and Total columns which will no longer be needed, Hence it is safe to delete too.

```
df.drop(columns=["Loan_ID","LoanStatus_Rejected","Total"],inplace=True)
```

```
df.head(2)
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Ar
0	Male	No	0	Graduate	No	5849	0.0	128.0	360.0	1.0	Urt
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Ru

So we see that the columns Loan_ID,LoanStatus_Rejected and Total got successfully dropped

3.5 Data Exploration and Transformation

Skewness

```
df_features=df.drop(columns=['Loan_Status'])
```

```
#Check Data Skewness
skew_df=pd.DataFrame({'Skewness':df_features.skew()})
skew_df
```

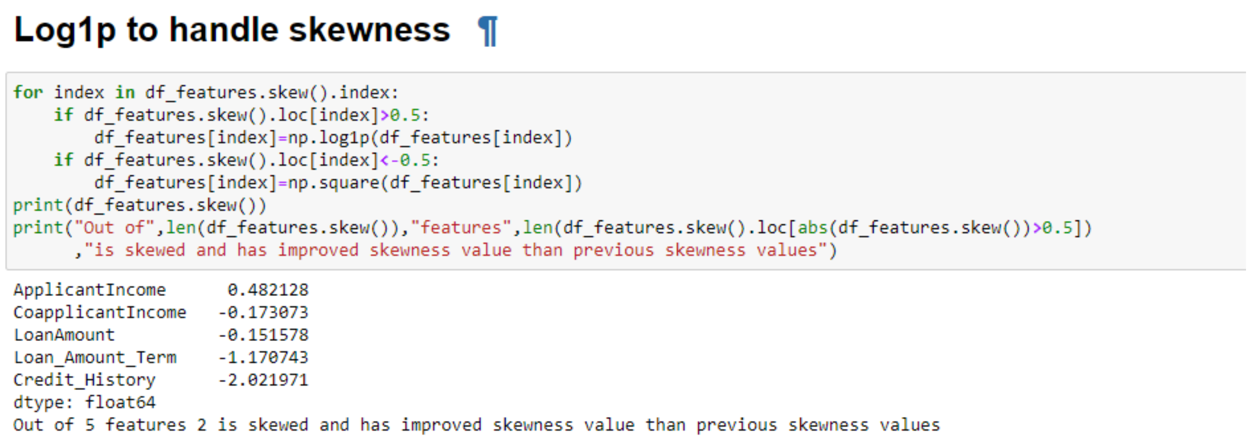
	Skewness
ApplicantIncome	6.539513
CoapplicantIncome	7.491531
LoanAmount	2.743053
Loan_Amount_Term	-2.116966
Credit_History	-2.021971

```
print("Out of",len(df_features.skew()),"features",len(df_features.skew().loc[abs(df_features.skew())>0.5]) ,"are skewed")
```

```
Out of 5 features 5 are skewed
```

Log Transform

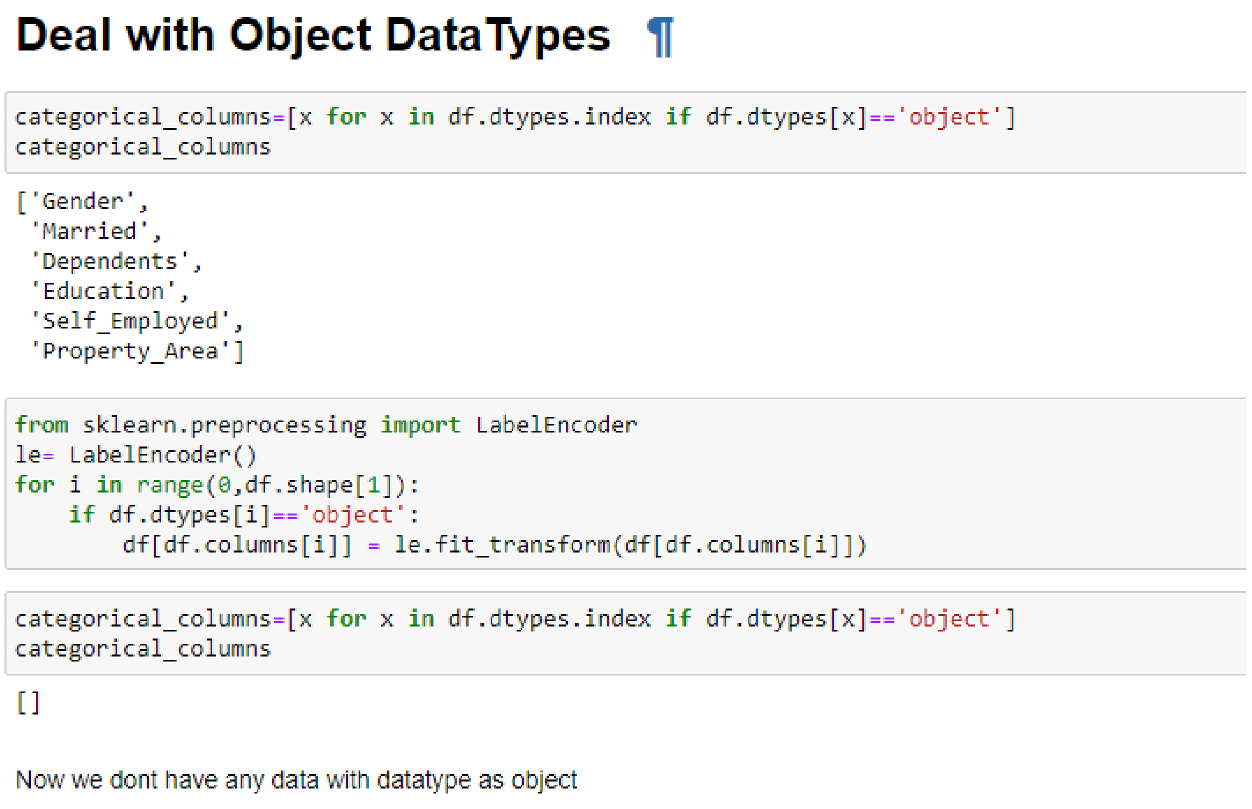
Log transformation is most likely the first thing you should do to remove skewness from the predictor. It can be easily done via *Numpy*, just by calling the **log()** function on the desired column. We can then just as easily check for skew:



Log1p to handle skewness ¶

```
for index in df_features.skew().index:
    if df_features.skew().loc[index]>0.5:
        df_features[index]=np.log1p(df_features[index])
    if df_features.skew().loc[index]<-0.5:
        df_features[index]=np.square(df_features[index])
print(df_features.skew())
print("Out of",len(df_features.skew()),"features",len(df_features.skew().loc[abs(df_features.skew())>0.5])
      ,"is skewed and has improved skewness value than previous skewness values")
```

```
ApplicantIncome       0.482128
CoapplicantIncome    -0.173073
LoanAmount           -0.151578
Loan_Amount_Term     -1.170743
Credit_History       -2.021971
dtype: float64
Out of 5 features 2 is skewed and has improved skewness value than previous skewness values
```



Deal with Object DataTypes ¶

```
categorical_columns=[x for x in df.dtypes.index if df.dtypes[x]=='object']
categorical_columns
```

```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area']
```

```
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
for i in range(0,df.shape[1]):
    if df.dtypes[i]=='object':
        df[df.columns[i]] = le.fit_transform(df[df.columns[i]])
```

```
categorical_columns=[x for x in df.dtypes.index if df.dtypes[x]=='object']
categorical_columns
```

```
[]
```

Now we dont have any data with datatype as object

```
df.info()
```

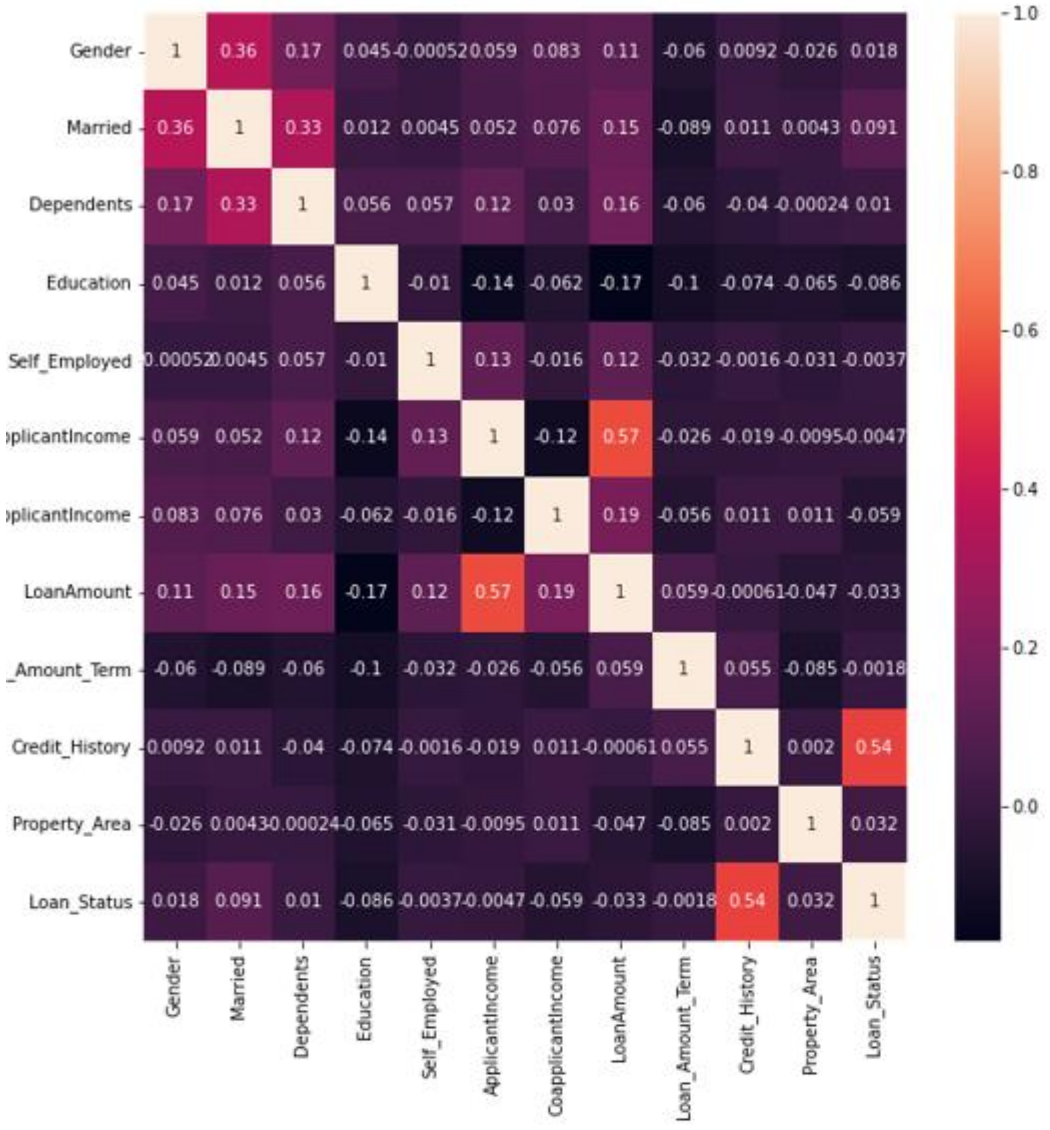
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             614 non-null    int32  
 1   Married            614 non-null    int32  
 2   Dependents         614 non-null    int32  
 3   Education          614 non-null    int32  
 4   Self_Employed      614 non-null    int32  
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         614 non-null    float64
 8   Loan_Amount_Term   614 non-null    float64
 9   Credit_History     614 non-null    float64
 10  Property_Area      614 non-null    int32  
 11  Loan_Status        614 non-null    int32  
dtypes: float64(4), int32(7), int64(1)
memory usage: 40.9 KB
```

We have successfully converted all object types into int32 and now we can proceed to find Outliers and Skewness in our data.

HeatMap

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
```

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information. Heat maps are used in many areas such as defense, marketing and understanding consumer behavior.



We notice that

- Applicant Income and Loan Amount are bit correlated.
- Credit History and Loan_Status are correlated to some extent

Outliers

An **outlier** is an object that deviates significantly from the rest of the objects. They can be caused by measurement or execution error. The analysis of outlier data is referred to as outlier analysis or outlier mining.

Why outlier analysis?

Most data mining methods discard outliers noise or exceptions, however, in some applications such as fraud detection, the rare events can be more interesting than the more regularly occurring one and hence, the outlier analysis becomes important in such case.

Outlier Checking ¶

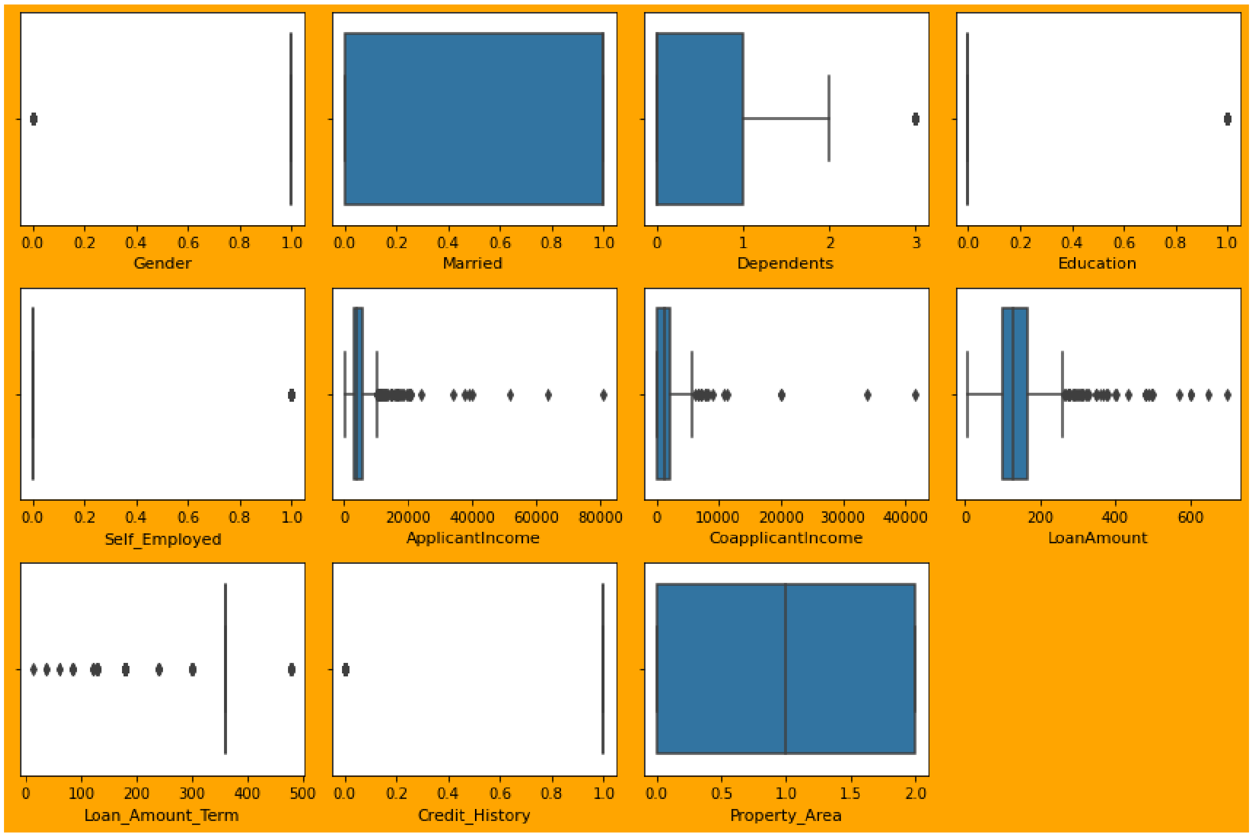
```
dataToPlot=df.loc[:, df.columns != 'Loan_Status']
dataToPlot.head(2)
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Ar
0	1	0	0	0	0	5849	0.0	128.0	360.0	1.0	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	

```
targetToPlot=df.loc[:, df.columns == 'Loan_Status']
targetToPlot.head(2)
```

	Loan_Status
0	1
1	0

```
#Lets see data distribution now using Boxplot
plt.figure(figsize=(12,8),facecolor='orange')
graph=1

for column in dataToPlot:

    if graph<=12:
        ax=plt.subplot(3,4,graph)
        sns.boxplot(dataToPlot[column])
        plt.xlabel(column,fontsize=11)

    graph+=1

plt.tight_layout()
```



We Applicant Income, Co-applicant Income, Loan Amount have outliers. We will use zscore method to check if we can remove the outliers or not.

```
from scipy.stats import zscore
z=np.abs(zscore(df))
z.shape
```

```
(614, 12)
```

```
threshold=3
print(np.where(z>3))
```

```
(array([  9,  14,  68,  94, 126, 130, 133, 155, 155, 171, 171, 177, 177,
       183, 185, 242, 262, 278, 308, 313, 333, 333, 369, 402, 409, 417,
       432, 443, 487, 495, 497, 506, 523, 525, 546, 561, 575, 581, 585,
       600, 604], dtype=int64), array([6, 8, 8, 8, 5, 7, 8, 5, 7, 5, 7, 6, 7, 5, 5, 8, 8, 7, 7, 8, 5, 7,
       7, 6, 5, 6, 7, 5, 7, 8, 8, 7, 7, 7, 8, 7, 8, 6, 8, 6, 7],
      dtype=int64))
```

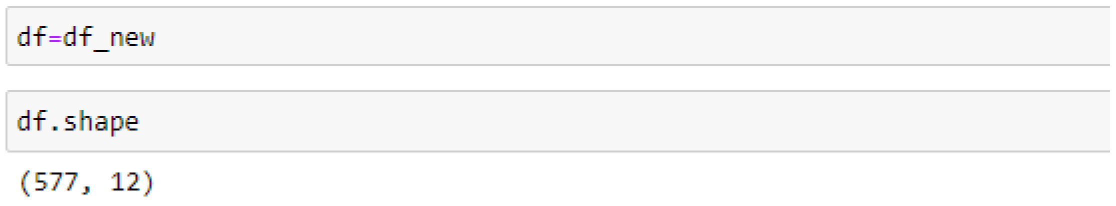
```
df_new=df[(z<3).all(axis=1)]
print(df.shape)
print(df_new.shape)
```

```
(614, 12)
(577, 12)
```

```
loss_percentage= ((614-577)/614)*100
loss_percentage
```

```
6.026058631921824
```

Loss Percentage is 6%, hence we will delete those data.



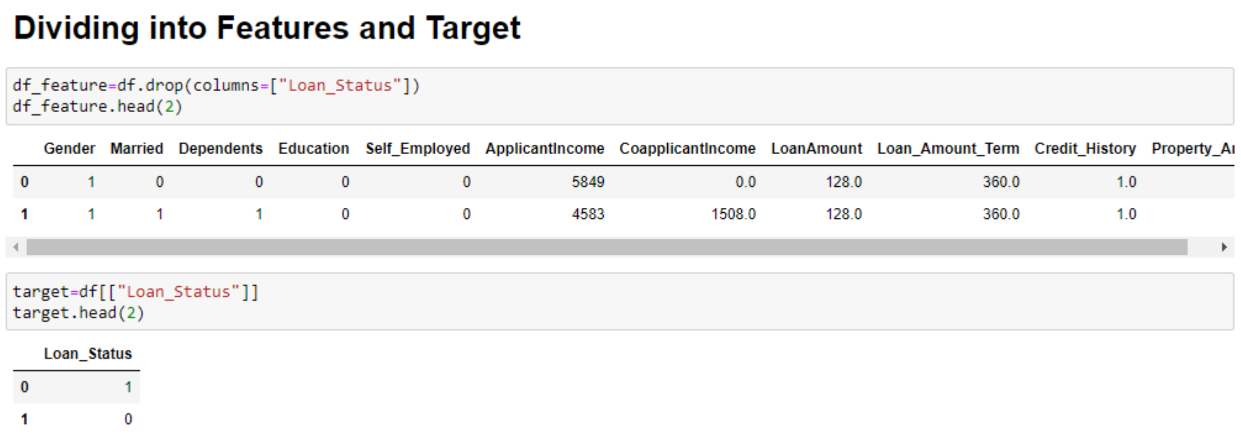
```
df=df_new
```

```
df.shape
```

```
(577, 12)
```

3.6 Split Dataset.

We will perform split dataset into features and target for train and test purpose.



Dividing into Features and Target

```
df_feature=df.drop(columns=["Loan_Status"])
df_feature.head(2)
```

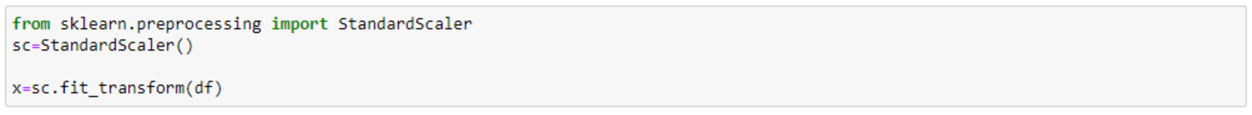
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
0	1	0	0	0	0	5849	0.0	128.0	360.0	1.0	
1	1	1	1	0	0	4583	1508.0	128.0	360.0	1.0	

```
target=df[["Loan_Status"]]
target.head(2)
```

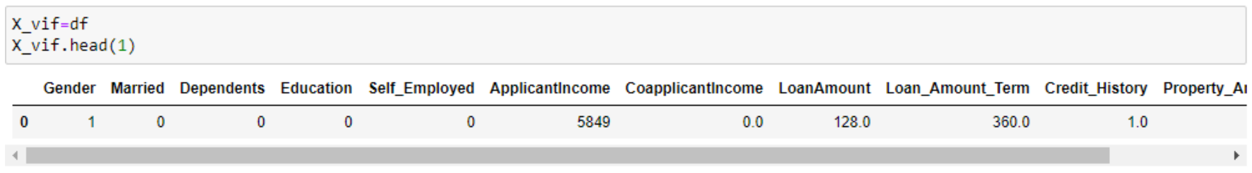
	Loan_Status
0	1
1	0

3.7 Normalization

StandardScaler follows **Standard Normal Distribution (SND)**. Therefore, it makes *mean = 0* and scales the data to unit variance.
We will also check Variance inflation factor (VIF). Variance inflation factor is a measure of the amount of multicollinearity in a set of multiple regression variables.

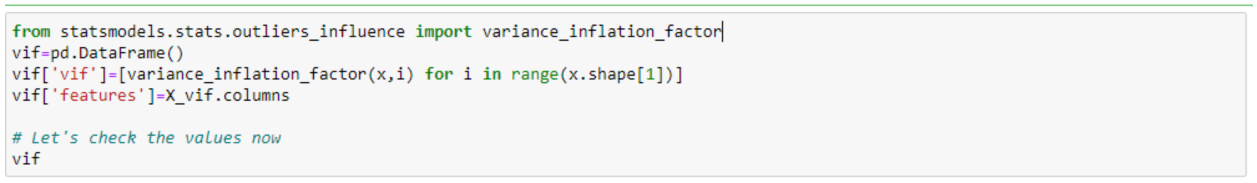


```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

x=sc.fit_transform(df)
```



```
X_vif=df
X_vif.head(1)
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_A
0	1	0	0	0	0	5849	0.0	128.0	360.0	1.0	



```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['vif']=[variance_inflation_factor(x,i) for i in range(x.shape[1])]
vif['features']=X_vif.columns

# Let's check the values now
vif
```

	vif	features
0	1.204378	Gender
1	1.349563	Married
2	1.171914	Dependents
3	1.085114	Education
4	1.072599	Self_Employed
5	1.624032	ApplicantIncome
6	1.465612	CoapplicantIncome
7	1.674810	LoanAmount
8	1.071871	Loan_Amount_Term
9	1.486837	Credit_History
10	1.028319	Property_Area
11	1.497532	Loan_Status

3.8 Feature Importance

One of the great quality of random forest is that they make it very easy to measure the relative importance of each feature. Sklearn measure a features importance by looking at how much the tree nodes, that use that feature, reduce impurity on average (across all trees in the forest). It computes this score automatically for each feature after training and scales the results so that the sum of all importance is equal to 1. We will access this below.

Top six Feature importance are represented below graphically.

```
features.shape, target.shape
```

```
((577, 11), (577, 1))
```

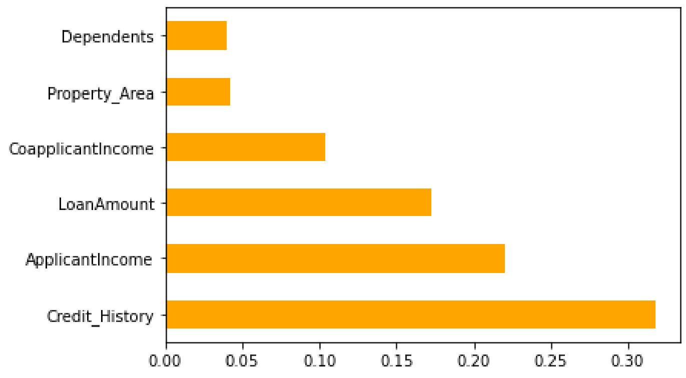
```
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
feature_rf = RandomForestRegressor()
feature_rf.fit(features,target)
```

```
RandomForestRegressor()
```

```
print(feature_rf.feature_importances_)
```

```
[0.01571873 0.02168561 0.04008325 0.01974963 0.01535998 0.2206454
 0.10381464 0.17210072 0.0310037  0.31772872 0.0421096 ]
```

```
#plot graph of feature importances for better visualization
featureImportance = pd.Series(feature_rf.feature_importances_, index=features.columns)
featureImportance.nlargest(6).plot(kind='barh',color="orange") # Lets plot for 6 features
plt.show()
```



We see that Credit_History is the most important feature.

4.Model Building.

I built 5 model and checked the Accuracy and cross validation score. Below is the output of all the five models.

```
modelDF=pd.concat([df_LR,df_SVM,df_dt,df_RF,df_NB]).reset_index()
modelDF.drop(columns=['index'],axis=1,inplace=True)
modelDF
```

	Model	Accuracy	Cross Validation Score
0	Logistic Regression	86.206897	81.112444
1	SVM	63.793103	68.977511
2	Decision Tree	79.310345	74.176912
3	Random Forest	83.620690	81.808096
4	Naive Byes	82.758621	79.383808

Also, we will check the roc_auc score of each graph.

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise'**. The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

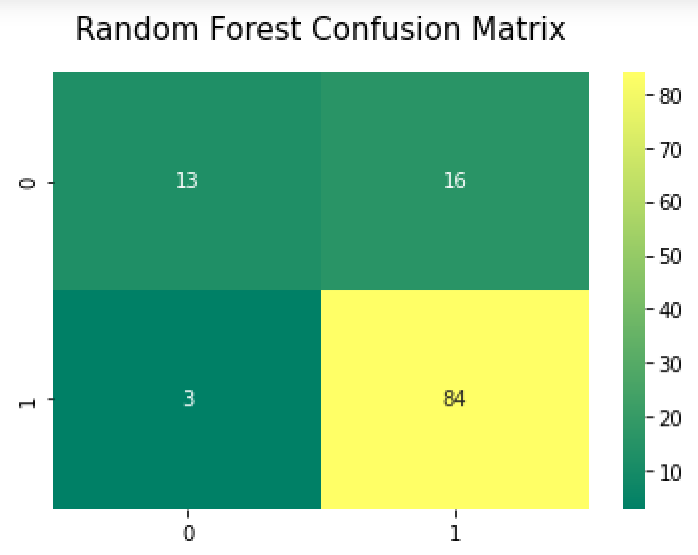
The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

Let's check the confusion matrix and classification report.

A **confusion matrix** is a technique for summarizing the performance of a classification algorithm.

Classification report provides a better understanding of the overall performance of our trained model.

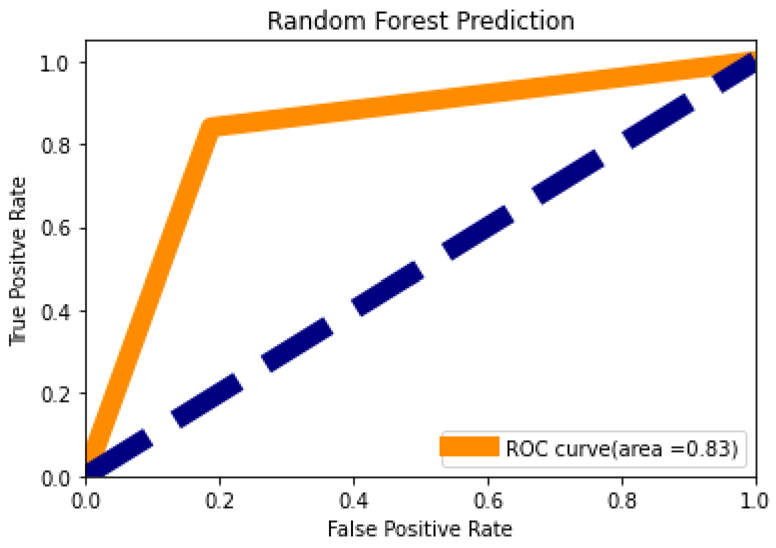
Metrics	**Definitions**
Precision	Precision is defined as the ratio of true positives to the sum of true and false positives
Recall	Recall is defined as the ratio of true positives to the sum of true positives and false negatives.
F1-Score	The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is
Support	Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.



```
: print(classification_report(y_test,pred_test))
```

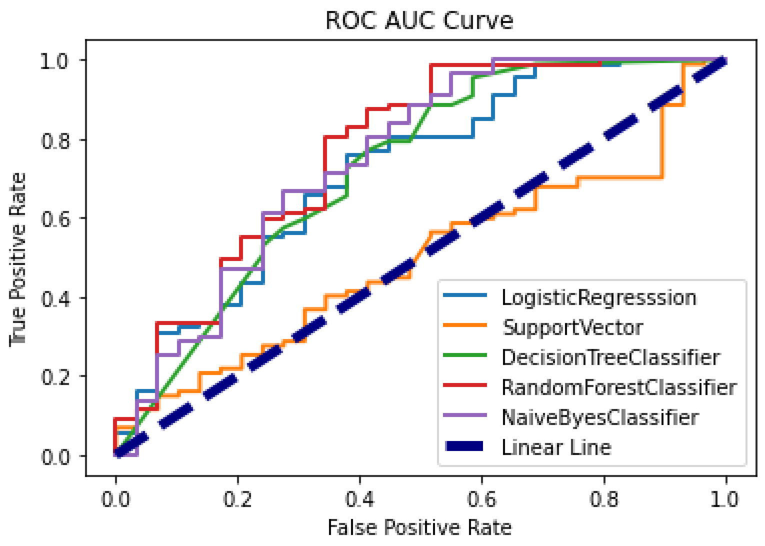
```
              precision    recall  f1-score   support

           0       0.81      0.45      0.58        29
           1       0.84      0.97      0.90        87

    accuracy                           0.84       116
   macro avg       0.83      0.71      0.74       116
weighted avg       0.83      0.84      0.82       116
```

Let’s plot accuracy score graph for Random Forest classifier and observe.

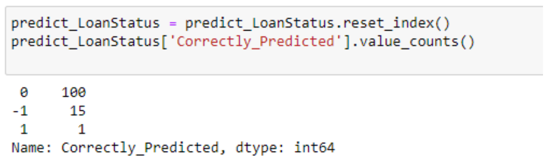


Let’s plot ROC AUC curve of all the models built and anlyse.

```
for key in roc_auc_dict:
    clf = roc_auc_dict[key]
    plt.plot(clf['fpr'], clf['tpr'], label=key,lw=2)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC AUC Curve")
plt.plot([0,1], [0,1], label='Linear Line', color='navy',lw=5,linestyle='--')
plt.legend()
plt.show()
```

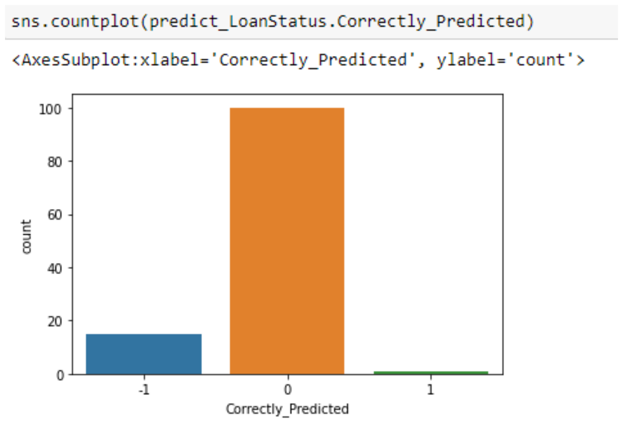


5.Evaluation.



```
predict_LoanStatus = predict_LoanStatus.reset_index()
predict_LoanStatus['Correctly_Predicted'].value_counts()
```

```
 0    100
-1     15
 1      1
Name: Correctly_Predicted, dtype: int64
```



```
sns.countplot(predict_LoanStatus.Correctly_Predicted)
```

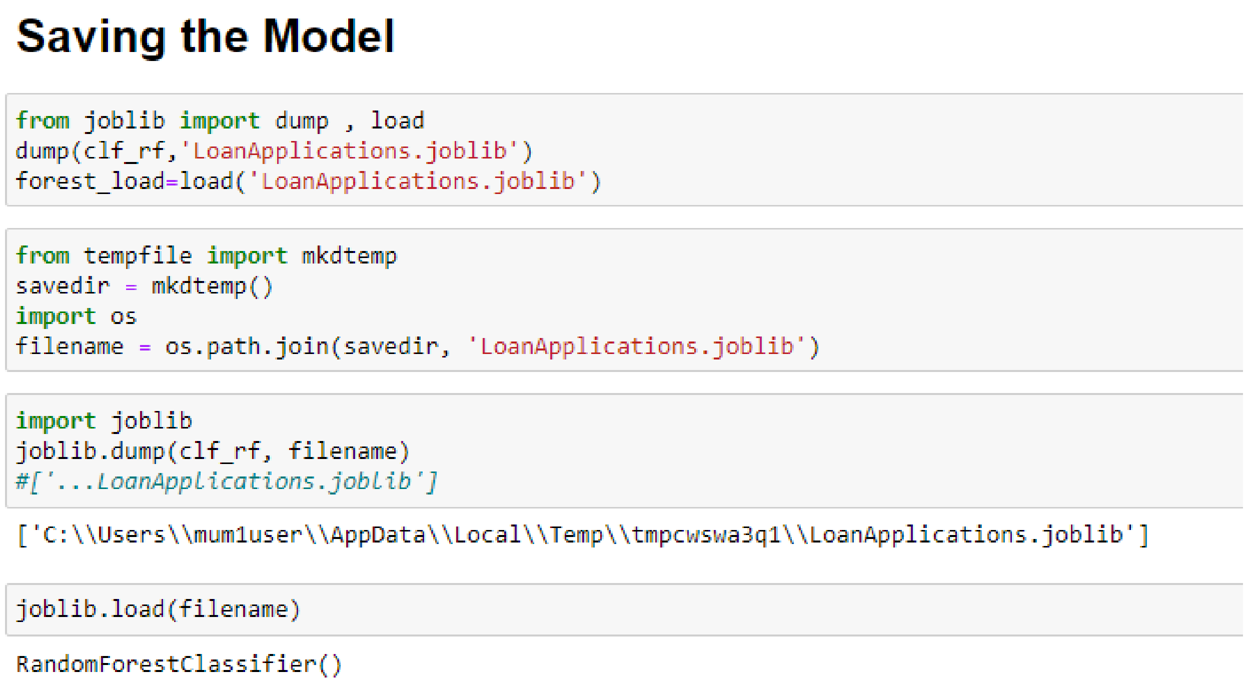
```
<AxesSubplot:xlabel='Correctly_Predicted', ylabel='count'>
```

The above graphs show us the comparison between the True Positive and False Positive rates and we see that Random Forest Classifier is performing best. Also the least difference between Accuracy Score and Cross Validation Score is for Random Forest Classifier, so we will select this model for our prediction. Also F1 Score is 0.84, recall is 0.71 which is pretty good.

85.47% is the accuracy of correctness and we see that model is behaving well.

6.Save the model.

We will save the model at last.



Saving the Model

```
from joblib import dump , load
dump(clf_rf,'LoanApplications.joblib')
forest_load=load('LoanApplications.joblib')
```

```
from tempfile import mkdtemp
savedir = mkdtemp()
import os
filename = os.path.join(savedir, 'LoanApplications.joblib')
```

```
import joblib
joblib.dump(clf_rf, filename)
#['...LoanApplications.joblib']
```

```
['C:\\Users\\mum1user\\AppData\\Local\\Temp\\tmpcwswa3q1\\LoanApplications.joblib']
```

```
joblib.load(filename)
```

```
RandomForestClassifier()
```

7. Conclusion.

This is one of the interesting articles that I have written because it was on today's current top technology machine learning, but I was used basic language to explain this article so that one can't find it difficult to understand.

- We did Exploratory Data Analysis on the features of this dataset and saw how each feature is distributed.
- We analyzed each variable to check if data is cleaned and normally distributed.
- We cleaned the data and removed NA values
- We also generated hypothesis to prove an association among the Independent variables and the Target variable. And based on the results, we assumed whether or not there is an association.
- We calculated correlation between independent variables and found that applicant income and loan amount have significant relation.
- We created dummy variables for constructing the model and extra Loan_Rejected variable for Visualization.
- We constructed models taking different variables into account and found through odds ratio that credit history is creating the most impact on loan giving decision
- We tested the data and got the accuracy of 85.47 %