Loan Prediction

Binary Classification using Logistic Regression

Importing Libraries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Importing & Loading the dataset

In [3]:
```
df = pd.read_csv('train.csv')
df.head()
```

Out[3]:

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

Dataset Info:-

In [4]:
```
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
```

Database shape:-

In [5]:
```
df.shape
```

Out[5]:
```
(614, 13)
```

Data Cleaning

Checking the Missing Values

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 will fill the Missing Values in "LoanAmount" & "Credit_History" by the 'Mean' & 'Median' of the respective variables.

In [7]:
```
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
```

In [8]:
```
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].median())
```

Let's confirm if there are any missing values in 'LoanAmount' & 'Credit_History'

In [9]:
```
df.isna().sum()
```

Out[9]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     14
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

Let's drop all the missing values remaining.

In [10]:
```
df.dropna(inplace=True)
```

Check missing values final time!

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
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
```

Here, we have dropped all the missing values to avoid disturbance in the model. The Loan Prediction requires all the details to work efficiently thus the missing value are dropped.

Let's check the final Dataset Shape

In [12]:
```
df.shape
```

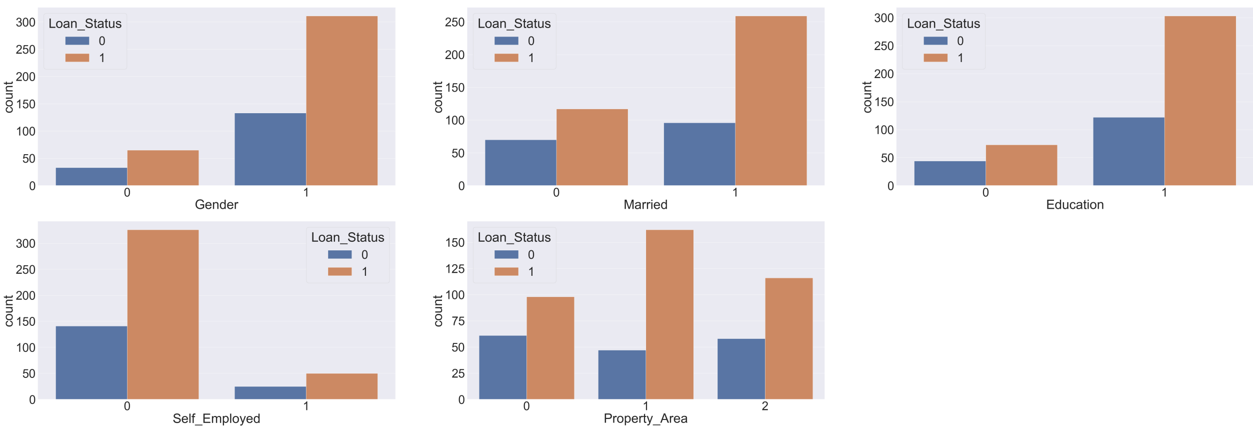
Out[12]:
```
(542, 13)
```

Exploratory Data Analyis

Comparison between Parameters in getting the Loan:

In [45]:
```
plt.figure(figsize = (100, 50))
sns.set(font_scale = 5)
plt.subplot(331)
sns.countplot(x ='Gender',hue=df['Loan_Status'], data=df)

plt.subplot(332)
sns.countplot(x ='Married',hue=df['Loan_Status'], data=df)

plt.subplot(333)
sns.countplot(x ='Education',hue=df['Loan_Status'], data=df)

plt.subplot(334)
sns.countplot(x ='Self_Employed',hue=df['Loan_Status'], data=df)

plt.subplot(335)
sns.countplot(x ='Property_Area',hue=df['Loan_Status'], data=df)
```

Out[45]:
```
<Axes: xlabel='Property_Area', ylabel='count'>
```



Let's replace the Variable values of Numeical form & display the Value Counts

In [27]:
```
df['Loan_Status'].replace('Y', 1, inplace=True)
df['Loan_Status'].replace('N', 0, inplace=True)
```

In [28]:
```
df['Loan_Status'].value_counts()
```

Out[28]:
```
1    376
0    166
Name: Loan_Status, dtype: int64
```

In [17]:
```
df.Married=df.Married.map({'Yes':1,'No':0})
df['Married'].value_counts()
```

Out[17]:
```
1    355
0    187
Name: Married, dtype: int64
```

In [16]:
```
df.Gender=df.Gender.map({'Male':1,'Female':0})
df['Gender'].value_counts()
```

Out[16]:
```
1    444
0     98
Name: Gender, dtype: int64
```

In [18]:
```
df.Dependents=df.Dependents.map({'0':0,'1':1,'2':2,'3+':3})
df['Dependents'].value_counts()
```

Out[18]:
```
0    309
1     94
2     94
3     45
Name: Dependents, dtype: int64
```

In [19]:
```
df.Education=df.Education.map({'Graduate':1,'Not Graduate':0})
df['Education'].value_counts()
```

Out[19]:
```
1    425
0    117
Name: Education, dtype: int64
```

In [20]:
```
df.Self_Employed=df.Self_Employed.map({'Yes':1,'No':0})
df['Self_Employed'].value_counts()
```

Out[20]:
```
0    467
1     75
Name: Self_Employed, dtype: int64
```

In [21]:
```
df.Property_Area=df.Property_Area.map({'Urban':2,'Rural':0,'Semiurban':1})
df['Property_Area'].value_counts()
```

Out[21]:
```
1    209
2    174
0    159
Name: Property_Area, dtype: int64
```

In [23]:
```
df['LoanAmount'].value_counts()
```

Out[23]:
```
146.412162    19
120.000000    15
100.000000    14
110.000000    13
187.000000    12
              ..
280.000000     1
240.000000     1
214.000000     1
59.000000      1
253.000000     1
Name: LoanAmount, Length: 195, dtype: int64
```

In [24]:
```
df['Loan_Amount_Term'].value_counts()
```

Out[24]:
```
360.0    464
180.0     38
480.0     13
300.0     12
84.0       4
120.0      3
240.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

In [25]:
```
df['Credit_History'].value_counts()
```

Out[25]:
```
1.0    468
0.0     74
Name: Credit_History, dtype: int64
```

In [ ]:

From the above figure, we can see that **Credit_History** (Independent Variable) has the maximum correlation with **Loan_Status** (Dependent Variable). Which denotes that the Loan_Status is heavily dependent on the Credit_History.

In [34]:
```
df.head()
```

Out[34]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	1	0	0	1	0	5849	0.0	146.412162	
1	LP001003	1	1	1	1	0	4583	1508.0	128.000000	
2	LP001005	1	1	0	1	1	3000	0.0	66.000000	
3	LP001006	1	1	0	0	0	2583	2358.0	120.000000	
4	LP001008	1	0	0	1	0	6000	0.0	141.000000	

In [35]:
```
df
```

Out[35]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
0	LP001002	1	0	0	1	0	5849	0.0	146.412162	
1	LP001003	1	1	1	1	0	4583	1508.0	128.000000	
2	LP001005	1	1	0	1	1	3000	0.0	66.000000	
3	LP001006	1	1	0	0	0	2583	2358.0	120.000000	
4	LP001008	1	0	0	1	0	6000	0.0	141.000000	
...	...	...	...	...	...	...	...	...	...	
609	LP002978	0	0	0	1	0	2900	0.0	71.000000	
610	LP002979	1	1	3	1	0	4106	0.0	40.000000	
611	LP002983	1	1	1	1	0	8072	240.0	253.000000	
612	LP002984	1	1	2	1	0	7583	0.0	187.000000	
613	LP002990	0	0	0	1	1	4583	0.0	133.000000	

542 rows × 13 columns

Importing Packages for Classification algorithms

In [32]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

splitting the data into Train and Test set

In [38]:
```
X = df.iloc[1:542,1:12].values
y = df.iloc[1:542,12].values
```

In [37]:
```
df.iloc[1:542,1:12].values
```

Out[37]:
```
array([[  1.,   1.,   1., ..., 360.,   1.,   0.],
       [  1.,   1.,   0., ..., 360.,   1.,   2.],
       [  1.,   1.,   0., ..., 360.,   1.,   2.],
       ...,
       [  1.,   1.,   1., ..., 360.,   1.,   2.],
       [  1.,   1.,   2., ..., 360.,   1.,   2.],
       [  0.,   0.,   0., ..., 360.,   0.,   1.]])
```

In [40]:
```
df.iloc[1:542,12].values
```

Out[40]:
```
array([0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 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, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0], dtype=int64)
```

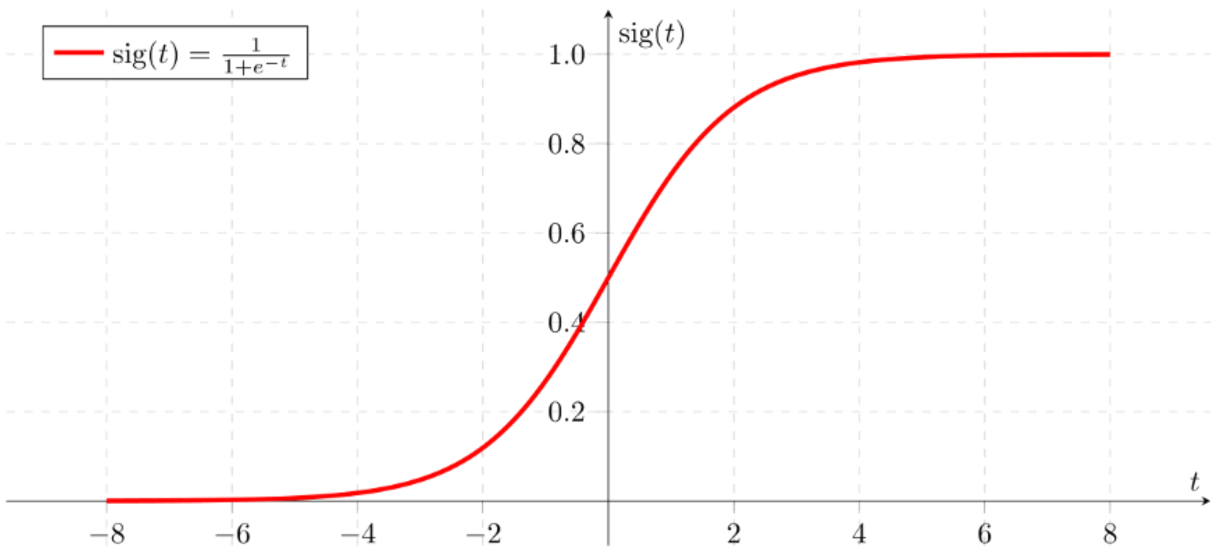
In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=0)
```

Logistic Regression (LR)

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable.

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

Sigmoid Function



In [43]:
```
model = LogisticRegression()
model.fit(X_train,y_train)

lr_prediction = model.predict(X_test)
print('Logistic Regression accuracy = ', metrics.accuracy_score(lr_prediction, y_test))
```

```
Logistic Regression accuracy =  0.7852760736196319
```

In [44]:
```
print('y_predicted', lr_prediction)
print('y_test',y_test)
```

```
y_predicted [1 0 0 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 1 0 1 1 0 0 1 1 0
 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 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 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1
 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0]
y_test [0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0 1 0
 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0
 1 0 0 0 0 1 0 1 0 1 1 1 1 1 0]
```

CONCLUSION:

1. The Loan Status is heavily dependent on the Credit History for Predictions.
2. The Logistic Regression algorithm gives us the maximum Accuracy (79% approx) compared to the other 3 Machine Learning Classification Algorithms.

Complete Project on Github : https://github.com/Vyas-Rishabh/Python_Loan_Prediction_using_Logistic_Regression