```
# importing the required libraries
import pandas as pd
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#CHECKING VERSION OF SKLEARN
print(sklearn.__version__)
```

```
0.22.2.post1
```

```
#loading the preprocessed dataset
data=pd.read_csv("/content/Loan_Prediction_New_Data.csv")
```

```
data.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncor
0	LP001002	0.0	0.0	0.000000	1.0	0.0	0.0704
1	LP001003	0.0	1.0	0.333333	1.0	0.0	0.0548
2	LP001005	0.0	1.0	0.000000	1.0	1.0	0.0352
3	LP001006	0.0	1.0	0.000000	0.0	0.0	0.0300
4	LP001008	0.0	0.0	0.000000	1.0	0.0	0.0723

```
#checking missing values
data.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

```
#checking data types
data.dtypes
```

```
Loan_ID               object
Gender               float64
Married              float64
Dependents           float64
Education            float64
Self_Employed        float64
ApplicantIncome      float64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area        float64
Loan_Status          float64
dtype: object
```

```
#dropint the loan id since it is just a unique value
data=data.drop("Loan_ID",axis=1)
```

```
data.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	0.0	0.0	0.000000	1.0	0.0	0.070489	0.0
1	0.0	1.0	0.333333	1.0	0.0	0.054830	0.0
2	0.0	1.0	0.000000	1.0	1.0	0.035250	0.0
3	0.0	1.0	0.000000	0.0	0.0	0.030093	0.0
4	0.0	0.0	0.000000	1.0	0.0	0.072356	0.0

```
data.shape
```

```
(614, 12)
```

```
#SEPERATE THE DATA INTO DEPENDENT AND INDEPENDENT
x=data.drop("Loan_Status",axis=1)
```

```
#independent variable
x
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	0.0	0.0	0.000000	1.0	0.0	0.070489	C
1	0.0	1.0	0.333333	1.0	0.0	0.054830	C
2	0.0	1.0	0.000000	1.0	1.0	0.035250	C
3	0.0	1.0	0.000000	0.0	0.0	0.030093	C
4	0.0	0.0	0.000000	1.0	0.0	0.072356	C
...	...	...	...	...	...	...	
609	1.0	0.0	0.000000	1.0	0.0	0.034014	C
610	0.0	1.0	1.000000	1.0	0.0	0.048930	C
611	0.0	1.0	0.333333	1.0	0.0	0.097984	C
612	0.0	1.0	0.666667	1.0	0.0	0.091936	C

```
y=data["Loan_Status"]
```

614 rows × 11 columns

```
#dependent variable(target variable)
y
```

```
0      1.0
1      0.0
2      1.0
3      1.0
4      1.0
      ...
609    1.0
610    1.0
611    1.0
612    1.0
613    0.0
Name: Loan_Status, Length: 614, dtype: float64
```

```
#shape of independent and dependent variable
x.shape,y.shape
```

```
((614, 11), (614,))
```

CREATING TEST AND TRAINING DATA

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,stratify=data["Loan_Status"],random_state=10,t
```

```
#shape of training and test set
(xtrain.shape,ytrain.shape),(xtest.shape,ytest.shape)
```

```
(((491, 11), (491,)), ((123, 11), (123,)))
```

```
xtrain
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplican
164	0.0	1.0	0.000000	1.0	0.0	0.113457	C
171	0.0	1.0	1.000000	1.0	0.0	0.638380	C
546	0.0	0.0	0.000000	0.0	0.0	0.039678	C
226	0.0	1.0	0.000000	0.0	1.0	0.056710	C
176	0.0	1.0	0.666667	1.0	0.0	0.023438	C
...	...	...	...	...	...	...	
560	0.0	1.0	0.666667	0.0	0.0	0.043599	C
503	0.0	1.0	0.333333	0.0	0.0	0.048237	C
343	0.0	1.0	1.000000	0.0	0.0	0.037390	C
148	1.0	0.0	0.000000	1.0	0.0	0.121831	C
303	0.0	1.0	0.333333	1.0	0.0	0.018244	C

491 rows × 11 columns

```
xtest
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplican
369	0.0	1.0	0.000000	1.0	0.0	0.242177	C

```
ytrain
```

```
164    1.0
171    1.0
546    0.0
226    0.0
176    1.0
      ...
560    1.0
503    0.0
343    1.0
148    0.0
303    1.0
Name: Loan_Status, Length: 491, dtype: float64
```

```
ytest
```

```
507    0.0
493    1.0
434    1.0
125    1.0
294    1.0
      ...
82     0.0
295    1.0
369    0.0
450    0.0
363    1.0
Name: Loan_Status, Length: 123, dtype: float64
```

DEFINING ARCHITECTURE OF THE MODL

```
#importing keras
import keras
```

```
#checking the version of the keras
print(keras.__version__)
```

```
2.5.0
```

```
#importting tensorflow
import tensorflow as tf
```

```
#checcking the version of the tensorflow
print(tf.__version__)
```

```
2.5.0
```

```
#importing sequential function from keras
from keras.models import Sequential
```

```
#importing different layers
from keras.layers import InputLayer,Dense
```

DEFINING THE NUMBER OF INPUT NEURONS

```
xtrain.shape
```

```
(491, 11)
```

```
xtrain.shape[1]
```

```
11
```

```
#defining the input neurons
input_neurons=xtrain.shape[1]
```

```
#defining the no of output neurons
#binary classification problem - output neuron = 1
output_neurons=1
```

```
#defining hidden layer and no of neurons in each layer
number_of_hidden_layers=2
neuron_hidden_layer_1=10
neuron_hidden_layer_2=5
```

```
#defining the architecture of the model
model = Sequential()
model.add(InputLayer(input_shape=(input_neurons,)))
model.add(Dense(units=neuron_hidden_layer_1, activation='relu'))
model.add(Dense(units=neuron_hidden_layer_2, activation='relu'))
model.add(Dense(units=output_neurons, activation='sigmoid'))
```

```
#summary of the model
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	120
dense_1 (Dense)	(None, 5)	55

```
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 6
=================================================================
Total params: 181
Trainable params: 181
Non-trainable params: 0
_________________________________________________________________
```

```
#no of parameter between input and first hidden layer
input_neurons*neuron_hidden_layer_1
```

```
110
```

```
#Adding bias for each neurons for first hidden layer
input_neurons*neuron_hidden_layer_1+10
```

```
120
```

```
#no.of parameter between first and second hidden layer
neuron_hidden_layer_1*neuron_hidden_layer_2+5
```

```
55
```

```
#no.of parameter between second hiden layer and output layer
neuron_hidden_layer_2*output_neurons+1
```

```
6
```

COMPILING THE MODEL (DEFINING LOSS AND OPTIMIZER)

```
#compiling the model
# loss as binary_crossentropy, since we have binary classification problem
# defining the optimizer as adam
# Evaluation metric as accuracy
model.compile(loss="bianry_crossentropy",optimizer="Adam",metrics=["accuracy"])
```

0s completed at 4:11 PM