```
# 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	ApplicantInco
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	Coappli
0	0.0	0.0	0.000000	1.0	0.0	0.070489	
1	0.0	1.0	0.333333	1.0	0.0	0.054830	
2	0.0	1.0	0.000000	1.0	1.0	0.035250	
3	0.0	1.0	0.000000	0.0	0.0	0.030093	
4	0.0	0.0	0.000000	1.0	0.0	0.072356	

```
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	Coapp
0	0.0	0.0	0.000000	1.0	0.0	0.070489	
1	0.0	1.0	0.333333	1.0	0.0	0.054830	
2	0.0	1.0	0.000000	1.0	1.0	0.035250	
3	0.0	1.0	0.000000	0.0	0.0	0.030093	
4	0.0	0.0	0.000000	1.0	0.0	0.072356	
...	...	...	...	...	...	...	
609	1.0	0.0	0.000000	1.0	0.0	0.034014	
610	0.0	1.0	1.000000	1.0	0.0	0.048930	
611	0.0	1.0	0.333333	1.0	0.0	0.097984	
612	0.0	1.0	0.666667	1.0	0.0	0.091936	

```
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	Coapp
164	0.0	1.0	0.000000	1.0	0.0	0.113457	
171	0.0	1.0	1.000000	1.0	0.0	0.638380	
546	0.0	0.0	0.000000	0.0	0.0	0.039678	
226	0.0	1.0	0.000000	0.0	1.0	0.056710	
176	0.0	1.0	0.666667	1.0	0.0	0.023438	
...	...	...	...	...	...	...	
560	0.0	1.0	0.666667	0.0	0.0	0.043599	
503	0.0	1.0	0.333333	0.0	0.0	0.048237	
343	0.0	1.0	1.000000	0.0	0.0	0.037390	
148	1.0	0.0	0.000000	1.0	0.0	0.121831	
303	0.0	1.0	0.333333	1.0	0.0	0.018244	

491 rows × 11 columns

```
xtest
```

```
          Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  Coapp
```

```
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
```

```
 369      0.0       1.0     0.000000         1.0             0.0          0.242177
```

```
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__)
```

```
2.6.0
```

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

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

```
2.6.0
```

```
#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="binary_crossentropy",optimizer="Adam",metrics=["accuracy"])
```

```
# training the model

# passing the independent and dependent features for training set for training the model

# validation data will be evaluated at the end of each epoch

# setting the epochs as 50

# storing the trained model in model_history variable which will be used to visualize the tra
```

```
model_history = model.fit(xtrain, ytrain, validation_data=(xtest, ytest), epochs=50)
```

```
Epoch 22/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4850 - accuracy: 0.810
Epoch 23/50
16/16 [==============================] - 0s 3ms/step - loss: 0.4822 - accuracy: 0.810
Epoch 24/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4791 - accuracy: 0.812
Epoch 25/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4764 - accuracy: 0.812
Epoch 26/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4752 - accuracy: 0.812
Epoch 27/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4731 - accuracy: 0.812
Epoch 28/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4722 - accuracy: 0.810
Epoch 29/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4710 - accuracy: 0.812
Epoch 30/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4698 - accuracy: 0.810
Epoch 31/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4693 - accuracy: 0.810
Epoch 32/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4688 - accuracy: 0.810
Epoch 33/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4678 - accuracy: 0.810
Epoch 34/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4671 - accuracy: 0.810
Epoch 35/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4668 - accuracy: 0.810
Epoch 36/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4664 - accuracy: 0.810
Epoch 37/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4657 - accuracy: 0.810
Epoch 38/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4654 - accuracy: 0.810
Epoch 39/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4650 - accuracy: 0.810
Epoch 40/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4659 - accuracy: 0.812
Epoch 41/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4638 - accuracy: 0.810
Epoch 42/50
16/16 [==============================] - 0s 3ms/step - loss: 0.4641 - accuracy: 0.810
Epoch 43/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4638 - accuracy: 0.810
Epoch 44/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4631 - accuracy: 0.810
Epoch 45/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4630 - accuracy: 0.810
Epoch 46/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4629 - accuracy: 0.810
Epoch 47/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4625 - accuracy: 0.810
Epoch 48/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4626 - accuracy: 0.810
Epoch 49/50
```

```
16/16 [==============================] - 0s 4ms/step - loss: 0.4622 - accuracy: 0.810
Epoch 50/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4623 - accuracy: 0.810
```

PREDICTIONS

```
prediction = model.predict(xtest)
```

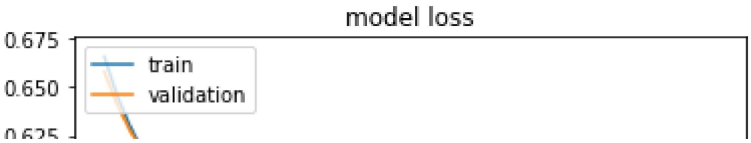
```
#calculating accuracy
accuracy_score(ytest,prediction)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-48-e66ad745415c> in <module>()
      1 #calculating accuracy
----> 2 accuracy_score(ytest,prediction)

                                  1 frames
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py in
_check_targets(y_true, y_pred)
     88     if len(y_type) > 1:
     89         raise ValueError("Classification metrics can't handle a mix of
{0} "
---> 90                          "and {1} targets".format(type_true, type_pred))
     91
     92     # We can't have more than one value on y_type => The set is no more
needed

ValueError: Classification metrics can't handle a mix of binary and continuous
```

```
# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```




```
# summarize history for accuracy
plt.plot(model_history.history['acc'])
plt.plot(model_history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-52-b7dd9c61dd7c> in <module>()
      1 # summarize history for accuracy
----> 2 plt.plot(model_history.history['acc'])
      3 plt.plot(model_history.history['val_acc'])
      4 plt.title('model accuracy')
      5 plt.ylabel('accuracy')

KeyError: 'acc'
```

SEARCH STACK OVERFLOW

0s completed at 11:35 AM