In [ ]:
```
pip install keras
```

In [ ]:
```
pip install tensorflow
```

In [ ]:
```
#import activation function in a keras sequential model
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
```

In [ ]:
```
#Import necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
#Read the training data
data = pd.read_csv(r'C:\Users\vxlli\Downloads\diabetes.csv')
```

In [ ]:
```
#show the dataframe
data.head(5)
```

In [ ]:
```
#separate the features
features = data.drop('Outcome', axis = 'columns')
```

In [ ]:
```
#separate the target
target = data['Outcome']
```

In [ ]:
```
#Assigning to conventional variables, the features and target
X_train = features
Y_train = target
```

In [ ]:
```
#solit the dataset into training and testing
from sklearn.model_selection import train_test_split
X_trian, X_test, Y_train, Y_test = train_test_split(features, target, test_si
#random_state = 0; we get the same train and test sets across different execu
```

In [ ]:
```
#print the dimension of train and test data
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

In [ ]:
```
#Define layers
model = Sequential ([
        Dense(units=16,input_shape=(8,), activation='relu'),#input feature -
        Dense(units=32, activation= 'relu'), #2nd hidden layer with 32 nodes
        Dense(units=2, activation='sigmoid') #2 outputs
])
```

In [ ]:
```
#before training the model, will be compileing it
model.compile(
optimizer=Adam(learning_rate=0.0001), #Adam is a variant of SGD
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
#to the compile() function, we are passing the optimizer, the loss function,
```

In [ ]:
```
model.fit (features,
           target,
           validation_split=0.1,  #10% validation set#
           batch_size=10, #how many training samples should be sent to the mo
           epochs=30, #the complete training set (all of the samples) will be
           shuffle= True,
           verbose=2)
```

In [ ]:
```
#predict the response for test dataset
#Y_pred = model.predict(X_test)
Y_pred = np.argmax(model.predict(X_test),axis=1)
```

In [ ]:
```
pip install sklearn
```

In [ ]:
```
#model Accuracy, how often is classifier correct
from sklearn import metrics
from sklearn.metrics import accuracy_score
print ("Accuracy:", accuracy_score(Y_test, Y_pred))
```

In [ ]:
```
#confusion matirix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
print (cm)
```

In [ ]:
```
TN = cm [0][0]
FN = cm [1][0]
FP = cm [0][1]
TP = cm [1][1]
```

In [ ]:
```
print ('TP = ', TP)
print ('TN = ', TN)
print ('FP = ', FP)
print ('FN = ', FN)
```

In [ ]:
```
#specificity quantifies the ability to avoid false positive
print ('Specificity = ', TN / (TN +FP))
```

In [ ]:
```
#specificity quantifies the ability to avoid false nigative
print ('Specificity = ', TP / (TP +FN))
```

In [ ]:
```
#Precision
from sklearn.metrics import precision_score
print ("Precision:", precision_score(Y_test, Y_pred, average = None))
```

In [ ]:
```
#Recall
from sklearn.metrics import recall_score
print ("Recall:", recall_score(Y_test, Y_pred, average = None))
```

In [ ]:
```
#F1 score
from sklearn.metrics import f1_score
print ("F-score:", f1_score(Y_test, Y_pred, average = None))
```

In [ ]:
```
#print classification report
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

In [ ]:
```

```