```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

## Data Processing

```
data=pd.read_csv("/content/drive/MyDrive/PROJECT_Final.csv")
data.head()
min_load=data.min()
print(min_load)
max_load=data.max()
print(max_load)
```

```
Load    3377.9196
dtype: float64
Load    8841.66948
dtype: float64
```

```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
data=ms.fit_transform(data)
print(data)
print(data.shape)
```

```
[[0.39787738]
 [0.29380046]
 [0.27645431]
 ...
```

```
 [0.24629825]
 [0.32438447]
 [0.65165045]]
(2184, 1)
```

```
a=np.zeros((len(data)-72,7))
print(a.shape)
```

```
(2112, 7)
```

```
k=72
for i in range(2112):
    a[i,0],a[i,1],a[i,2],a[i,3],a[i,4],a[i,5],a[i,6]=data[k-1],data[k-2],data[k-3],data[k-24],data[k-48],data[k-72],data[k]
    k=k+1
print(a)
```

```
[[0.44580754 0.55202419 0.4649796  ... 0.28430002 0.39787738 0.41017375]
 [0.41017375 0.44580754 0.55202419 ... 0.25579869 0.29380046 0.36723631]
 [0.36723631 0.41017375 0.44580754 ... 0.24450087 0.27645431 0.30646772]
 ...
 [0.21779692 0.22852415 0.3052124  ... 0.63393341 0.66531625 0.24629825]
 [0.24629825 0.21779692 0.22852415 ... 0.48380931 0.47074263 0.32438447]
 [0.32438447 0.24629825 0.21779692 ... 0.39228553 0.4173918  0.65165045]]
```

```
#Train-Test Split
from sklearn.model_selection import train_test_split
a,a_test=train_test_split(a,test_size=0.05,random_state=1)
print(a.shape)
print(a_test.shape)
q1=len(a)
print(q1)
q2=len(a_test)
print(q2)
```

```
(2006, 7)
(106, 7)
2006
106
```

## Delta Learning

```
#Random Weight Initialization
weights=np.random.uniform(-1,1,size=(6,1))
print(weights.shape)
print(weights)
```

```
(6, 1)
[[ 0.8313804 ]
 [ 0.95619909]
 [-0.68146564]
 [-0.38917163]
 [ 0.66861926]
 [-0.16388426]]
```

```
#Delta Learning

l=0.001
for i in range(100):
    for j in range(q1):
        n=np.dot(a[j,0:6],weights) #n.shape-1,1
        o=1/(1+np.exp(-n))
        dw=(l*(a[[j],6]-o))*o*(1-o)*a[[j],0:6].T
        #dw3=dw3.reshape((6,1))
        weights=weights+dw
print(dw)
print(weights)
```

```
[[-2.42268038e-05]
 [-2.30934374e-05]
 [-2.47539509e-05]
 [-2.27112558e-05]
 [-3.16332210e-05]
 [-2.70997557e-05]]
[[ 0.86961054]
 [ 0.65315685]
```

```
[-1.23016548]
[-0.31005958]
[ 0.53735461]
[-0.30695689]]
```

```
#Delta Learning Testing
o2=np.zeros((q2,1))
for j in range(q2):
    n2=np.dot(a_test[j,0:6],weights) #n.shape-1,1
    o2[j]=1/(1+np.exp(-n2))

from sklearn.metrics import mean_squared_error
print(mean_squared_error(a_test[:,6],o2))
```

```
0.03265287858024362
```

```
# Prediction
c=10
n=np.dot(a[c,0:6],weights) #n.shape-1,1
o=1/(1+np.exp(-n))
Actual_load=a[c,6]*(max_load-min_load)+min_load
Estimated_Load=o*(max_load-min_load)+min_load
print('Actual Load=', Actual_load)
print('Estimated Load=', Estimated_Load)
```

```
Actual Load= Load    5260.01472
dtype: float64
Estimated Load= Load    5963.276331
dtype: float64
```

## *Percepton Learning*

---

```
#Random Weight Initialization
weights=np.random.uniform(-1,1,size=(6,1))
print(weights.shape)
```

```
(6, 1)
```

```
#Percepton Learning

l=0.001
for i in range(10):
    for j in range(q1):
        n=np.dot(a[j,0:6],weights) #n.shape-1,1
        if(n>0):
            o=1
        else:
            o=0
        dw=(l*(a[j,6]-o))*a[j,0:6].T
        dw=dw.reshape((6,1))
        weights=weights+dw
print(dw)
print(weights)
```

```
[[-0.00024697]
 [-0.00023542]
 [-0.00025235]
 [-0.00023152]
 [-0.00032248]
 [-0.00027626]]
[[-0.09892914]
 [ 0.01361865]
 [ 0.02794325]
 [ 0.00420188]
 [ 0.05707086]
 [-0.00329747]]
```

## *Widrow-Hoff Learning*

```
#Random Weight Initialization
weights=np.random.uniform(-1,1,size=(6,1))
```

```
print(weights.shape)

    (6, 1)


#Widrow-Hoff

l=0.001

for i in range(100):
    for j in range(q1):
        o=np.dot(a[j,0:6],weights) #n.shape-1,1
        dw=(l*(a[j,6]-o))*a[j,0:6].T
        dw=dw.reshape((6,1))
        weights=weights+dw
print(dw)
print(weights)

    [[-3.43472874e-05]
     [-3.27404696e-05]
     [-3.50946445e-05]
     [-3.21986357e-05]
     [-4.48476549e-05]
     [-3.84203837e-05]]
    [[ 0.42135481]
     [ 0.19796593]
     [-0.17187005]
     [ 0.27819358]
     [ 0.20912137]
     [ 0.055513  ]]


#Widro-off Testing
n3=np.zeros((q2,1))
j=0
for j in range(q2):
    n3[j]=np.dot(a_test[j,0:6],weights)

print(mean_squared_error(a_test[:,6],n3))

    0.010004416357981018
```

```
# Prediction
c=100
n=np.dot(a[c,0:6],weights) #n.shape-1,1
o=1/(1+np.exp(-n))
print(o*(max_load-min_load)+min_load)
print(a[c,6]*(max_load-min_load)+min_load)
```

```
Load    6969.10396
dtype: float64
Load    8063.04888
dtype: float64
```

## *Back Propagation Algorithm - Stochastic Gradient Descent Optimizer*

---

```
# Random weight initialization
w_ij=np.random.uniform(-1,1,size=(6,9))
w_jk=np.random.uniform(-1,1,size=(9,1))
print(w_ij.shape,w_ij)
print(w_jk.shape,w_jk)
```

```
(6, 9) [[-0.59482148 -0.73960882 -0.83806011 -0.12360915 -0.26677319  0.58011487
   0.22763291 -0.25572153 -0.34312126]
 [ 0.50494945 -0.40017443 -0.93641421 -0.1363237   0.23247974 -0.528524
   0.64306293  0.10676056 -0.89132745]
 [ 0.34025202 -0.03396779 -0.2234081   0.93628763 -0.33446971 -0.78869486
   0.27270719 -0.65696864  0.54255924]
 [ 0.30123374 -0.90524777 -0.18050852 -0.36602098  0.38513655  0.64327311
   0.4899085   0.40656255  0.61182284]
 [ 0.7208976   0.8642463  -0.94835411 -0.9865912  -0.21387376 -0.73126479
  -0.83789006 -0.67588242 -0.23887989]
 [-0.26124355 -0.024582    0.28513343  0.93605488  0.30331545  0.53853705
  -0.85042706 -0.67882117  0.36574263]]
(9, 1) [[-0.06883214]
 [-0.58127148]
 [ 0.67229065]
 [ 0.44773562]
```

```
 [-0.3137932 ]
 [-0.88517272]
 [-0.49407162]
 [-0.7513405 ]
 [-0.46136104]]
```

```
#Forward pass
def forward_pass(a,w_ij,w_jk,j):
    net_j=np.dot(a[[j],0:6],w_ij)
    o_j=1/(1+np.exp(-net_j))
    net_k=np.dot(o_j,w_jk)
    o_k=1/(1+np.exp(-net_k))
    return o_j,o_k


#Backward pass
def weights_updation_bp(a,l,o_j,o_k,w_ij,w_jk,j):
    dw_ij=np.zeros((6,9))
    dw_jk=np.dot(l*(a[[j],6]-o_k)*(o_k*(1-o_k)),o_j)
    w_jk=w_jk+dw_jk.T
    for k in range(9):
        dw_ij[:,k]=np.dot(l*(a[[j],6]-o_k)*(o_k*(1-o_k))*w_jk[[k]]*o_j[0,k]*(1-o_j[0,k]),a[[j],0:6])
    w_ij=w_ij+dw_ij
    return w_jk,w_ij


#Neural Nets
l=0.001
for i in range(100):
    for j in range(q1):
        o_j,o_k=forward_pass(a,w_ij,w_jk,j)
        w_jk,w_ij=weights_updation_bp(a,l,o_j,o_k,w_ij,w_jk,j)
print(w_jk)
print(w_ij)
```

```
[[ 0.47610798]
 [-0.78557408]
 [ 0.3490286 ]
 [ 0.7716443 ]
 [ 0.01017118]]
```

```
 [-0.6739683 ]
 [-0.37602983]
 [-0.93067531]
 [-0.29755399]]
[[-0.49732511 -0.88113064 -0.75695252  0.1288583  -0.28742616  0.33851079
   0.11071092 -0.41169464 -0.44357747]
 [ 0.56350976 -0.48258263 -0.89835941  0.03297404  0.21502361 -0.69455836
   0.56150447  0.01687753 -0.96148316]
 [ 0.35963649 -0.05727361 -0.22782859  1.02115107 -0.34853489 -0.87819808
   0.22846166 -0.67768686  0.50299077]
 [ 0.39562178 -1.04324378 -0.10028798 -0.11896531  0.36477951  0.40557868
   0.37600743  0.25453511  0.51313449]
 [ 0.80428024  0.74067597 -0.8820432  -0.76163707 -0.23334977 -0.94771316
  -0.9410412  -0.80610594 -0.3289869 ]
 [-0.18817504 -0.13008991  0.34044651  1.13745776  0.28469083  0.3421058
  -0.94337088 -0.78988385  0.28397994]]
```

```
#Error Calculation
p=np.zeros((q2,1))
for z in range(q2):
    _,p[z]=forward_pass(a_test,w_ij,w_jk,z)
print(p.shape)
```

```
(106, 1)
```

```
from sklearn.metrics import mean_squared_error
print(mean_squared_error(a_test[:,6],p))
```

```
0.02818262267211648
```

```
c=10
_,p=forward_pass(a,w_ij,w_jk,c)
print(p[0]*(max_load-min_load)+min_load)
print(a[c,6]*(max_load-min_load)+min_load)
```

```
Load    5829.553545
dtype: float64
Load    5260.01472
dtype: float64
```

# Implement ANN with keras - Regression Problem

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

import numpy as np
import matplotlib.pyplot as plt
import keras  #Keras is the deep learning library that helps you to code Deep Neural Networks with fewer lines of code
#from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import RMSprop,Adadelta,SGD,Adagrad,Adam
#import pylab as plt
#import seaborn as sns #For data visualization
import pandas as pd # For Data manipulation
```

```
load_data=pd.read_csv("/content/drive/MyDrive/PROJECT_Final.csv")
load_data.head()
min_load=load_data.min()
print(min_load)
max_load=load_data.max()
print(max_load)
```

```
Load    3377.9196
dtype: float64
Load    8841.66948
dtype: float64
```

```
print(load_data.shape) # details about number of samples and features
load_data.describe()
```

(2184, 1)

| | Load |
|---|---|
| **count** | 2184.000000 |
| **mean** | 6028.125312 |
| **std** | 1066.398766 |
| **min** | 3377.919600 |
| **25%** | 5258.767680 |
| **50%** | 5935.910400 |

```
load_data.isnull().any()
#load_data = load_data.fillna(method='ffill')
```

```
Load    False
dtype: bool
```

```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
load_data=ms.fit_transform(load_data)
print(load_data)
```

```
[[0.39787738]
 [0.29380046]
 [0.27645431]
 ...
 [0.24629825]
 [0.32438447]
 [0.65165045]]
```

```
load_data_process=np.zeros((len(load_data)-72,7))
print(load_data_process.shape)
```

```
(2112, 7)
```

```
k=72
for i in range(2112):
    load_data_process[i,0],load_data_process[i,1],load_data_process[i,2],load_data_process[i,3],load_data_process[i,4],load_
    k=k+1
print(load_data_process)
```

```
[[0.44580754 0.55202419 0.4649796  ... 0.28430002 0.39787738 0.41017375]
 [0.41017375 0.44580754 0.55202419 ... 0.25579869 0.29380046 0.36723631]
 [0.36723631 0.41017375 0.44580754 ... 0.24450087 0.27645431 0.30646772]
 ...
 [0.21779692 0.22852415 0.3052124  ... 0.63393341 0.66531625 0.24629825]
 [0.24629825 0.21779692 0.22852415 ... 0.48380931 0.47074263 0.32438447]
 [0.32438447 0.24629825 0.21779692 ... 0.39228553 0.4173918  0.65165045]]
```

```
dataset=pd.DataFrame(data=load_data_process[0:,0:])
print(dataset[1].values)
```

```
[0.55202419 0.44580754 0.41017375 ... 0.22852415 0.21779692 0.24629825]
```

```
X=dataset.iloc[:,0:6].values
Y=dataset.iloc[:,6:].values
print(X)
print(Y)
```

```
[[0.44580754 0.55202419 0.4649796  0.38115888 0.28430002 0.39787738]
 [0.41017375 0.44580754 0.55202419 0.31479844 0.25579869 0.29380046]
 [0.36723631 0.41017375 0.44580754 0.30840775 0.24450087 0.27645431]
 ...
 [0.21779692 0.22852415 0.3052124  0.68414596 0.63393341 0.66531625]
 [0.24629825 0.21779692 0.22852415 0.51467861 0.48380931 0.47074263]
 [0.32438447 0.24629825 0.21779692 0.41730621 0.39228553 0.4173918 ]]
[[0.41017375]
 [0.36723631]
 [0.30646772]
 ...
 [0.24629825]
 [0.32438447]
 [0.65165045]]
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print(y_test[0:5])
print(X_test.shape)
print(X_train.shape)
```

```
[[0.73122022]
 [0.54791589]
 [0.27154717]
 [0.83178797]
 [0.7313914 ]]
(423, 6)
(1689, 6)
```

```
#First_Layer_Size = 32 # Number of neurons in first layer
model=Sequential()
model.add(Dense(32,activation='tanh', input_shape=(6,)))
model.add(Dense(32,activation='tanh'))
model.add(Dense(32,activation='tanh'))
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                224
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,369
Trainable params: 2,369
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='MSE',
              optimizer=SGD(),
```

```
              metrics=['MSE'])
```

```
# Write the Training input and output variables, size of the batch, number of epochs
history = model.fit(X_train,y_train,
                    batch_size=1,
                    epochs=10,verbose=1)
```

```
Epoch 1/10
1689/1689 [==============================] - 2s 919us/step - loss: 0.0337 - MSE: 0.0337
Epoch 2/10
1689/1689 [==============================] - 2s 915us/step - loss: 0.0151 - MSE: 0.0151
Epoch 3/10
1689/1689 [==============================] - 2s 904us/step - loss: 0.0116 - MSE: 0.0116
Epoch 4/10
1689/1689 [==============================] - 2s 917us/step - loss: 0.0111 - MSE: 0.0111
Epoch 5/10
1689/1689 [==============================] - 2s 937us/step - loss: 0.0107 - MSE: 0.0107
Epoch 6/10
1689/1689 [==============================] - 2s 912us/step - loss: 0.0105 - MSE: 0.0105
Epoch 7/10
1689/1689 [==============================] - 2s 910us/step - loss: 0.0106 - MSE: 0.0106
Epoch 8/10
1689/1689 [==============================] - 2s 910us/step - loss: 0.0111 - MSE: 0.0111
Epoch 9/10
1689/1689 [==============================] - 2s 891us/step - loss: 0.0116 - MSE: 0.0116
Epoch 10/10
1689/1689 [==============================] - 2s 905us/step - loss: 0.0106 - MSE: 0.0106
```

```
# Write the testing input and output variables
score = model.evaluate(X_test, y_test, verbose=2)
print('Test loss:', score[0])
```

```
14/14 - 0s - loss: 0.0124 - MSE: 0.0124
Test loss: 0.012383264489471912
```

```
# Write the index of the test sample to test
print(X_test[0])
prediction = model.predict(X_test[0].reshape(1,6))
print(prediction[0]*(max_load-min_load)+min_load)
```

```
print(y_test[0]*(max_load-min_load)+min_load)
```

```
[0.74063507 0.44249807 0.37659411 0.6966991  0.82779379 0.74982169]
Load    7440.040768
dtype: float64
Load    7373.124
dtype: float64
```

# Binary Classification

---

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

import numpy as np
import matplotlib.pyplot as plt
import keras  #Keras is the deep learning library that helps you to code Deep Neural Networks with fewer lines of code
#from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import RMSprop,Adadelta,SGD,Adagrad,Adam
#import pylab as plt
#import seaborn as sns #For data visualization
import pandas as pd # For Data manipulation
```

```
diabetes_data=pd.read_csv("/content/drive/MyDrive/diabetes.csv")
diabetes_data.head()
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|

```
print(diabetes_data.shape) # details about number of samples and features
diabetes_data.describe()
```

(768, 9)

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | A |
|---|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.0000 |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.2408 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.7602 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.0000 |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.0000 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.0000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.0000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.0000 |

```
diabetes_data.isnull().any()
```

```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
diabetes_data=ms.fit_transform(diabetes_data)
```

```
print(diabetes_data)
```

```
[[0.35294118 0.74371859 0.59016393 ... 0.23441503 0.48333333 1.        ]
 [0.05882353 0.42713568 0.54098361 ... 0.11656704 0.16666667 0.        ]
 [0.47058824 0.91959799 0.52459016 ... 0.25362938 0.18333333 1.        ]
 ...
 [0.29411765 0.6080402  0.59016393 ... 0.07130658 0.15       0.        ]
 [0.05882353 0.63316583 0.49180328 ... 0.11571307 0.43333333 1.        ]
 [0.05882353 0.46733668 0.57377049 ... 0.10119556 0.03333333 0.        ]]
```

```
dataset=pd.DataFrame(data=diabetes_data[0:,0:])
```

```
X=dataset.iloc[:,0:8].values
Y=dataset.iloc[:,8:].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print(y_test[0:5])
print(X_train)
```

```
[[1.]
 [0.]
 [0.]
 [1.]
 [0.]]
[[0.41176471 0.75376884 0.63934426 ... 0.52459016 0.26216909 0.55      ]
 [0.23529412 0.48743719 0.49180328 ... 0.42026826 0.1558497  0.01666667]
 [0.         0.82914573 0.73770492 ... 0.77943368 0.14901793 0.03333333]
 ...
 [0.23529412 0.47236181 0.53278689 ... 0.3681073  0.02988898 0.        ]
 [0.64705882 0.42713568 0.60655738 ... 0.4485842  0.09479078 0.23333333]
 [0.29411765 0.68341709 0.67213115 ... 0.         0.23996584 0.8       ]]
```

```
First_Layer_Size = 32 # Number of neurons in first layer
model=Sequential()
model.add(Dense(First_Layer_Size,activation='tanh', input_shape=(8,)))
model.add(Dense(32,activation='tanh'))
model.add(Dense(32,activation='tanh'))
model.add(Dense(1,activation='sigmoid'))
```

```
model.add(Dense(1,activation='sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 32)                288
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_6 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,433
Trainable params: 2,433
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# Write the Training input and output variables, size of the batch, number of epochs
history = model.fit(X_train,y_train,batch_size=1,epochs=10,verbose=1)
```

```
Epoch 1/10
614/614 [==============================] - 1s 1ms/step - loss: 0.6481 - accuracy: 0.6529
Epoch 2/10
614/614 [==============================] - 1s 1ms/step - loss: 0.6003 - accuracy: 0.6830
Epoch 3/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5780 - accuracy: 0.7054
Epoch 4/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5081 - accuracy: 0.7720
Epoch 5/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5197 - accuracy: 0.7581
Epoch 6/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5147 - accuracy: 0.7571
Epoch 7/10
614/614 [==============================] - 1s 1ms/step - loss: 0.4946 - accuracy: 0.7423
Epoch 8/10
614/614 [==============================] - 1s 1ms/step - loss: 0.4937 - accuracy: 0.7552
```

```
Epoch 9/10
614/614 [==============================] - 1s 1ms/step - loss: 0.4781 - accuracy: 0.7632
Epoch 10/10
614/614 [==============================] - 1s 1ms/step - loss: 0.4809 - accuracy: 0.7634
```

```
# Write the testing input and output variables
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.5085 - accuracy: 0.7597
Test loss: 0.5085141062736511
Test accuracy: 0.7597402334213257
```

```
# Write the index of the test sample to test
print(X_test[0])
prediction = model.predict(X_test[0].reshape(1,8))
print("Prediction class:",np.round(prediction[0]))
print("Actual class:",y_test[0])
```

```
[0.05882353 1.         0.62295082 0.43434343 0.         0.63934426
 0.56191289 0.01666667]
Prediction class: [1.]
Actual class: [1.]
```

## *Categorical Classification*

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

import numpy as np
import matplotlib.pyplot as plt
import keras  #Keras is the deep learning library that helps you to code Deep Neural Networks with fewer lines of code
#from keras.datasets import mnist
from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.optimizers import RMSprop,Adadelta,SGD,Adagrad,Adam
#import pylab as plt
#import seaborn as sns #For data visualization
import pandas as pd # For Data manipulation

from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils


dataframe = pd.read_csv("/content/drive/MyDrive/winequality-red.csv")
dataframe.head()
```

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| **1** | 7.8 | 0.88 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 | 5 |
| **2** | 7.8 | 0.76 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.9970 | 3.26 | 0.65 | 9.8 | 5 |
| **3** | 11.2 | 0.28 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.9980 | 3.16 | 0.58 | 9.8 | 6 |

```
dataset = dataframe.values
X = dataset[0:,0:11].astype(float)
Y = dataset[0:,11]
print(X[0:])
```

```
[[ 7.4    0.7    0.    ...  3.51   0.56   9.4  ]
 [ 7.8    0.88   0.    ...  3.2    0.68   9.8  ]
 [ 7.8    0.76   0.04  ...  3.26   0.65   9.8  ]
 ...
 [ 6.3    0.51   0.13  ...  3.42   0.75  11.   ]
 [ 5.9    0.645  0.12  ...  3.57   0.71  10.2  ]
 [ 6.     0.31   0.47  ...  3.39   0.66  11.   ]]
```

```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
```

```
red_wine_data_X=ms.fit_transform(X)
print(red_wine_data_X)
```

```
[[0.24778761 0.39726027 0.         ... 0.60629921 0.13772455 0.15384615]
 [0.28318584 0.52054795 0.         ... 0.36220472 0.20958084 0.21538462]
 [0.28318584 0.43835616 0.04       ... 0.40944882 0.19161677 0.21538462]
 ...
 [0.15044248 0.26712329 0.13       ... 0.53543307 0.25149701 0.4       ]
 [0.11504425 0.35958904 0.12       ... 0.65354331 0.22754491 0.27692308]
 [0.12389381 0.13013699 0.47       ... 0.51181102 0.19760479 0.4       ]]
```

```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
print(dummy_y)
```

```
[[0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 ...
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(red_wine_data_X, dummy_y, test_size=0.2, random_state=0)
print(y_test[0:5])
print(X_train)
```

```
[[0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]]
[[0.46902655 0.28767123 0.45       ... 0.51181102 0.17365269 0.15384615]
 [0.54867257 0.09589041 0.45       ... 0.30708661 0.1257485  0.18461538]
 [0.46902655 0.15753425 0.55       ... 0.40944882 0.2754491  0.33846154]
```

```
...
[0.2920354  0.30821918 0.31       ... 0.43307087 0.21556886 0.16923077]
[0.74336283 0.23972603 0.49       ... 0.44094488 0.20958084 0.66153846]
[0.46017699 0.5890411  0.32       ... 0.4015748  0.08982036 0.15384615]]
```

```
First_Layer_Size = 32 # Number of neurons in first layer
model=Sequential()
model.add(Dense(First_Layer_Size,activation='tanh', input_shape=(11,)))
model.add(Dense(32,activation='tanh'))
model.add(Dense(32,activation='tanh'))
model.add(Dense(6,activation='softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 32)                384
_________________________________________________________________
dense_9 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_10 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 198
=================================================================
Total params: 2,694
Trainable params: 2,694
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# Write the Training input and output variables, size of the batch, number of epochs
history = model.fit(X_train,y_train,batch_size=1,epochs=10,verbose=1)
```

```
Epoch 1/10
1279/1279 [==============================] - 2s 1ms/step - loss: 1.2191 - accuracy: 0.4569
Epoch 2/10
1279/1279 [==============================] - 1s 1ms/step - loss: 1.0047 - accuracy: 0.5702
```

```
Epoch 3/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9768 - accuracy: 0.5674
Epoch 4/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9569 - accuracy: 0.5681
Epoch 5/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9165 - accuracy: 0.5975
Epoch 6/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9670 - accuracy: 0.5726
Epoch 7/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9466 - accuracy: 0.5618
Epoch 8/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9530 - accuracy: 0.5753
Epoch 9/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9377 - accuracy: 0.6004
Epoch 10/10
1279/1279 [==============================] - 1s 1ms/step - loss: 0.9659 - accuracy: 0.5721
```

```
# Write the testing input and output variables
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
10/10 [==============================] - 0s 1ms/step - loss: 0.9264 - accuracy: 0.6187
Test loss: 0.9264217615127563
Test accuracy: 0.6187499761581421
```

```
# Write the index of the test sample to test
prediction = model.predict(X_test[35].reshape(1,11))
print(prediction[0])
print(np.round(prediction[0]))
print(y_test[0])
```

```
[4.3296666e-04 7.8502595e-03 1.6516376e-01 5.1110464e-01 3.0484772e-01
 1.0600625e-02]
[0. 0. 0. 1. 0. 0.]
[0. 0. 0. 1. 0. 0.]
```