```
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.shape)
```

```
(2112, 7)
```

```
#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
```

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

```
(6, 1)
[[ 0.29852368]
 [-0.80070603]
 [-0.42843934]
 [ 0.19778899]
 [-0.43912462]
 [ 0.71304091]]
```

```
#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)
```

```
[[-2.35686084e-05]
 [-2.24660335e-05]
 [-2.40814340e-05]
 [-2.20942349e-05]
 [-3.07738077e-05]
 [-2.63635078e-05]]
[[ 0.72491797]
 [-0.58946206]
 [-0.60213211]
 [ 0.39050065]
 [-0.31213618]
 [ 0.61202274]]
```

```
#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))
```

```
0.03359470348431108
```

```
# 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    6298.687931
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)
```

```
[[6.77113098e-05]
 [6.45436728e-05]
 [6.91846293e-05]
 [6.34755162e-05]
 [8.84114491e-05]
 [7.57409012e-05]]
[[-0.03169163]
 [ 0.08800486]
 [-0.06013543]
 [ 0.0137022 ]
 [-0.01085609]
 [-0.00045358]]
```

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):
        n=np.dot(a[j,0:6],weights) #n.shape-1,1
        dw=(l*(a[j,6]-n))*a[j,0:6].T
        dw=dw.reshape((6,1))
        weights=weights+dw
print(dw)
print(weights)
```

```
[[-3.78749726e-05]
 [-3.61031244e-05]
 [-3.86990881e-05]
 [-3.55056407e-05]
 [-4.94537947e-05]
 [-4.23664018e-05]]
[[ 0.57917864]
 [-0.02667027]
 [-0.06043613]
 [ 0.18656582]
 [ 0.26074718]
 [ 0.0524128 ]]
```

```
#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))
```

```
0.009887281734308345
```

```
# 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    6942.057233
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) [[ 0.684798   -0.85649502  0.65928505  0.36607614  0.90978604 -0.7163582
  -0.67334432 -0.6591412   0.78388166]
 [-0.47069933 -0.60163822  0.81582263  0.17486459 -0.64657353  0.21051601
  -0.62166068 -0.3421558  -0.45986084]
 [-0.33229892 -0.6335112  -0.38185769 -0.93263509 -0.57131861 -0.98620545
  -0.44759659  0.09273474 -0.66121637]
 [-0.48562154 -0.62131808 -0.96330847 -0.91706107  0.33880256  0.52576331
   0.9622727   0.44698343  0.78876215]
 [-0.07494795 -0.41213321  0.26208475 -0.86634359  0.35713188  0.27014708
   0.79027673  0.23269342  0.75141678]
 [-0.99692535 -0.78258913 -0.14175653 -0.57471874  0.20659751 -0.86084259
  -0.22801057  0.46313543 -0.49682871]]
(9, 1) [[ 0.72737372]
 [ 0.69858287]
 [ 0.49060026]
 [ 0.45005994]
 [-0.38791886]
 [-0.20029799]
 [ 0.03587118]
 [-0.79852305]
 [ 0.68092439]]
```

```
#Forward pass
```

```
#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)
```

```
[[ 0.25378952]
 [ 0.12090838]
 [ 0.34659069]
 [-0.12652935]
 [-0.35875701]
 [-0.65549915]
 [-0.13187249]
 [-1.14666833]
 [ 0.86266825]]
[[ 0.76582169 -0.87150136  0.75345316  0.36980267  0.80224074 -0.80233577
  -0.70179919 -0.92980579  0.91992437]
 [-0.43210504 -0.63376477  0.87081635  0.16991704 -0.71091046  0.1672343
```

```
  -0.63948958 -0.50326763 -0.38864474]
 [-0.3384424  -0.68504441 -0.3674437  -0.9474973  -0.58787341 -0.98467845
  -0.45428697  0.04607992 -0.66261768]
 [-0.40396309 -0.63610304 -0.86613153 -0.9135409   0.22963533  0.43585033
   0.93324992  0.16918073  0.92573327]
 [-0.00525728 -0.43294698  0.34750237 -0.8655376   0.26128834  0.19307395
   0.76434155 -0.01345743  0.86730764]
 [-0.93564002 -0.80713127 -0.06333192 -0.57562888  0.11896596 -0.92957786
  -0.25207555  0.23650076 -0.39158188]]
```

```
#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))
```

```
0.030409820559406515
```

```
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    5890.992911
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()
```

	Load
0	5551.82208
1	4983.17184
2	4888.39680
3	5072.95872
4	5196.25980

```
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

```
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
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.shape)
```

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

```
First_Layer_Size = 32 # Number of neurons in first layer
model=Sequential()
model.add(Dense(First_Layer_Size,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 [==============================] - 1s 863us/step - loss: 0.0301 - MSE: 0.0301
    Epoch 2/10
    1689/1689 [==============================] - 1s 848us/step - loss: 0.0167 - MSE: 0.0167
    Epoch 3/10
    1689/1689 [==============================] - 1s 830us/step - loss: 0.0133 - MSE: 0.0133
    Epoch 4/10
    1689/1689 [==============================] - 1s 833us/step - loss: 0.0122 - MSE: 0.0122
    Epoch 5/10
    1689/1689 [==============================] - 1s 860us/step - loss: 0.0115 - MSE: 0.0115
    Epoch 6/10
    1689/1689 [==============================] - 1s 845us/step - loss: 0.0113 - MSE: 0.0113
    Epoch 7/10
    1689/1689 [==============================] - 1s 821us/step - loss: 0.0111 - MSE: 0.0111
    Epoch 8/10
    1689/1689 [==============================] - 1s 832us/step - loss: 0.0109 - MSE: 0.0109
    Epoch 9/10
    1689/1689 [==============================] - 1s 847us/step - loss: 0.0109 - MSE: 0.0109
    Epoch 10/10
    1689/1689 [==============================] - 1s 852us/step - loss: 0.0109 - MSE: 0.0109
```

```
# 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: 0.0122 - MSE: 0.0122
    Test loss: 0.012188175693154335
```

```
# 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    7465.097098
    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
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

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

```
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.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: 0.6357 - accuracy: 0.6515
Epoch 2/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5761 - accuracy: 0.7020
Epoch 3/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5548 - accuracy: 0.7280
Epoch 4/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5275 - accuracy: 0.7215
Epoch 5/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5113 - accuracy: 0.7508
Epoch 6/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5095 - accuracy: 0.7573
Epoch 7/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5088 - accuracy: 0.7622
Epoch 8/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5052 - accuracy: 0.7638
Epoch 9/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5071 - accuracy: 0.7622
```

```
Epoch 10/10
614/614 [==============================] - 1s 1ms/step - loss: 0.5041 - accuracy: 0.7459
```

```
# 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: 0.4412 - accuracy: 0.7662
Test loss: 0.4412461221218109
Test accuracy: 0.7662337422370911
```

```
# 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 70/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6919 - accuracy: 0.7162
Epoch 71/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6857 - accuracy: 0.7146
Epoch 72/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6775 - accuracy: 0.7224
Epoch 73/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6701 - accuracy: 0.7193
Epoch 74/100
```

```
Epoch 74/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6613 - accuracy: 0.7248
Epoch 75/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6623 - accuracy: 0.7185
Epoch 76/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6651 - accuracy: 0.7170
Epoch 77/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6461 - accuracy: 0.7232
Epoch 78/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6402 - accuracy: 0.7342
Epoch 79/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6387 - accuracy: 0.7389
Epoch 80/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6296 - accuracy: 0.7467
Epoch 81/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6287 - accuracy: 0.7373
Epoch 82/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6255 - accuracy: 0.7435
Epoch 83/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6135 - accuracy: 0.7475
Epoch 84/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6152 - accuracy: 0.7459
Epoch 85/100
1279/1279 [==============================] - 2s 1ms/step - loss: 0.6021 - accuracy: 0.7482
Epoch 86/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.6021 - accuracy: 0.7428
Epoch 87/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5914 - accuracy: 0.7529
Epoch 88/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5835 - accuracy: 0.7576
Epoch 89/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5830 - accuracy: 0.7608
Epoch 90/100

1279/1279 [==============================] - 1s 1ms/step - loss: 0.5847 - accuracy: 0.7482
Epoch 91/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5604 - accuracy: 0.7694
Epoch 92/100
1279/1279 [==============================] - 2s 1ms/step - loss: 0.5660 - accuracy: 0.7717
Epoch 93/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5565 - accuracy: 0.7670
Epoch 94/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5547 - accuracy: 0.7725
Epoch 95/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5507 - accuracy: 0.7670
```

```
Epoch 96/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5391 - accuracy: 0.7842
Epoch 97/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5302 - accuracy: 0.7811
Epoch 98/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5271 - accuracy: 0.7897
Epoch 99/100
1279/1279 [==============================] - 1s 1ms/step - loss: 0.5286 - accuracy: 0.7811
```

```
# 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: 1.0735 - accuracy: 0.6406
Test loss: 1.0734533071517944
Test accuracy: 0.640625
```

```
# Write the index of the test sample to test
prediction = model.predict(X_test[9].reshape(1,11))
print(prediction[0])
print(np.round(prediction[0]))
print(y_test[0])
```

```
[5.2537644e-06 5.4543203e-04 6.5502274e-01 3.4437990e-01 4.5258694e-05
 1.4017568e-06]
[0. 0. 1. 0. 0. 0.]
[0. 0. 0. 1. 0. 0.]
```