In [1]:

```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error,mean_squared_error
from sklearn.metrics import confusion_matrix , classification_report,accuracy_score


import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

Binary Classifier

In [2]:

```
df = pd.read_csv("kaggle_diabetes.csv")
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	2	138	62	35	0	33.6	0.12
1	0	84	82	31	125	38.2	0.23
2	0	145	0	0	0	44.2	0.63
3	0	135	68	42	250	42.3	0.36
4	1	139	62	41	480	40.7	0.53

In [3]:

```
df.shape
```

Out[3]:

```
(2000, 9)
```

In [4]:

```
X = df.drop('Outcome',axis=1)
y = df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=5)
```

Apply ANN

In [5]:

```
classifier = Sequential()
##input 1st layer
classifier.add(Dense(16,activation='relu',input_dim=8))


## second hidden layer
classifier.add(Dense(8,activation='relu'))


## output layer
classifier.add(Dense(1,activation='sigmoid'))
```

In [6]:

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

In [7]:

```
classifier.fit(X_train, y_train, epochs=150)
```

```
Epoch 1/150
50/50 [==============================] - 1s 1ms/step - loss: 6.1302 - accu
racy: 0.4691
Epoch 2/150
50/50 [==============================] - 0s 1ms/step - loss: 1.5064 - accu
racy: 0.6145
Epoch 3/150
50/50 [==============================] - 0s 1ms/step - loss: 1.1917 - accu
racy: 0.6264
Epoch 4/150
50/50 [==============================] - 0s 1ms/step - loss: 0.9443 - accu
racy: 0.6452
Epoch 5/150
50/50 [==============================] - 0s 1ms/step - loss: 0.8631 - accu
racy: 0.6444
Epoch 6/150
50/50 [==============================] - 0s 2ms/step - loss: 0.8198 - accu
racy: 0.6555
Epoch 7/150
```

In [8]:

```
classifier.evaluate(X_test, y_test)
```

```
13/13 [==============================] - 0s 1ms/step - loss: 0.4997 - accura
cy: 0.7775
```

Out[8]:

```
[0.4997430741786957, 0.7774999737739563]
```

In [9]:

```
y_pred = classifier.predict(X_test)
```

In [10]:

```
yp = classifier.predict(X_test)
yp[:5]
```

Out[10]:

```
array([[0.1072953 ],
       [0.03356129],
       [0.20200807],
       [0.16860384],
       [0.50361   ]], dtype=float32)
```

In [11]:

```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [12]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.92      0.85       269
           1       0.74      0.49      0.59       131

    accuracy                           0.78       400
   macro avg       0.77      0.70      0.72       400
weighted avg       0.77      0.78      0.76       400
```

Multi Classification Using ANN

In [13]:

```
data = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a
data.shape
```

Out[13]:

```
(150, 5)
```

In [14]:

```
data.head()
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Split in to X and y

In [15]:

```
X = data.drop('species',axis=1)
y = data['species']
```

Encoding target variable Using Label or Dummy

imp1:- If you use dummy then in loss function you use categorical_crossentropy

imp2:- if you use label_encoding then in loss function you use sparse_categorical_crossentropy

If your targets are **one-hot encoded**, use `categorical_crossentropy`.

- Examples of one-hot encodings:
    - `[1,0,0]`
    - `[0,1,0]`
    - `[0,0,1]`

But if your targets are **integers**, use `sparse_categorical_crossentropy`.

- Examples of integer encodings (*for the sake of completion*):
    - `1`
    - `2`
    - `3`

In [16]:

```
## label

lb = LabelEncoder()
y_enc = lb.fit_transform(y)
y_enc
```

Out[16]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [17]:

```
# dummy
# y_dummy = pd.get_dummies(y).values
```

In [18]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y_enc,test_size=0.25,random_state=4)
```

In [19]:

```
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

In [20]:

```
X.shape
```

Out[20]:

```
(150, 4)
```

Apply ANN

In [21]:

```
classifier = Sequential()
classifier.add(Dense(10,input_dim = 4,activation = "relu"))

classifier.add(Dense(3,activation = "softmax"))
```

In [22]:

```
## if target dummy encoding use categorical_crossentropy if use label encoding use sparse_c
classifier.compile(optimizer = 'adam' , loss = 'sparse_categorical_crossentropy',
                   metrics = ['accuracy'] )
```

In [23]:

```
classifier.fit(X_train_scaled , y_train ,epochs = 100)
```

```
Epoch 1/100
4/4 [==============================] - 0s 2ms/step - loss: 0.8949 - accura
cy: 0.6033
Epoch 2/100
4/4 [==============================] - 0s 2ms/step - loss: 0.9165 - accura
cy: 0.5705
Epoch 3/100
4/4 [==============================] - 0s 2ms/step - loss: 0.8442 - accura
cy: 0.6268
Epoch 4/100
4/4 [==============================] - 0s 4ms/step - loss: 0.8412 - accura
cy: 0.6121
Epoch 5/100
4/4 [==============================] - 0s 4ms/step - loss: 0.8268 - accura
cy: 0.5933
Epoch 6/100
4/4 [==============================] - 0s 3ms/step - loss: 0.8125 - accura
cy: 0.6260
Epoch 7/100
```

In [24]:

```
y_pred = classifier.predict(X_test_scaled)
# y_test= np.argmax(y_test,axis=1) # when use dummy encoding in target
y_pred = np.argmax(y_pred,axis=1)
```

In [25]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.88      0.93         8
           2       0.92      1.00      0.96        12

    accuracy                           0.97        38
   macro avg       0.97      0.96      0.96        38
weighted avg       0.98      0.97      0.97        38
```

In [ ]:

In [ ]:

Regression

In [26]:

```
df=pd.read_csv('https://raw.githubusercontent.com/krishnaik06/Keras-Tuner/main/Real_Combine
```

In [27]:

```
df.head()
```

Out[27]:

	T	TM	Tm	SLP	H	VV	V	VM	PM 2.5
0	7.4	9.8	4.8	1017.6	93.0	0.5	4.3	9.4	219.720833
1	7.8	12.7	4.4	1018.5	87.0	0.6	4.4	11.1	182.187500
2	6.7	13.4	2.4	1019.4	82.0	0.6	4.8	11.1	154.037500
3	8.6	15.5	3.3	1018.7	72.0	0.8	8.1	20.6	223.208333
4	12.4	20.9	4.4	1017.3	61.0	1.3	8.7	22.2	200.645833

In [28]:

```
df.dropna(inplace=True)
```

In [29]:

```
X=df.drop('PM 2.5',axis=1) ## independent features
y=df['PM 2.5'] ## dependent features
```

In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [31]:

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [32]:

```
X.shape
```

Out[32]:

```
(1092, 8)
```

Apply ANN

In [33]:

```
classifier = Sequential()
classifier.add(Dense(10,input_dim = 8,activation = "relu"))

## second hidden layer
classifier.add(Dense(8,activation='relu'))

classifier.add(Dense(1,activation = "linear"))
```

In [34]:

```
## if target dummy encoding use categorical_crossentropy if use label encoding use sparse_c
classifier.compile(optimizer = 'adam' , loss = 'mse',
                   metrics = ['mse'] )
```

In [35]:

```
classifier.fit(X_train , y_train ,epochs = 100)
```

```
Epoch 1/100
24/24 [==============================] - 1s 2ms/step - loss: 18824.7756 -
mse: 18824.7756
Epoch 2/100
24/24 [==============================] - 0s 2ms/step - loss: 20844.0150 -
mse: 20844.0150
Epoch 3/100
24/24 [==============================] - 0s 2ms/step - loss: 21034.4066 -
mse: 21034.4066
Epoch 4/100
24/24 [==============================] - 0s 2ms/step - loss: 21785.1067 -
mse: 21785.1067
Epoch 5/100
24/24 [==============================] - 0s 2ms/step - loss: 16766.3743 -
mse: 16766.3743
Epoch 6/100
24/24 [==============================] - 0s 2ms/step - loss: 19805.6353 -
mse: 19805.6353
Epoch 7/100
```

In [36]:

```
classifier.evaluate(X_test, y_test)
```

```
11/11 [==============================] - 0s 2ms/step - loss: 3349.8369 - ms
e: 3349.8369
```

Out[36]:

```
[3349.8369140625, 3349.8369140625]
```

In [37]:

```
y_pred = classifier.predict(X_test)
```

In [38]:

```
print("MAE",mean_absolute_error(y_test,y_pred))
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",mean_squared_error(y_test,y_pred,squared=False))
```

```
MAE 40.52856917885261
MSE 3349.837197332338
RMSE 57.87777809602177
```