Artificial Neural Network

Importing the libraries

```
import numpy as np
import tensorflow as tf
import pandas as pd
#these libraries are needed to make the ANN model
```

Part 1 - Data Preprocessing

Importing the dataset

```
dataset = pd.read_csv("transcritical.csv")
#this function creates a datadframe named as dataset
# all the values from the file are read and stored in the dataset variable as pandas dataframe
X = dataset.iloc[:,:-1].values
# creating independent variable (input features)
y = dataset.iloc[:,-1].values
# creating dependent variable COP
```

```
print(y)
```

```
[0.4436 0.4443 0.445  ... 2.518  2.518  2.518 ]
```

Splitting the dataset into the Training set and Test set

✓ 0s completed at 5:16 PM

```
#this is done in order to keep some data separate so as to test our model on that new data
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)

# 2 variables of independent (features) and 2 variables of dependent variable created.
```

Feature scaling

```
#this is done in order to bring all the features on par with each other so that model doesn't discriminate
# (value - mean)/ Standard Deviation
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_y = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test=sc.transform(X_test)
y_train = sc_y.fit_transform(y_train.reshape(len(y_train),1))
y_test = sc_y.transform(y_test.reshape(len(y_test),1))


print(X_train)
```

```
[[ 0.83156177  0.          0.          0.          0.7333704   0.84391893]
 [ 1.48335211  0.          0.          0.          0.87806972  1.41279926]
 [-0.68730723  0.          0.          0.         -0.11070899 -0.64719911]
 ...
 [-0.59447649  0.          0.          0.          0.00703654 -0.54820892]
 [ 1.68037055  0.          0.          0.          0.91211662  1.57820059]
 [-0.77816286  0.          0.          0.         -0.250585   -0.7461893 ]]
```

Part 2 - Building the ANN

Initializing the ANN

```
ann = tf.keras.models.Sequential()
```

Adding the input layer and the first hidden layer

```
ann.add(tf.keras.layers.Dense(units = 6, activation= "relu"))
#rectifier linear unit activation function used to break the linearity betwn input and 1st hidden layer
```

Adding the second hidden layer

```
 ann.add(tf.keras.layers.Dense(units = 6, activation= "relu"))
```

Adding the output layer

```
ann.add(tf.keras.layers.Dense(units = 1, activation="linear"))
#no actiivation function in the output layer if we are doing continous prediction(regression) or use linear
```

Part 3 - Training the ANN

Compiling the ANN

```
ann.compile(optimizer="adam", loss = "mean_squared_logarithmic_error", metrics=['mse'])
#optimizer updates all the weights in the network during backpropagation
```

Training the ANN model on the Training set

Training the ANN model on the Training set

```
ann.fit(X_train , y_train ,validation_data=(X_test, y_test), batch_size = 32 , epochs = 100)
```

```
Epoch 1/100
250/250 [==============================] - 1s 3ms/step - loss: 0.0382 - mse: 0.7273 - val_loss: 0.0013 - val_mse: 0
Epoch 2/100
250/250 [==============================] - 1s 2ms/step - loss: 3.9429e-04 - mse: 0.3522 - val_loss: 1.6219e-04 - va
Epoch 3/100
250/250 [==============================] - 1s 4ms/step - loss: 1.1000e-04 - mse: 0.3045 - val_loss: 8.0348e-05 - va
Epoch 4/100
250/250 [==============================] - 1s 5ms/step - loss: 5.3005e-05 - mse: 0.2920 - val_loss: 3.4872e-05 - va
Epoch 5/100
250/250 [==============================] - 1s 5ms/step - loss: 2.5530e-05 - mse: 0.2838 - val_loss: 1.9517e-05 - va
Epoch 6/100
250/250 [==============================] - 1s 4ms/step - loss: 1.7007e-05 - mse: 0.2784 - val_loss: 1.5602e-05 - va
Epoch 7/100
250/250 [==============================] - 1s 4ms/step - loss: 1.3894e-05 - mse: 0.2751 - val_loss: 1.5860e-05 - va
Epoch 8/100
250/250 [==============================] - 1s 4ms/step - loss: 1.2446e-05 - mse: 0.2732 - val_loss: 1.2215e-05 - va
Epoch 9/100
250/250 [==============================] - 1s 4ms/step - loss: 1.1350e-05 - mse: 0.2694 - val_loss: 1.0884e-05 - va
Epoch 10/100
250/250 [==============================] - 1s 4ms/step - loss: 1.0649e-05 - mse: 0.2674 - val_loss: 1.1359e-05 - va
Epoch 11/100
250/250 [==============================] - 1s 5ms/step - loss: 1.0529e-05 - mse: 0.2629 - val_loss: 8.7898e-06 - va
Epoch 12/100
250/250 [==============================] - 1s 5ms/step - loss: 9.1946e-06 - mse: 0.2598 - val_loss: 8.4490e-06 - va
Epoch 13/100
250/250 [==============================] - 1s 4ms/step - loss: 9.4260e-06 - mse: 0.2568 - val_loss: 9.8132e-06 - va
Epoch 14/100
250/250 [==============================] - 2s 6ms/step - loss: 8.8990e-06 - mse: 0.2542 - val_loss: 8.6868e-06 - va
Epoch 15/100
250/250 [==============================] - 1s 5ms/step - loss: 8.5707e-06 - mse: 0.2521 - val_loss: 9.4373e-06 - va
Epoch 16/100
250/250 [==============================] - 1s 6ms/step - loss: 9.0501e-06 - mse: 0.2506 - val_loss: 8.5163e-06 - va
Epoch 17/100
250/250 [==============================] - 1s 5ms/step - loss: 8.2067e-06 - mse: 0.2485 - val_loss: 7.1854e-06 - va
Epoch 18/100
250/250 [==============================] - 1s 4ms/step - loss: 8.7614e-06 - mse: 0.2478 - val_loss: 7.9197e-06 - va
Epoch 19/100
```

```
Epoch 19/100
250/250 [==============================] - 1s 5ms/step - loss: 8.1165e-06 - mse: 0.2462 - val_loss: 1.1230e-05 - va
Epoch 20/100
250/250 [==============================] - 1s 6ms/step - loss: 7.7084e-06 - mse: 0.2443 - val_loss: 8.0995e-06 - va
Epoch 21/100
250/250 [==============================] - 1s 3ms/step - loss: 6.9518e-06 - mse: 0.2422 - val_loss: 6.0874e-06 - va
Epoch 22/100
250/250 [==============================] - 1s 4ms/step - loss: 7.4211e-06 - mse: 0.2412 - val_loss: 1.1306e-05 - va
Epoch 23/100
250/250 [==============================] - 1s 5ms/step - loss: 6.6992e-06 - mse: 0.2396 - val_loss: 6.8506e-06 - va
Epoch 24/100
250/250 [==============================] - 1s 3ms/step - loss: 6.5045e-06 - mse: 0.2374 - val_loss: 5.3655e-06 - va
Epoch 25/100
250/250 [==============================] - 0s 2ms/step - loss: 6.5882e-06 - mse: 0.2361 - val_loss: 5.4477e-06 - va
Epoch 26/100
250/250 [==============================] - 0s 2ms/step - loss: 6.0951e-06 - mse: 0.2354 - val_loss: 4.8604e-06 - va
Epoch 27/100
250/250 [==============================] - 0s 2ms/step - loss: 6.8963e-06 - mse: 0.2331 - val_loss: 4.7922e-06 - va
Epoch 28/100
250/250 [==============================] - 0s 2ms/step - loss: 7.0441e-06 - mse: 0.2314 - val_loss: 4.4476e-06 - va
Epoch 29/100
250/250 [==============================] - 1s 2ms/step - loss: 5.6283e-06 - mse: 0.2302 - val_loss: 4.9971e-06 - va
```

Predicting the results of the Test set

```
y_pred = ann.predict(X_test)
print(y_pred)
```

```
[[ 0.19459417]
 [-1.1043317 ]
 [ 0.16207054]
 ...
 [ 0.17718947]
 [ 0.42381835]
 [ 0.5131429 ]]
```

```
y_test = sc_y.inverse_transform(y_test.reshape(len(y_test),1))
y_pred = sc_y.inverse_transform(y_pred.reshape(len(y_pred),1))
```

```
np.set_printoptions(precision=2)
print(np.concatenate((y_test, y_pred),1))
```

```
[[2.56 2.56]
 [1.17 1.74]
 [2.54 2.54]
 ...
 [2.55 2.55]
 [2.71 2.71]
 [2.77 2.76]]
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.8326430313563113
```