```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
```

```
df=pd.read_csv("/content/drug200.csv")
```

```
# Task 1 : Read the dataset and do data pre-processing
```

```
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['BP'] = label_encoder.fit_transform(df['BP'])
df['Cholesterol'] = label_encoder.fit_transform(df['Cholesterol'])
df['Drug'] = label_encoder.fit_transform(df['Drug'])
print(df.head())
```

```
   Age  Sex  BP  Cholesterol  Na_to_K  Drug
0   23    0   0            0   25.355     0
1   47    1   1            0   13.093     3
2   47    1   1            0   10.114     3
3   28    0   2            0    7.798     4
4   61    0   1            0   18.043     0
```

```
# Scale numerical variables
scaler = StandardScaler()
df[['Age', 'Na_to_K']] = scaler.fit_transform(df[['Age', 'Na_to_K']])
```

```
# Separate features and labels
x = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']]
y = df['Drug']
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,random_state=42)
print(X_train.shape)
print(y_test.shape)
```

```
(160, 5)
(40,)
```

```
# Task 2 : Build the ANN model with (input layer, min 3 hidden layers & output layer)
```

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

```
# Define the model architecture
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(5,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

```
x = df.iloc[:,0:5]
y = df.iloc[:,5:]
print(x)
print(y)
```

```
          Age  Sex  BP  Cholesterol   Na_to_K
0   -1.291591    0   0            0  1.286522
1    0.162699    1   1            0 -0.415145
2    0.162699    1   1            0 -0.828558
```

```
3   -0.988614    0   2           0 -1.149963
4    1.011034    0   1           0  0.271794
..        ...  ...  ..         ...       ...
195  0.708057    0   1           0 -0.626917
196 -1.715759    1   1           0 -0.565995
197  0.465676    1   2           0 -0.859089
198 -1.291591    1   2           1 -0.286500
199 -0.261469    0   1           1 -0.657170

[200 rows x 5 columns]
     Drug
0       0
1       3
2       3
3       4
4       0
..    ...
195     3
196     3
197     4
198     4
199     4

[200 rows x 1 columns]
```

```
# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
model.fit(X_train, y_train_encoded, epochs=20, batch_size=20,validation_data=(X_test, y_test_encoded))
```

```
Epoch 1/20
8/8 [==============================] - 1s 25ms/step - loss: 1.4942 - accuracy: 0.4750 - val_loss: 1.4112 - val_accuracy: 0.4750
Epoch 2/20
8/8 [==============================] - 0s 7ms/step - loss: 1.2593 - accuracy: 0.6125 - val_loss: 1.2172 - val_accuracy: 0.5000
Epoch 3/20
8/8 [==============================] - 0s 4ms/step - loss: 1.0438 - accuracy: 0.6313 - val_loss: 1.0103 - val_accuracy: 0.5500
Epoch 4/20
8/8 [==============================] - 0s 4ms/step - loss: 0.8401 - accuracy: 0.7125 - val_loss: 0.8435 - val_accuracy: 0.6500
Epoch 5/20
8/8 [==============================] - 0s 7ms/step - loss: 0.6998 - accuracy: 0.7188 - val_loss: 0.7497 - val_accuracy: 0.6500
Epoch 6/20
8/8 [==============================] - 0s 4ms/step - loss: 0.6113 - accuracy: 0.7125 - val_loss: 0.6578 - val_accuracy: 0.7000
Epoch 7/20
8/8 [==============================] - 0s 5ms/step - loss: 0.5378 - accuracy: 0.7812 - val_loss: 0.5956 - val_accuracy: 0.8500
Epoch 8/20
8/8 [==============================] - 0s 6ms/step - loss: 0.4643 - accuracy: 0.8625 - val_loss: 0.5473 - val_accuracy: 0.8500
Epoch 9/20
8/8 [==============================] - 0s 5ms/step - loss: 0.4053 - accuracy: 0.8625 - val_loss: 0.4892 - val_accuracy: 0.8750
Epoch 10/20
8/8 [==============================] - 0s 4ms/step - loss: 0.3608 - accuracy: 0.8875 - val_loss: 0.4433 - val_accuracy: 0.8500
Epoch 11/20
8/8 [==============================] - 0s 5ms/step - loss: 0.3240 - accuracy: 0.8938 - val_loss: 0.4050 - val_accuracy: 0.9000
Epoch 12/20
8/8 [==============================] - 0s 4ms/step - loss: 0.2788 - accuracy: 0.9062 - val_loss: 0.3262 - val_accuracy: 0.9000
Epoch 13/20
8/8 [==============================] - 0s 4ms/step - loss: 0.2341 - accuracy: 0.9312 - val_loss: 0.3145 - val_accuracy: 0.8750
Epoch 14/20
8/8 [==============================] - 0s 4ms/step - loss: 0.1993 - accuracy: 0.9563 - val_loss: 0.2407 - val_accuracy: 0.9250
Epoch 15/20
8/8 [==============================] - 0s 4ms/step - loss: 0.1721 - accuracy: 0.9688 - val_loss: 0.2230 - val_accuracy: 0.9250
Epoch 16/20
8/8 [==============================] - 0s 7ms/step - loss: 0.1482 - accuracy: 0.9688 - val_loss: 0.1949 - val_accuracy: 0.9500
Epoch 17/20
8/8 [==============================] - 0s 5ms/step - loss: 0.1314 - accuracy: 0.9688 - val_loss: 0.1788 - val_accuracy: 0.9000
Epoch 18/20
8/8 [==============================] - 0s 6ms/step - loss: 0.1291 - accuracy: 0.9688 - val_loss: 0.1594 - val_accuracy: 0.9500
Epoch 19/20
8/8 [==============================] - 0s 5ms/step - loss: 0.1160 - accuracy: 0.9688 - val_loss: 0.1553 - val_accuracy: 0.9000
Epoch 20/20
8/8 [==============================] - 0s 5ms/step - loss: 0.1055 - accuracy: 0.9563 - val_loss: 0.1135 - val_accuracy: 1.0000
<keras.callbacks.History at 0x7f17f1d85450>
```

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

```
2/2 [==============================] - 0s 3ms/step
array([[1.75110588e-04, 1.73789554e-03, 1.44960586e-05, 5.00680618e-02,
        9.48004484e-01],
       [7.82621980e-01, 2.16004521e-01, 3.51645227e-04, 8.19196168e-04,
```

```
        2.02563679e-04],
       [1.31217871e-06, 2.48581528e-05, 1.79198825e-08, 5.90924686e-03,
        9.94064510e-01],
       [1.10933697e-03, 1.77982450e-02, 4.35904972e-03, 6.03691101e-01,
        3.73042256e-01],
       [9.99999940e-01, 7.58614950e-17, 1.11155350e-15, 4.90187216e-14,
        2.88596865e-14],
       [9.97639656e-01, 5.33648767e-04, 1.25601247e-03, 4.66648547e-04,
        1.04025996e-04],
       [9.99999940e-01, 2.11565112e-08, 3.85568066e-09, 3.67226649e-09,
        2.14750395e-09],
       [5.58930449e-03, 7.64277065e-05, 2.70903138e-06, 1.96509212e-02,
        9.74680543e-01],
       [7.11830258e-02, 7.97181308e-01, 1.06061690e-01, 1.54021150e-02,
        1.01719005e-02],
       [4.90675075e-06, 8.19227716e-06, 1.01620469e-06, 6.60446566e-03,
        9.93381321e-01],
       [1.19740621e-03, 9.79822159e-01, 3.71804740e-03, 1.43727325e-02,
        8.89612304e-04],
       [3.19074001e-03, 2.98283119e-02, 2.55257346e-05, 1.16510354e-01,
        8.50445032e-01],
       [9.99771595e-01, 2.05337278e-06, 1.19692015e-06, 1.05952800e-04,
        1.19230383e-04],
       [7.37087475e-03, 9.77415800e-01, 1.09454768e-03, 1.33749107e-02,
        7.43944314e-04],
       [2.31595943e-03, 3.57543305e-02, 9.58233595e-01, 2.74140318e-03,
        9.54734918e-04],
       [9.99958932e-01, 5.80015769e-09, 1.28344291e-09, 4.28444127e-06,
        3.67586545e-05],
       [6.05987292e-03, 1.19903535e-02, 9.80454624e-01, 5.73633006e-04,
        9.21427389e-04],
       [5.87925115e-07, 8.99137740e-06, 1.96542231e-07, 1.05692893e-02,
        9.89421010e-01],
       [9.16689823e-05, 4.04868610e-02, 1.29135238e-04, 7.11071789e-01,
        2.48220518e-01],
       [9.99999940e-01, 2.19593251e-13, 1.69598005e-12, 3.67178815e-10,
        1.16479504e-09],
       [4.69511189e-02, 1.24038823e-01, 7.73378849e-01, 9.49087460e-03,
        4.61404324e-02],
       [1.22326771e-02, 2.10736250e-03, 2.44371686e-02, 8.00465569e-02,
        8.81176233e-01],
       [1.18224889e-04, 5.85588487e-03, 1.48123223e-03, 2.45662704e-01,
        7.46882021e-01],
       [9.99999940e-01, 2.37163072e-13, 2.00672508e-12, 3.03997938e-11,
        3.64928400e-11],
       [9.99999940e-01, 2.31263246e-15, 9.80369282e-14, 6.24916915e-12,
        8.22816537e-11],
       [9.99999940e-01, 1.22260275e-13, 8.69180975e-13, 1.49254706e-11,
        1.57134045e-11],
       [3.38916754e-04, 1.96966045e-02, 9.79244243e-04, 5.35575211e-01,
        4.43410069e-01],
       [1.20229915e-05, 1.17763320e-05, 7.55176188e-09, 4.47146734e-03,
        9.95504737e-01],
       [9.99999940e-01, 3.26769545e-09, 8.39940673e-10, 1.10017750e-09,
```

```
comp = pd.DataFrame(y_test_encoded) # Creating a dataframe
comp.columns = ['Actual Value'] # Changing the column name
comp
```

	Actual Value
0	4
1	0
2	4
3	3
4	0
5	0
6	0
7	4
8	1
9	4
10	1
11	4
12	0
13	1
14	2
15	0
16	2
17	4

```
# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                384

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 5)                 165

=================================================================
Total params: 19,205
Trainable params: 19,205
Non-trainable params: 0
_________________________________________________________________
```

```
# Task 3 : Test the model with random data
```

```
# Generate random data for testing
random_data = np.random.rand(1, 5)
random_data
```

```
array([[0.5902534 , 0.76205588, 0.66713313, 0.58979546, 0.59070951]])
```

37 1

```
# Make predictions
predictions = model.predict(random_data)
predictions
```

```
1/1 [==============================] - 0s 50ms/step
array([[9.9999714e-01, 2.3604699e-08, 2.1938440e-07, 8.7394307e-07,
        1.7675379e-06]], dtype=float32)
```

```
# Get the predicted drug class
predicted_class = np.argmax(predictions)
```

```
# Print the predicted class
print("Predicted Drug Class :", predicted_class)
```

```
Predicted Drug Class : 0
```