```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten,Dense
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
```

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

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

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	DrugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	DrugY

```
df.tail()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

```
df.describe(include='all')
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
count	200.000000	200	200	200	200.000000	200
unique	NaN	2	3	2	NaN	5
top	NaN	M	HIGH	HIGH	NaN	DrugY
freq	NaN	104	77	103	NaN	91
mean	44.315000	NaN	NaN	NaN	16.084485	NaN
std	16.544315	NaN	NaN	NaN	7.223956	NaN
min	15.000000	NaN	NaN	NaN	6.269000	NaN
25%	31.000000	NaN	NaN	NaN	10.445500	NaN
50%	45.000000	NaN	NaN	NaN	13.936500	NaN
75%	58.000000	NaN	NaN	NaN	19.380000	NaN

```
df.isnull().sum()
```

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
```

```
Drug           0
dtype: int64
```

```
df.shape
```

```
(200, 6)
```

```
df.info()
```

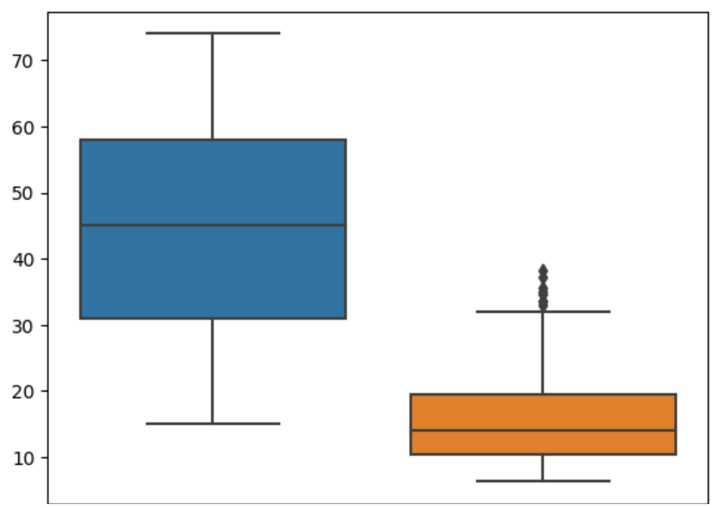
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

```
df['Drug'].value_counts()
```

```
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

```
sns.boxplot(df)
```

```
<Axes: >
```

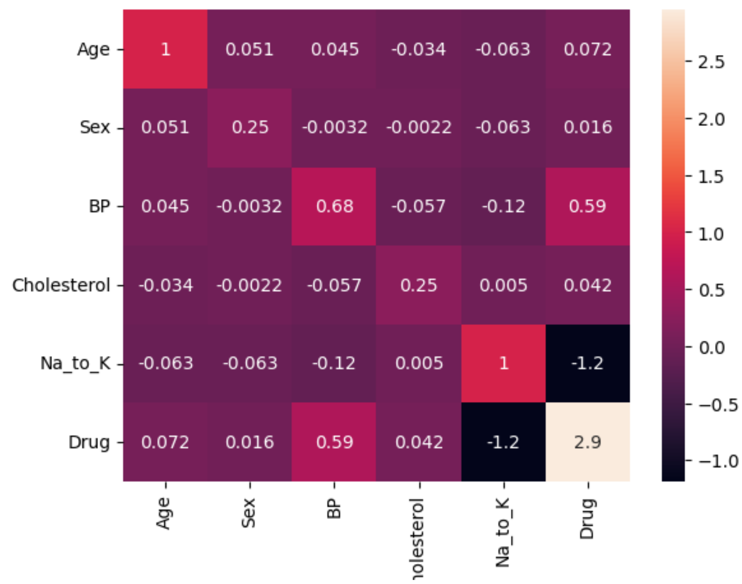



```
sns.heatmap(df.corr(),annot = True)
```

```
<Axes: >
```


```
sns.heatmap(df.cov(),annot=True)
```

```
<Axes: >
```




```
df.shape
```

```
(200, 6)
```

```
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)

```
# 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 [==============================] - 2s 48ms/step - loss: 1.5276 - accuracy: 0.4563 - val_loss: 1.4343 - val_accuracy: 0.5750
Epoch 2/20
8/8 [==============================] - 0s 11ms/step - loss: 1.3258 - accuracy: 0.6562 - val_loss: 1.2539 - val_accuracy: 0.6000
Epoch 3/20
8/8 [==============================] - 0s 8ms/step - loss: 1.1038 - accuracy: 0.6687 - val_loss: 1.0544 - val_accuracy: 0.6000
Epoch 4/20
8/8 [==============================] - 0s 8ms/step - loss: 0.8836 - accuracy: 0.7000 - val_loss: 0.8803 - val_accuracy: 0.7000
Epoch 5/20
8/8 [==============================] - 0s 8ms/step - loss: 0.6925 - accuracy: 0.7563 - val_loss: 0.7380 - val_accuracy: 0.7250
Epoch 6/20
8/8 [==============================] - 0s 9ms/step - loss: 0.5494 - accuracy: 0.8625 - val_loss: 0.6239 - val_accuracy: 0.8250
Epoch 7/20
8/8 [==============================] - 0s 8ms/step - loss: 0.4428 - accuracy: 0.8938 - val_loss: 0.5383 - val_accuracy: 0.8500
Epoch 8/20
8/8 [==============================] - 0s 10ms/step - loss: 0.3550 - accuracy: 0.9187 - val_loss: 0.4465 - val_accuracy: 0.8750
Epoch 9/20
8/8 [==============================] - 0s 12ms/step - loss: 0.3044 - accuracy: 0.9125 - val_loss: 0.3764 - val_accuracy: 0.8750
Epoch 10/20
8/8 [==============================] - 0s 8ms/step - loss: 0.2504 - accuracy: 0.9187 - val_loss: 0.3313 - val_accuracy: 0.8750
Epoch 11/20
8/8 [==============================] - 0s 11ms/step - loss: 0.2223 - accuracy: 0.9250 - val_loss: 0.2773 - val_accuracy: 0.9000
Epoch 12/20
8/8 [==============================] - 0s 10ms/step - loss: 0.1818 - accuracy: 0.9438 - val_loss: 0.2641 - val_accuracy: 0.9000
```

```
Epoch 13/20
8/8 [==============================] - 0s 8ms/step - loss: 0.1692 - accuracy: 0.9438 - val_loss: 0.2183 - val_accuracy: 0.9250
Epoch 14/20
8/8 [==============================] - 0s 10ms/step - loss: 0.1350 - accuracy: 0.9750 - val_loss: 0.1990 - val_accuracy: 0.9000
Epoch 15/20
8/8 [==============================] - 0s 8ms/step - loss: 0.1202 - accuracy: 0.9688 - val_loss: 0.1813 - val_accuracy: 0.9000
Epoch 16/20
8/8 [==============================] - 0s 11ms/step - loss: 0.1061 - accuracy: 0.9875 - val_loss: 0.1632 - val_accuracy: 0.9250
Epoch 17/20
8/8 [==============================] - 0s 10ms/step - loss: 0.0917 - accuracy: 0.9750 - val_loss: 0.1426 - val_accuracy: 0.9250
Epoch 18/20
8/8 [==============================] - 0s 10ms/step - loss: 0.0840 - accuracy: 0.9750 - val_loss: 0.1382 - val_accuracy: 0.9250
Epoch 19/20
8/8 [==============================] - 0s 8ms/step - loss: 0.0727 - accuracy: 0.9750 - val_loss: 0.1192 - val_accuracy: 0.9750
Epoch 20/20
8/8 [==============================] - 0s 10ms/step - loss: 0.0633 - accuracy: 0.9812 - val_loss: 0.1088 - val_accuracy: 0.9500
<keras.callbacks.History at 0x7fa4318fc490>
```

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

```
       [4.77323774e-04, 9.86557543e-01, 3.29303148e-04, 1.26218796e-02,
        1.38284076e-05],
       [1.26239341e-02, 8.35227408e-03, 7.18321680e-05, 5.05299389e-01,
        4.73652631e-01],
       [9.99876738e-01, 2.30531873e-06, 4.79095434e-07, 4.58031027e-05,
        7.48992970e-05],
       [4.47749766e-03, 9.65250671e-01, 4.88354824e-04, 2.97321938e-02,
        5.11882972e-05],
       [4.13611888e-05, 6.55233115e-03, 9.90925848e-01, 2.46071909e-03,
        1.97996615e-05],
       [9.99969780e-01, 1.18879626e-08, 1.35398803e-09, 4.27575287e-06,
        2.58900491e-05],
       [1.93605083e-04, 6.73294708e-04, 9.98399198e-01, 6.22513471e-04,
        1.11444700e-04],
       [2.72503087e-07, 4.23087236e-08, 5.76928905e-08, 3.90499807e-03,
        9.96094644e-01],
       [9.43804975e-04, 2.04935055e-02, 7.78236019e-04, 6.49481714e-01,
        3.28302771e-01],
       [9.99999940e-01, 2.47323065e-11, 5.76830337e-12, 2.05556283e-09,
        1.42746135e-08],
       [1.74128392e-03, 1.21899918e-02, 9.74617302e-01, 7.35693704e-03,
        4.09445167e-03],
       [1.09431325e-02, 4.27930281e-05, 7.84201175e-03, 4.76941392e-02,
        9.33477819e-01],
       [4.85268756e-05, 5.75254322e-04, 4.39714408e-03, 1.73598841e-01,
        8.21380317e-01],
       [9.99999940e-01, 5.10490044e-12, 8.72017194e-13, 1.14098925e-10,
        8.78420392e-10],
       [9.99999940e-01, 1.71382602e-12, 8.23305508e-13, 2.67156575e-10,
        4.48125803e-09],
       [9.99999940e-01, 2.26067580e-12, 3.28040302e-13, 4.84924566e-11,
        3.78847342e-10],
       [1.19902857e-03, 5.70465345e-03, 1.91556336e-03, 4.59326953e-01,
        5.31853914e-01],
       [5.00341157e-05, 1.13906211e-07, 5.05954212e-09, 4.10313532e-03,
        9.95846629e-01],
       [9.99999940e-01, 1.41064183e-09, 8.23568644e-11, 3.31224825e-09,
        6.58533272e-09],
       [3.59373614e-02, 1.31474990e-05, 3.20165454e-05, 1.29173081e-02,
        9.51100171e-01],
       [9.99996603e-01, 8.44012593e-09, 3.41800477e-09, 4.81045618e-07,
        2.86960312e-06],
       [6.26854307e-05, 1.15796879e-01, 8.22293514e-05, 8.61472785e-01,
        2.25852877e-02],
       [1.84965253e-01, 2.45325547e-02, 9.86454543e-04, 5.77957511e-01,
        2.11558163e-01],
       [9.99999821e-01, 1.58950364e-09, 1.53266233e-10, 4.14519938e-08,
        1.43860689e-07],
       [1.42232902e-05, 9.29070354e-01, 5.57741448e-02, 1.49999987e-02,
        1.41336219e-04],
       [9.99999940e-01, 1.81026461e-12, 7.30400441e-14, 1.08205216e-11,
        3.42230515e-11],
       [6.97014704e-02, 3.49381771e-05, 2.07613558e-02, 3.81239094e-02,
        8.71378303e-01],
       [4.38392814e-03, 9.32926357e-01, 1.03940554e-02, 5.10942787e-02,
        1.20139495e-03],
       [9.99995768e-01, 2.79096025e-06, 9.54093249e-09, 1.28936870e-06,
        1.68372466e-07],
```

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

comp

8	1
9	4
10	1
11	4
12	0
13	1
14	2
15	0
16	2
17	4
18	3
19	0
20	2
21	4
22	4
23	0
24	0
25	0
26	3
27	4
28	0
29	4
30	0
31	3
32	3
33	0
34	1
35	0
36	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([[0.19134112, 0.80901969, 0.95135444, 0.350701  , 0.93702362]])
```

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

```
1/1 [==============================] - 0s 76ms/step
array([[9.9999952e-01, 7.5995299e-09, 1.8451682e-09, 9.8660905e-08,
        3.6800341e-07]], 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
```

+ Code + Text