Question

Build an ANN model for Drug classification. This project aims to analyze the relationship between various medical parameters and drug effectiveness. The dataset consists of patient information, including age, sex, blood pressure levels (BP), cholesterol levels, sodium-to-potassium ratio (Na_to_K), drug type, and corresponding labels. The goal is to develop a model that can accurately predict the class or category of a given drug based on its features. Dataset Link: https://www.kaggle.com/datasets/prathamtripathi/drug-classification

Task 1: Read the dataset and do data pre-processing Task 2: Build the ANN model with (input layer, min 3 hidden layers & output layer) Task 3: Test the model with random data

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:

```
df = pd.read_csv('drug200.csv')
df
```

Out[6]:

	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
...	...	...	...	...	...	...
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

200 rows × 6 columns

In [7]:

```
df.head()
```

Out[7]:

	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

In [14]:

```
df.isnull().all()
```

Out[14]:

```
Age            False
Sex            False
BP             False
Cholesterol    False
Na_to_K        False
Drug           False
dtype: bool
```

In [15]:

```
df.info
```

Out[15]:

```
<bound method DataFrame.info of      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
..   ...  ..      ...         ...      ...    ...
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

[200 rows x 6 columns]>
```

In [16]:
```
df.shape
```

Out[16]:
```
(200, 6)
```

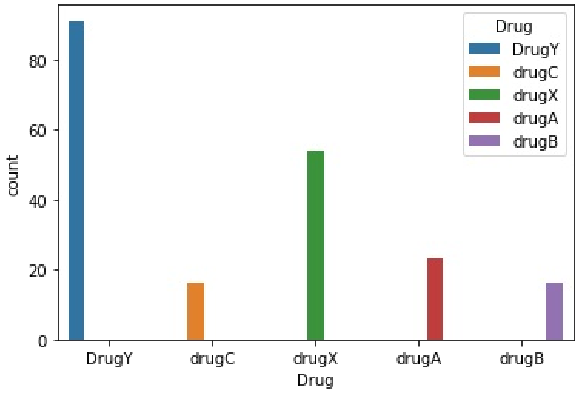
In [13]:
```
df.describe
```

Out[13]:
```
<bound method NDFrame.describe of      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
..   ...  ..      ...         ...      ...    ...
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

[200 rows x 6 columns]>
```

In [32]:
```
sns.countplot(x='Drug',data=df,hue='Drug')
```

Out[32]:
```
<AxesSubplot:xlabel='Drug', ylabel='count'>
```



In [33]:
```
X=df.iloc[:,0:5]
Y=df['Drug']
```

In [34]:
```
Y_class=len(np.unique(Y))
print(Y_class)
```

```
5
```

In [35]:
```
X
```

Out[35]:

	Age	Sex	BP	Cholesterol	Na_to_K
0	23	F	HIGH	HIGH	25.355

1	47	M	LOW	HIGH	13.093
2	47	M	LOW	HIGH	10.114
3	28	F	NORMAL	HIGH	7.798
4	61	F	LOW	HIGH	18.043
...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567
196	16	M	LOW	HIGH	12.006
197	52	M	NORMAL	HIGH	9.894
198	23	M	NORMAL	NORMAL	14.020
199	40	F	LOW	NORMAL	11.349

200 rows × 5 columns

In [21]:

```
Y
```

Out[21]:

```
0      DrugY
1      drugC
2      drugC
3      drugX
4      DrugY
       ...
195    drugC
196    drugC
197    drugX
198    drugX
199    drugX
Name: Drug, Length: 200, dtype: object
```

In [36]:

```
from sklearn.preprocessing import LabelEncoder

X=pd.get_dummies(X,columns=['Sex','BP','Cholesterol'],drop_first = True)
Le=LabelEncoder()
Y=Le.fit_transform(Y)
```

In [37]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.4,random_state=10)
```

In [38]:

```
from sklearn.preprocessing import StandardScaler

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

In [39]:

```
from tensorflow import keras
Y_train=keras.utils.to_categorical(Y_train)
Y_test=keras.utils.to_categorical(Y_test)
```

In [40]:

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

mod=Sequential()
mod.add(Dense(48,input_dim=6, activation='relu'))
mod.add(Dense(36,activation='relu'))
mod.add(Dense(24,activation='relu'))
mod.add(Dense(12,activation='relu'))
mod.add(Dense(Y_class,activation='softmax'))
```

In [42]:

```
mod.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

In [43]:

```
mod.fit(X_train,Y_train,epochs=48,batch_size=6)
```

```
Epoch 1/48
20/20 [==============================] - 4s 4ms/step - loss: 1.5573 - accuracy: 0.3250
Epoch 2/48
20/20 [==============================] - 0s 4ms/step - loss: 1.4643 - accuracy: 0.5667
Epoch 3/48
```

```
20/20 [==============================] - 0s 4ms/step - loss: 1.3311 - accuracy: 0.6833
Epoch 4/48
20/20 [==============================] - 0s 4ms/step - loss: 1.1402 - accuracy: 0.7500
Epoch 5/48
20/20 [==============================] - 0s 4ms/step - loss: 0.9213 - accuracy: 0.7333
Epoch 6/48
20/20 [==============================] - 0s 4ms/step - loss: 0.7377 - accuracy: 0.7417
Epoch 7/48
20/20 [==============================] - 0s 4ms/step - loss: 0.6065 - accuracy: 0.7500
Epoch 8/48
20/20 [==============================] - 0s 4ms/step - loss: 0.5262 - accuracy: 0.7917
Epoch 9/48
20/20 [==============================] - 0s 3ms/step - loss: 0.4584 - accuracy: 0.8167
Epoch 10/48
20/20 [==============================] - 0s 4ms/step - loss: 0.3857 - accuracy: 0.8667
Epoch 11/48
20/20 [==============================] - 0s 6ms/step - loss: 0.3331 - accuracy: 0.8833
Epoch 12/48
20/20 [==============================] - 0s 8ms/step - loss: 0.2916 - accuracy: 0.9083
Epoch 13/48
20/20 [==============================] - 0s 5ms/step - loss: 0.2527 - accuracy: 0.9167
Epoch 14/48
20/20 [==============================] - 0s 4ms/step - loss: 0.2213 - accuracy: 0.9167
Epoch 15/48
20/20 [==============================] - 0s 3ms/step - loss: 0.1974 - accuracy: 0.9250
Epoch 16/48
20/20 [==============================] - 0s 4ms/step - loss: 0.1604 - accuracy: 0.9750
Epoch 17/48
20/20 [==============================] - 0s 4ms/step - loss: 0.1327 - accuracy: 0.9833
Epoch 18/48
20/20 [==============================] - 0s 8ms/step - loss: 0.1041 - accuracy: 0.9833
Epoch 19/48
20/20 [==============================] - 0s 7ms/step - loss: 0.0826 - accuracy: 1.0000
Epoch 20/48
20/20 [==============================] - 0s 6ms/step - loss: 0.0693 - accuracy: 1.0000
Epoch 21/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0545 - accuracy: 1.0000
Epoch 22/48
20/20 [==============================] - 0s 5ms/step - loss: 0.0428 - accuracy: 1.0000
Epoch 23/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0351 - accuracy: 1.0000
Epoch 24/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0299 - accuracy: 1.0000
Epoch 25/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0330 - accuracy: 1.0000
Epoch 26/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0247 - accuracy: 1.0000
Epoch 27/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0195 - accuracy: 1.0000
Epoch 28/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0170 - accuracy: 1.0000
Epoch 29/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0148 - accuracy: 1.0000
Epoch 30/48
20/20 [==============================] - 0s 3ms/step - loss: 0.0134 - accuracy: 1.0000
Epoch 31/48
20/20 [==============================] - 0s 5ms/step - loss: 0.0133 - accuracy: 1.0000
Epoch 32/48
20/20 [==============================] - 0s 5ms/step - loss: 0.0116 - accuracy: 1.0000
Epoch 33/48
20/20 [==============================] - 0s 5ms/step - loss: 0.0102 - accuracy: 1.0000
Epoch 34/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0089 - accuracy: 1.0000
Epoch 35/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0082 - accuracy: 1.0000
Epoch 36/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0079 - accuracy: 1.0000
Epoch 37/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0067 - accuracy: 1.0000
Epoch 38/48
20/20 [==============================] - 0s 3ms/step - loss: 0.0059 - accuracy: 1.0000
Epoch 39/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0057 - accuracy: 1.0000
Epoch 40/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0052 - accuracy: 1.0000
Epoch 41/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 42/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0047 - accuracy: 1.0000
Epoch 43/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0044 - accuracy: 1.0000
Epoch 44/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0041 - accuracy: 1.0000
```

```
Epoch 45/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0038 - accuracy: 1.0000
Epoch 46/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0038 - accuracy: 1.0000
Epoch 47/48
20/20 [==============================] - 0s 6ms/step - loss: 0.0035 - accuracy: 1.0000
Epoch 48/48
20/20 [==============================] - 0s 4ms/step - loss: 0.0032 - accuracy: 1.0000
```

Out[43]:
```
<keras.callbacks.History at 0x1cda2b3a0d0>
```

In [46]:
```
test_loss,test_acc=mod.evaluate(X_test,Y_test)
print('Test Accuracy:',test_acc*100)
```

```
3/3 [==============================] - 0s 4ms/step - loss: 0.4782 - accuracy: 0.8500
Test Accuracy: 85.00000238418579
```

In [51]:
```
pred = mod.predict(X_test[:1])
```

```
1/1 [==============================] - 0s 20ms/step
```

In [52]:
```
pred
```

Out[52]:
```
array([[9.9998724e-01, 5.9284008e-07, 1.0111640e-06, 1.1128964e-05,
        6.6235086e-12]], dtype=float32)
```

In [ ]:
```

```