ASSESSMENT 2

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. ¶

Import the libraries

In [1]:

```
import pandas as pd
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
import numpy as np
import seaborn as sb
import matplotlib.pyplot as mp
```

Load the dataset

In [2]:

```
dataset = pd.read_csv('drug.csv')
```

Glimpse of the dataset

In [3]:

```
dataset.head()
```

Out[3]:

	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

Data Preprocessing

Check for total no. of rows and column

In [4]:

```
dataset.shape
```

Out[4]:

```
(200, 6)
```

Total number of rows are 200 Total number of columns are 6

Check for missing values

In [5]:

```
dataset.isnull().sum()
```

Out[5]:

```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

Check for dataset information

In [6]:

```
dataset.info
```

Out[6]:

```
<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]>
```

Check for statistical info

```
In [7]:
```

```
dataset.describe
```

```
Out[7]:
```

```
<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]>
```

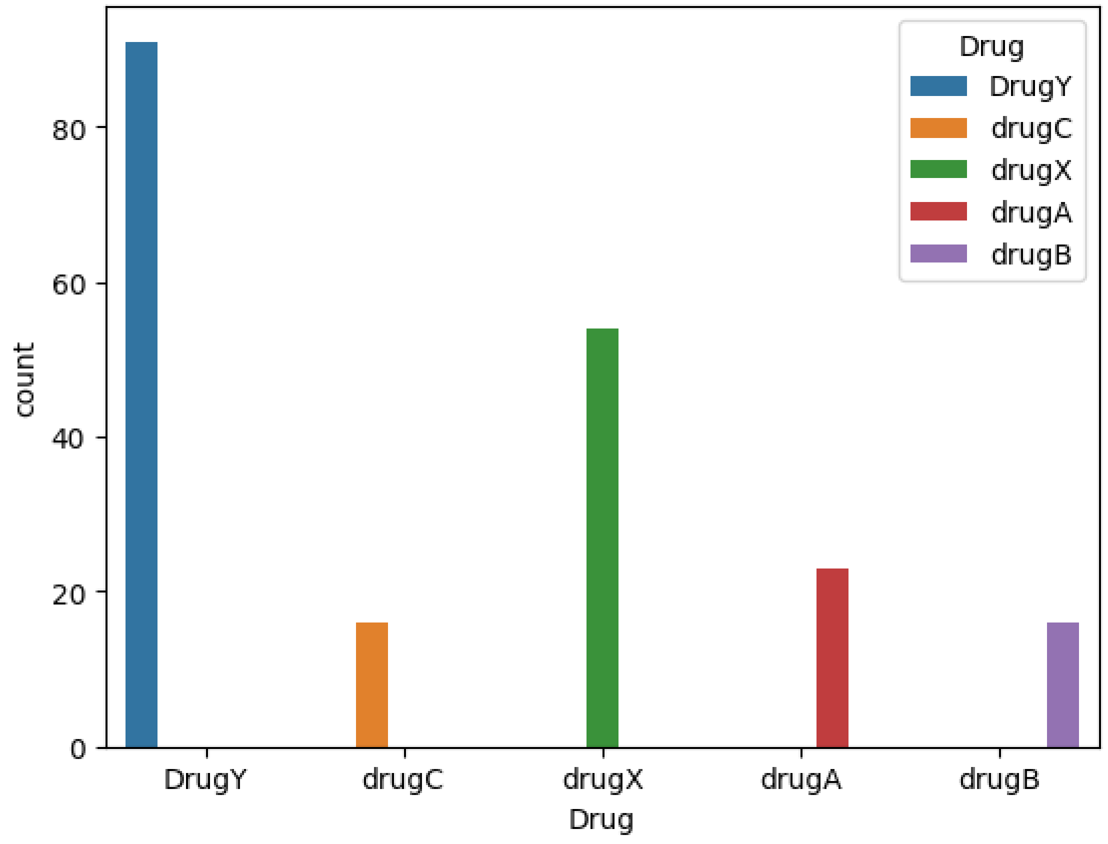
Let's Visualize in the Graph forms

In [8]:

```
sb.countplot(x='Drug',data=dataset,hue='Drug')
```

Out[8]:

```
<AxesSubplot: xlabel='Drug', ylabel='count'>
```



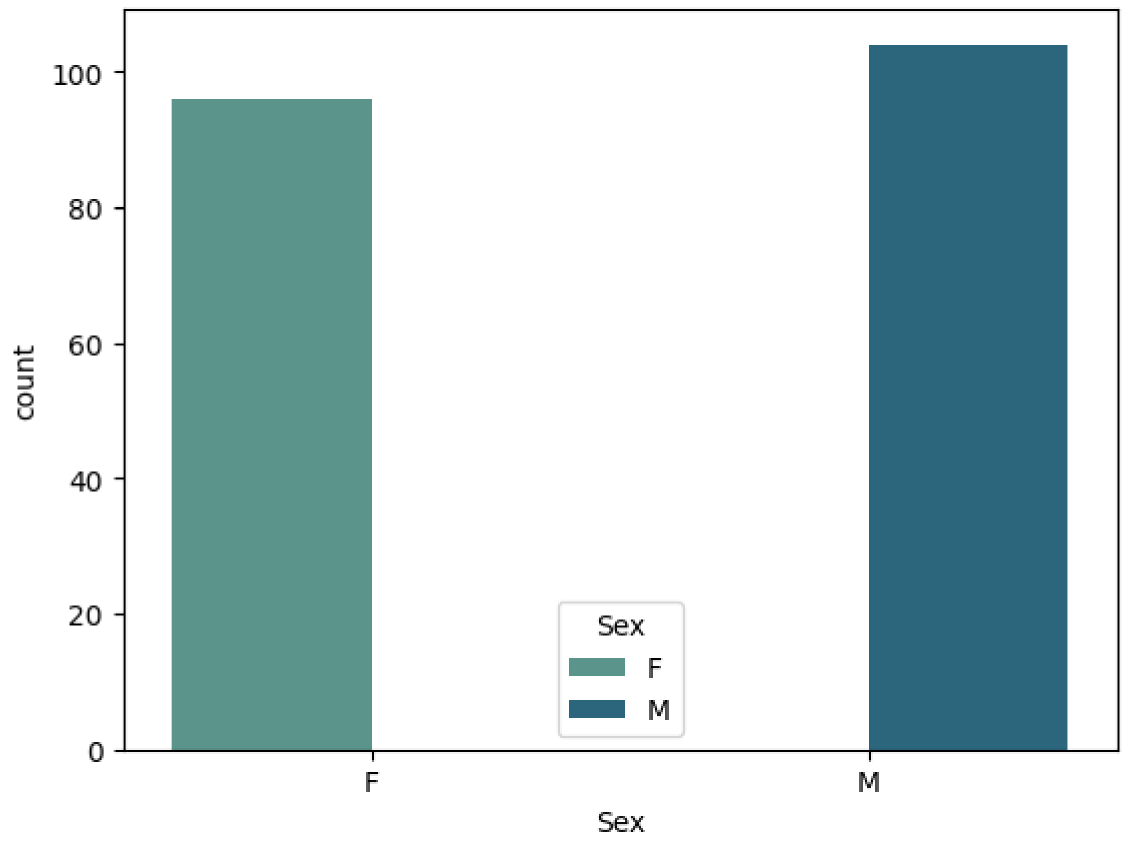
Consumption of Drug Y is more and Drug B is less compare to all

In [9]:

```
sb.countplot(x='Sex',data=dataset,palette='crest',hue='Sex')
```

Out[9]:

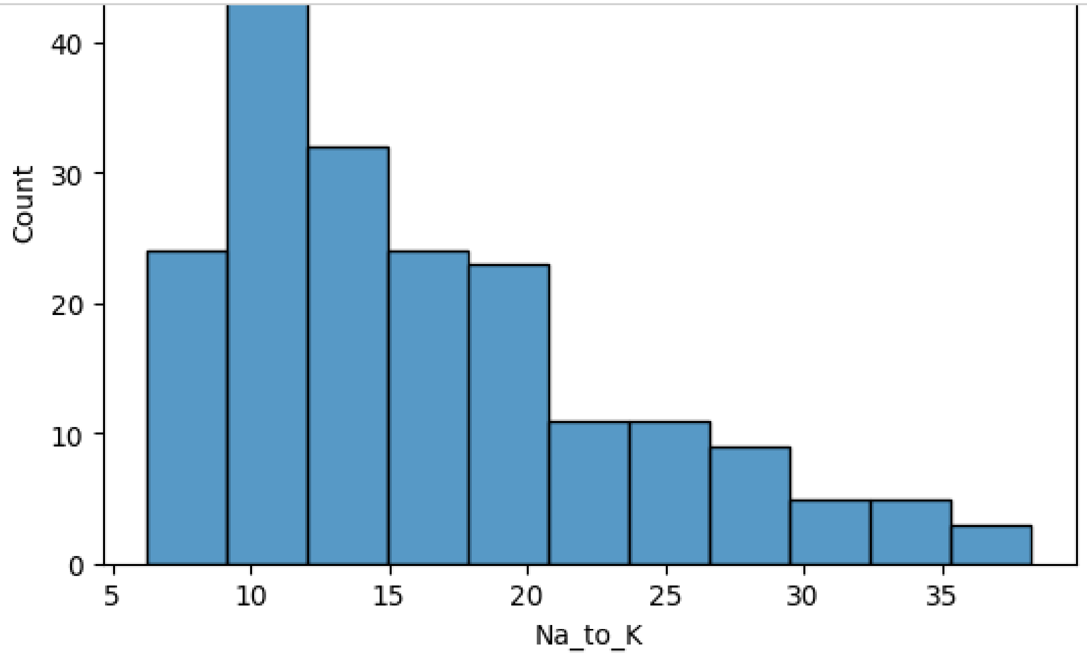
```
<AxesSubplot: xlabel='Sex', ylabel='count'>
```



Male Drug's consumption is more compare to Female

In [10]:

```
sb.histplot(x='Na_to_K',data=dataset)
```



Segregate the data into train & test

In [11]:

```
X = dataset.drop(columns = ['Drug'],axis=1)
Y = dataset['Drug']
```

Drug column is the output(target) column in this dataset.

In [12]:

```
Y_class = len(np.unique(Y))
print(Y_class)
```

```
5
```

The above code snippet will fetch the count of unique class value of the target column.

Convert the Categorical data into interger data

In [13]:

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

LE = LabelEncoder()
Y = LE.fit_transform(Y)
```

get_dummies is a pandas function which will convert the one-hot encoding categorical value into numeric value

Labelencoder will assign a numeric value to a category value.

In [14]:

```
X_train, X_test, Y_train, Y_test = train_test_split (X,Y,test_size=0.3,random_state=2)
```

In the above code snippet, I have divide the dataset into train and test

Feature Scaling

In [15]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

StandardScaler is used for scaling the numeric values

In [16]:

```
Y_train = keras.utils.to_categorical(Y_train)
Y_test = keras.utils.to_categorical(Y_test)
```

the to_categorical() function from the Keras library to convert the target variable arrays Y_train and Y_test into one-hot encoded arrays.

create a ANN model

In [17]:

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

In [19]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 48)                336       
                                                                 
 dense_1 (Dense)             (None, 36)                1764      
                                                                 
 dense_2 (Dense)             (None, 24)                888       
                                                                 
 dense_3 (Dense)             (None, 12)                300       
                                                                 
 dense_4 (Dense)             (None, 5)                 65        
                                                                 
=================================================================
Total params: 3,353
Trainable params: 3,353
Non-trainable params: 0
_________________________________________________________________
```

compile the model

In [18]:

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

train the model

In [19]:

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

```
Epoch 1/48
24/24 [==============================] - 1s 2ms/step - loss: 1.5547 - accu
racy: 0.2929
Epoch 2/48
24/24 [==============================] - 0s 2ms/step - loss: 1.3912 - accu
racy: 0.4214
Epoch 3/48
24/24 [==============================] - 0s 1ms/step - loss: 1.2214 - accu
racy: 0.6429
Epoch 4/48
24/24 [==============================] - 0s 1ms/step - loss: 1.0234 - accu
racy: 0.6643
Epoch 5/48
24/24 [==============================] - 0s 1ms/step - loss: 0.8249 - accu
racy: 0.6714
Epoch 6/48
24/24 [==============================] - 0s 1ms/step - loss: 0.6538 - accu
racy: 0.7500
Epoch 7/48
24/24 [==============================] - 0s 1ms/step - loss: 0.5144 - accu
racy: 0.8071
Epoch 8/48
24/24 [==============================] - 0s 1ms/step - loss: 0.3943 - accu
racy: 0.9286
Epoch 9/48
24/24 [==============================] - 0s 1ms/step - loss: 0.3025 - accu
racy: 0.9786
Epoch 10/48
24/24 [==============================] - 0s 1ms/step - loss: 0.2323 - accu
racy: 0.9786
Epoch 11/48
24/24 [==============================] - 0s 1ms/step - loss: 0.1760 - accu
racy: 0.9857
Epoch 12/48
24/24 [==============================] - 0s 1ms/step - loss: 0.1455 - accu
racy: 0.9786
Epoch 13/48
24/24 [==============================] - 0s 1ms/step - loss: 0.1178 - accu
racy: 0.9929
Epoch 14/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0933 - accu
racy: 0.9857
Epoch 15/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0745 - accu
racy: 0.9929
Epoch 16/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0651 - accu
racy: 0.9929
Epoch 17/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0555 - accu
racy: 0.9929
Epoch 18/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0474 - accu
racy: 1.0000
Epoch 19/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0413 - accu
racy: 0.9929
Epoch 20/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0356 - accu
racy: 1.0000
Epoch 21/48
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.0355 - accu
racy: 1.0000
Epoch 22/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0301 - accu
racy: 1.0000
Epoch 23/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0273 - accu
racy: 1.0000
Epoch 24/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0244 - accu
racy: 1.0000
Epoch 25/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0222 - accu
racy: 1.0000
Epoch 26/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0202 - accu
racy: 1.0000
Epoch 27/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0189 - accu
racy: 1.0000
Epoch 28/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0170 - accu
racy: 1.0000
Epoch 29/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0164 - accu
racy: 1.0000
Epoch 30/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0148 - accu
racy: 1.0000
Epoch 31/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0138 - accu
racy: 1.0000
Epoch 32/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0137 - accu
racy: 1.0000
Epoch 33/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0134 - accu
racy: 1.0000
Epoch 34/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0118 - accu
racy: 1.0000
Epoch 35/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0119 - accu
racy: 1.0000
Epoch 36/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0105 - accu
racy: 1.0000
Epoch 37/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0088 - accu
racy: 1.0000
Epoch 38/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0092 - accu
racy: 1.0000
Epoch 39/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0080 - accu
racy: 1.0000
Epoch 40/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0083 - accu
racy: 1.0000
Epoch 41/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0083 - accu
```

```
racy: 1.0000
Epoch 42/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0084 - accu
racy: 1.0000
Epoch 43/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0153 - accu
racy: 0.9929
Epoch 44/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0059 - accu
racy: 1.0000
Epoch 45/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0055 - accu
racy: 1.0000
Epoch 46/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0055 - accu
racy: 1.0000
Epoch 47/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0050 - accu
racy: 1.0000
Epoch 48/48
24/24 [==============================] - 0s 1ms/step - loss: 0.0047 - accu
racy: 1.0000
```

Out[19]:

```
<keras.callbacks.History at 0x12f6ca780d0>
```

evaluate the model on the test set

In [20]:

```
test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc * 100)
```

```
2/2 [==============================] - 0s 5ms/step - loss: 0.3741 - accura
cy: 0.8833
Test accuracy: 88.33333253860474
```