Decision Tree Model for Drugs A, B, C, X, Y dataset

Objective

- The objective is to build a model to find out which drug might be appropriate for a future patient with the same illness. It is to predict the class of a unknown patient, or to prescribe a drug to a new patient.

Decision Tree

- A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data.
- It uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

Decision Tree terminologies

- Root Node: It is the topmost node in the tree, which represents the complete dataset. It is the starting point of the decision-making process.
- Decision/Internal Node: A node that symbolizes a choice regarding an input feature. Branching off of internal nodes connects them to leaf nodes or other internal nodes.
- Leaf/Terminal Node: A node without any child nodes that indicates a class label or a numerical value.
- Branch/Sub-Tree: A subsection of the decision tree starts at an internal node and ends at the leaf nodes.
- Parent Node: The node that divides into one or more child nodes.
- Child Node: The nodes that emerge when a parent node is split.
- Splitting: The process of splitting a node into two or more sub-nodes using a split criterion and a selected feature.
- Impurity: A measurement of the target variable’s homogeneity in a subset of data. It refers to the degree of randomness or uncertainty in a set of examples. The Gini index and entropy are two commonly used impurity measurements in decision trees for classifications task.
- Information Gain: Information gain is a measure of the reduction in impurity achieved by splitting a dataset on a particular feature in a decision tree. The splitting criterion is determined by the feature that offers the greatest information gain. It is used to determine the most informative feature to split on at each node of the tree, with the goal of creating pure subsets
- Pruning: The process of removing branches from the tree that do not provide any additional information or lead to overfitting.

Decision Tree assumptions

- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values, records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.

Dataset source & brief

- Dataset has been taken from Kaggle.
- The features of this dataset are Age, Sex, Blood Pressure (Low, Normal, or High), Cholesterol: level of cholesterol in the blood (either Normal or High), Na_to_K: Sodium to Potassium ratio, Drug: the Drug the patient responded to or was treated with (Either Drug A, B, C, X, or Y), it is the traget variable

Outline

- Import Libraries & Dataset
- Basic information about the dataset
- Data Pre Processing & EDA
- Split data into Dependent and Independent variables & then into Train and test
- Build Decision Tree model - Gini & Entropy,Predict & Evaluate the model
- Feature importance
- Decision Tree visualization
- Post Prunning
- Conclusion

Import required libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Import dataset

In [2]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\Drug a b c d.csv")
df
```

Out[2]:

	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 [3]:

```
# Check shape
df.shape
```

Out[3]:

```
(200, 6)
```

In [4]:

```
# Check null values
df.isnull().sum()     # no null values
```

Out[4]:

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

In [5]:

```
# Check info
df.info()  # 4 object columns
```

```
<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
```

Encoding

In [6]:

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

Out[6]:

```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [7]:

```
# One hot encoding for BP variable
df = pd.get_dummies(df,columns=['BP'])
```

In [8]:

```
df.head(2)
```

Out[8]:

	Age	Sex	Cholesterol	Na_to_K	Drug	BP_HIGH	BP_LOW	BP_NORMAL
0	23	F	HIGH	25.355	drugY	1	0	0
1	47	M	HIGH	13.093	drugC	0	1	0

In [9]:

```
# Drop dummy variable
df = df.drop(['BP_HIGH'], axis=1)
```

In [10]:

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

Out[10]:

```
M    104
F     96
Name: Sex, dtype: int64
```

In [11]:

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

Out[11]:

```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [12]:

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

Out[12]:

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

In [13]:

```
# Label encoder for Sex, Cholesterol and Drug(as its dependent variable)
df['Sex'] = df['Sex'].astype('category')
df['Sex'] = df['Sex'].cat.codes

df['Cholesterol'] = df['Cholesterol'].astype('category')
df['Cholesterol'] = df['Cholesterol'].cat.codes

df['Drug'] = df['Drug'].astype('category')
df['Drug'] = df['Drug'].cat.codes
```

In [14]:

```
df.head()
```

Out[14]:

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

In [15]:

```
df.describe().T.style.background_gradient(cmap='Reds')
```

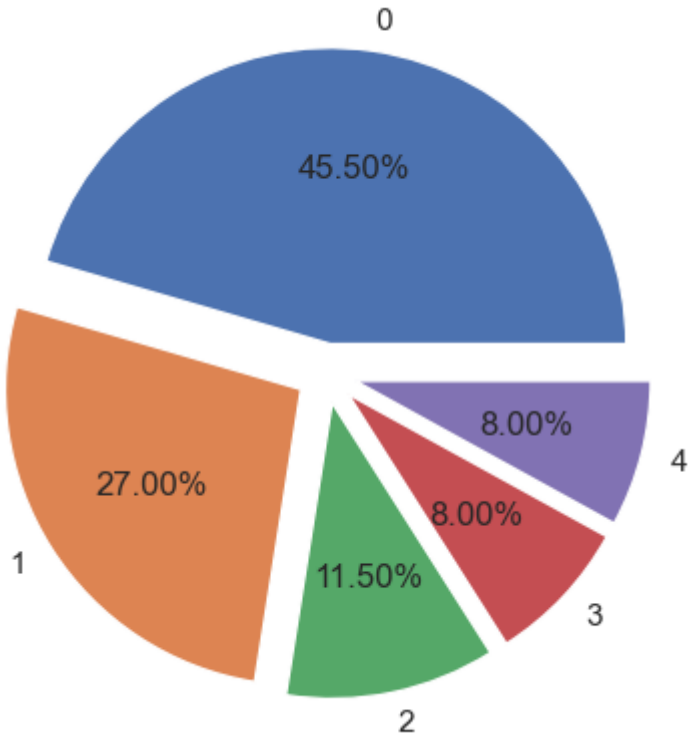
Out[15]:

	count	mean	std	min	25%	50%	75%
Age	200.000000	44.315000	16.544315	15.000000	31.000000	45.000000	58.000000
Sex	200.000000	0.520000	0.500854	0.000000	0.000000	1.000000	1.000000
Cholesterol	200.000000	0.485000	0.501029	0.000000	0.000000	0.000000	1.000000
Na_to_K	200.000000	16.084485	7.223956	6.269000	10.445500	13.936500	19.380000
Drug	200.000000	2.870000	1.372047	0.000000	2.000000	3.000000	4.000000
BP_LOW	200.000000	0.320000	0.467647	0.000000	0.000000	0.000000	1.000000
BP_NORMAL	200.000000	0.295000	0.457187	0.000000	0.000000	0.000000	1.000000

Exploratory Data Analysis

In [16]:

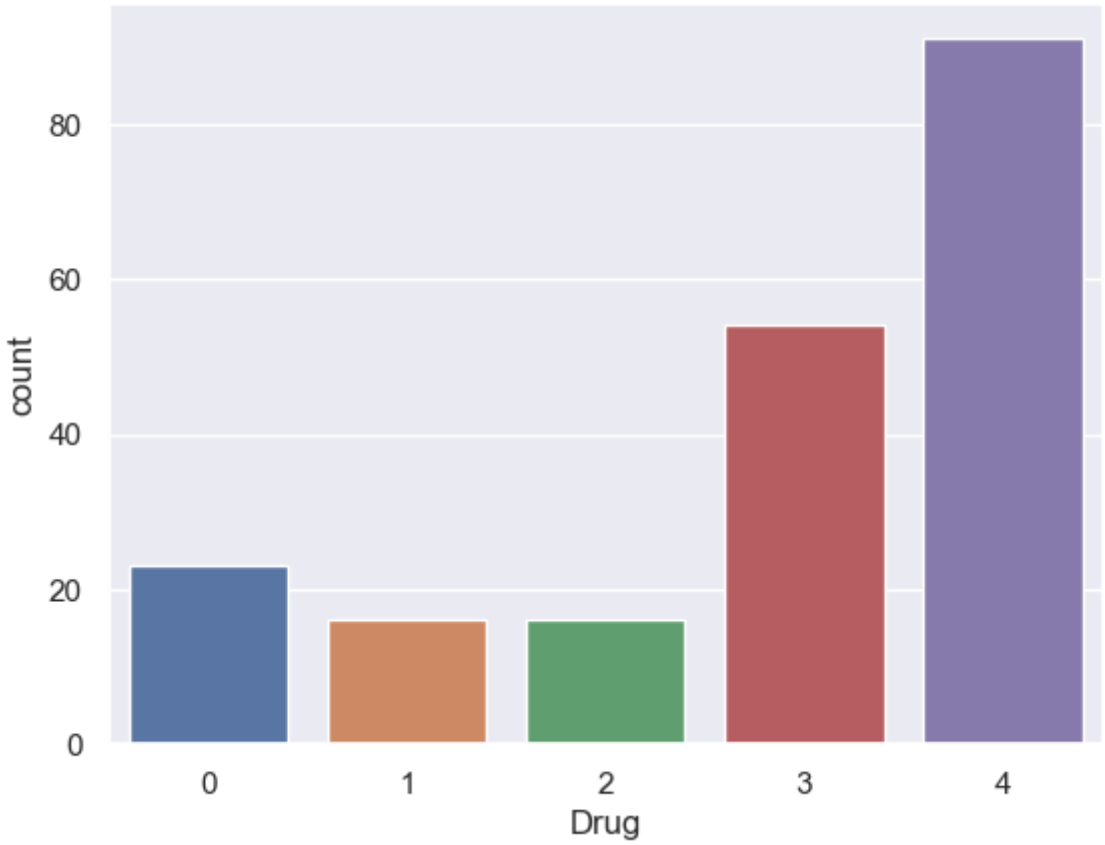
```
plt.pie(df.Drug.value_counts(),labels=[0,1,2,3,4,],autopct='%.2f%%',explode=(0.1,0.1,0.1
plt.show()
```



In [17]:

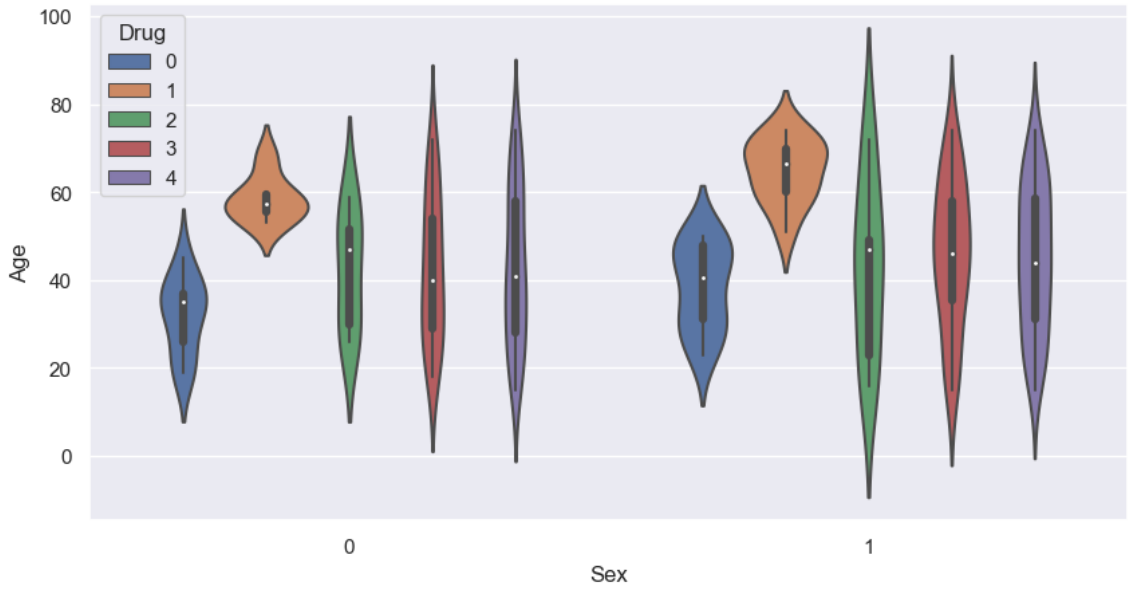
```
# imbalance check
sns.countplot(data=df, x='Drug')
A,B,C,D,E= df['Drug'].value_counts()
print("Number of A :", A)
print("Number of B :", B)
print("Number of C:", C)
print("Number of D:", D)
print("Number of E:", E)
plt.show()
```

```
Number of A : 91
Number of B : 54
Number of C: 23
Number of D: 16
Number of E: 16
```



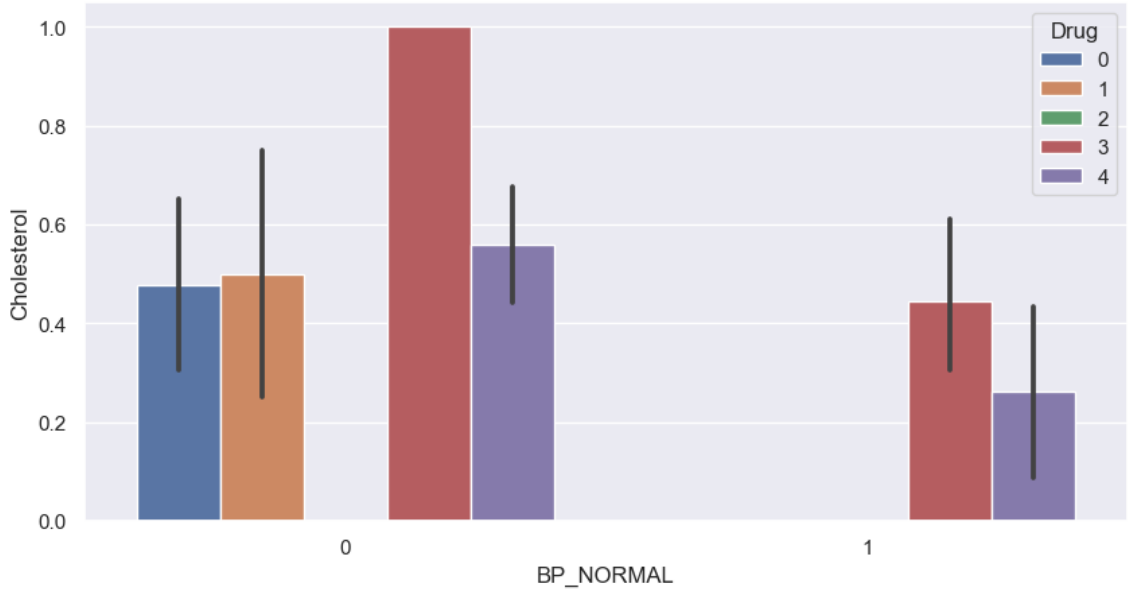
In [18]:

```
plt.figure(figsize=(10,5),dpi=100)
sns.violinplot(y='Age',x='Sex',hue='Drug',data=df)
plt.show()
```



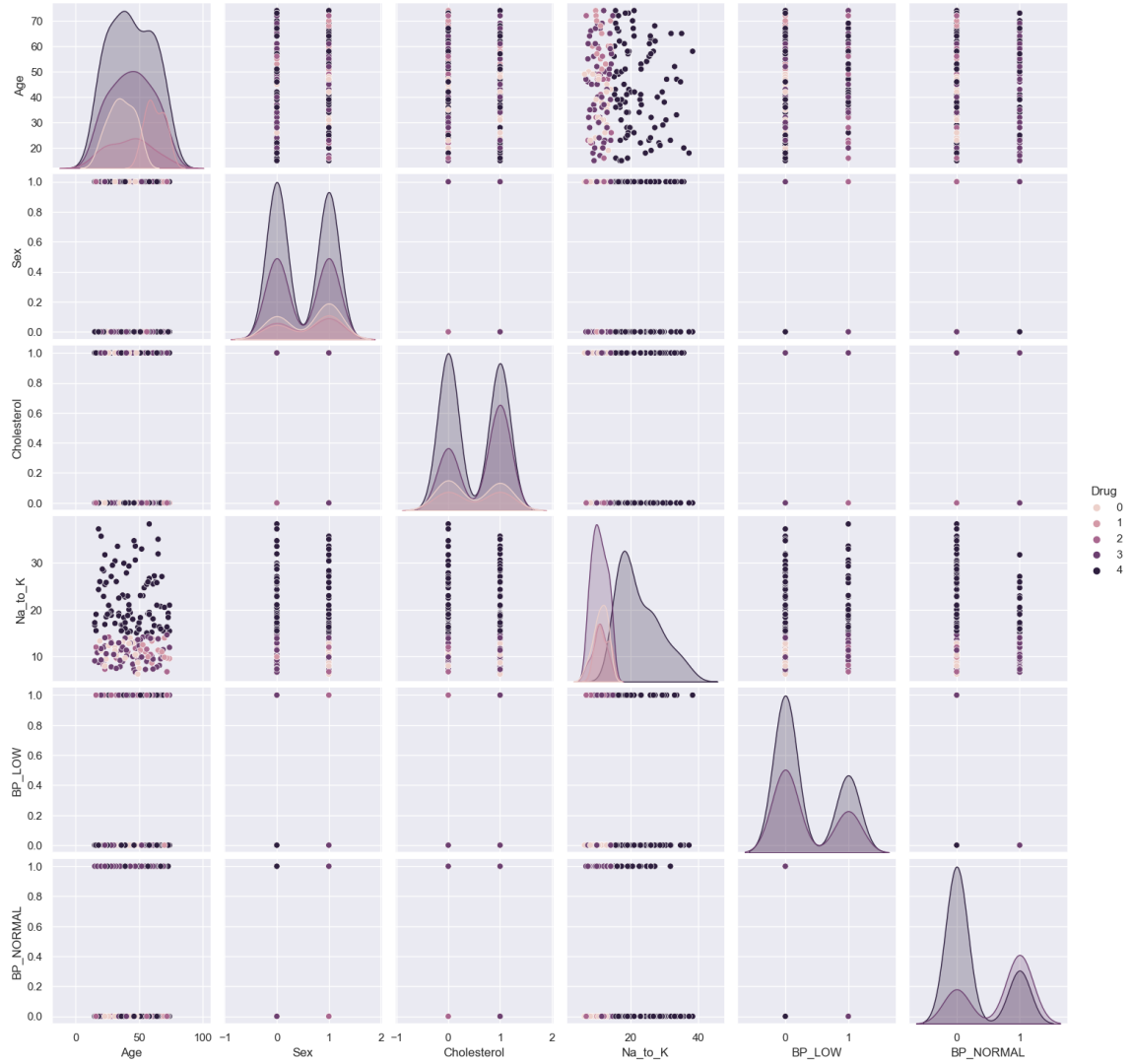
In [19]:

```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='Cholesterol',x='BP_NORMAL',hue='Drug',data=df)
plt.show()
```



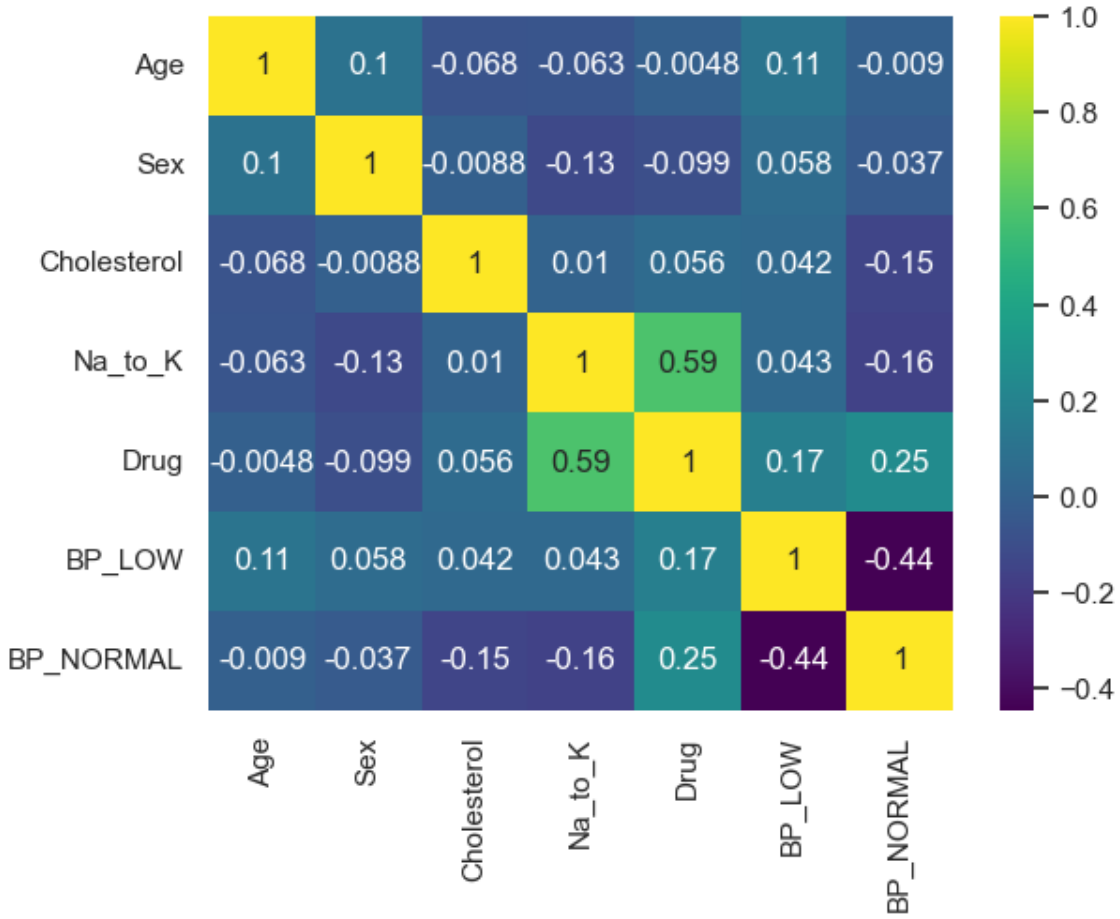
In [20]:

```
sns.pairplot(df, hue='Drug')
plt.show()
```



In [21]:

```
# Correlation
sns.heatmap(df.corr(), annot=True, cmap='viridis')
plt.show()
```



Data Splitting

In [22]:

```
# Split data into independent and Dependent variables
x = df.drop(['Drug'],axis=1)
y = df[['Drug']]
```

In [23]:

```
x.head(2)
```

Out[23]:

	Age	Sex	Cholesterol	Na_to_K	BP_LOW	BP_NORMAL
0	23	0	0	25.355	0	0
1	47	1	0	13.093	1	0

In [24]:

```
y.head(2)
```

Out[24]:

	Drug
0	4
1	2

In [25]:

```
# Splitting data into train and test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=101)
```

In [26]:

```
# shape after splitting
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

Out[26]:

```
((160, 6), (160, 1), (40, 6), (40, 1))
```

Modelling by

- Gini (default) criterion
- Entropy criterion

Gini Impurity

- It measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled.
- The minimum value of the Gini Index is 0. This happens when the node is pure, this means that all the contained elements in the node are of one unique class. Therefore, this node will not be split again. Thus, the optimum split is chosen by the features with less Gini Index. Moreover, it gets the maximum value when the probability of the two classes are the same.
- Gini Index formula = 1–∑jp2j, where pj is the probability of class j.

In [27]:

```
from sklearn.tree import DecisionTreeClassifier,plot_tree
dt_gn=DecisionTreeClassifier(criterion ='gini')
dt_gn.fit(x_train,y_train)
```

Out[27]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Entropy

- It is a measure of information that indicates the disorder of the features with the target. Similar to the Gini Index, the optimum split is chosen by the feature with less entropy. It gets its maximum value when the probability of the two classes is the same and a node is pure when the entropy has its minimum value, which is 0.
- Entropy formula: –∑jpj·log2·pj where pj is the probability of class j.

In [28]:

```
dt_et=DecisionTreeClassifier(criterion ='entropy')
dt_et.fit(x_train,y_train)
```

Out[28]:

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [29]:

```
# Predict the model
y_pred_train_dt_gn=dt_gn.predict(x_train)
y_pred_test_dt_gn=dt_gn.predict(x_test)

y_pred_train_dt_et=dt_et.predict(x_train)
y_pred_test_dt_et=dt_et.predict(x_test)
```

In [30]:

```
# Evaluation
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [31]:

```
# Gini
print(classification_report(y_train,y_pred_train_dt_gn))
print(classification_report(y_test,y_pred_test_dt_gn))
print('******************'*5)
print(confusion_matrix(y_train,y_pred_train_dt_gn))
print()
print(confusion_matrix(y_test,y_pred_test_dt_gn))
print('******************'*5)
print(accuracy_score(y_train,y_pred_train_dt_gn))
print()
print(accuracy_score(y_test,y_pred_test_dt_gn))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00        44
           4       1.00      1.00      1.00        73

    accuracy                           1.00       160
   macro avg       1.00      1.00      1.00       160
weighted avg       1.00      1.00      1.00       160

              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         5
           3       1.00      0.90      0.95        10
           4       0.95      1.00      0.97        18

    accuracy                           0.97        40
   macro avg       0.99      0.98      0.98        40
weighted avg       0.98      0.97      0.97        40

**********************************************************************
***************
[[20  0  0  0  0]
 [ 0 12  0  0  0]
 [ 0  0 11  0  0]
 [ 0  0  0 44  0]
 [ 0  0  0  0 73]]

[[ 3  0  0  0  0]
 [ 0  4  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0  9  1]
 [ 0  0  0  0 18]]
**********************************************************************
**********
1.0

0.975
```

In [32]:

```
# Entropy
print(classification_report(y_train,y_pred_train_dt_et))
print()
print(classification_report(y_test,y_pred_test_dt_et))
print('******************'*5)
print(confusion_matrix(y_train,y_pred_train_dt_et))
print()
print(confusion_matrix(y_test,y_pred_test_dt_et))
print('******************'*5)
print(accuracy_score(y_train,y_pred_train_dt_et))
print()
print(accuracy_score(y_test,y_pred_test_dt_et))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00        44
           4       1.00      1.00      1.00        73

    accuracy                           1.00       160
   macro avg       1.00      1.00      1.00       160
weighted avg       1.00      1.00      1.00       160


              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         4
           2       1.00      1.00      1.00         5
           3       1.00      0.90      0.95        10
           4       0.95      1.00      0.97        18

    accuracy                           0.97        40
   macro avg       0.99      0.98      0.98        40
weighted avg       0.98      0.97      0.97        40

**********************************************************************
***************
[[20  0  0  0  0]
 [ 0 12  0  0  0]
 [ 0  0 11  0  0]
 [ 0  0  0 44  0]
 [ 0  0  0  0 73]]

[[ 3  0  0  0  0]
 [ 0  4  0  0  0]
 [ 0  0  5  0  0]
 [ 0  0  0  9  1]
 [ 0  0  0  0 18]]
**********************************************************************
***************
1.0

0.975
```

Feature Importance

- Feature importance refers to technique that assigns a score to features based on how significant they are at predicting a target variable.

In [33]:

```
# Check Feature importance- Gini
dt_gn.feature_importances_
pd.DataFrame(index=x.columns,data=dt_gn.feature_importances_, columns=['Gini Feature Imp
```

Out[33]:

	Gini Feature Importance
Age	0.135823
Sex	0.000000
Cholesterol	0.107904
Na_to_K	0.482934
BP_LOW	0.128493
BP_NORMAL	0.144845

In [34]:

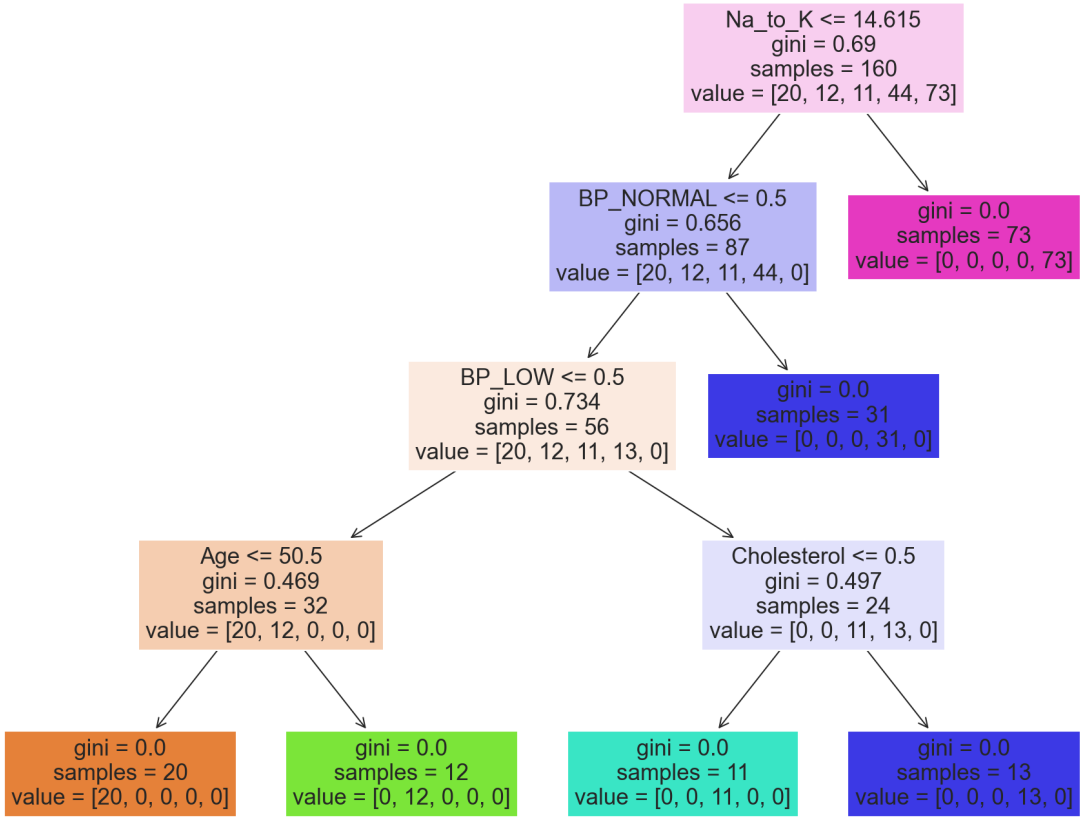
```
# Check Feature importance- Entropy
dt_et.feature_importances_
pd.DataFrame(index=x.columns,data=dt_et.feature_importances_, columns=['Entropy Feature
```

Out[34]:

	Entropy Feature Importance
Age	0.097914
Sex	0.000000
Cholesterol	0.076556
Na_to_K	0.510108
BP_LOW	0.176879
BP_NORMAL	0.138543

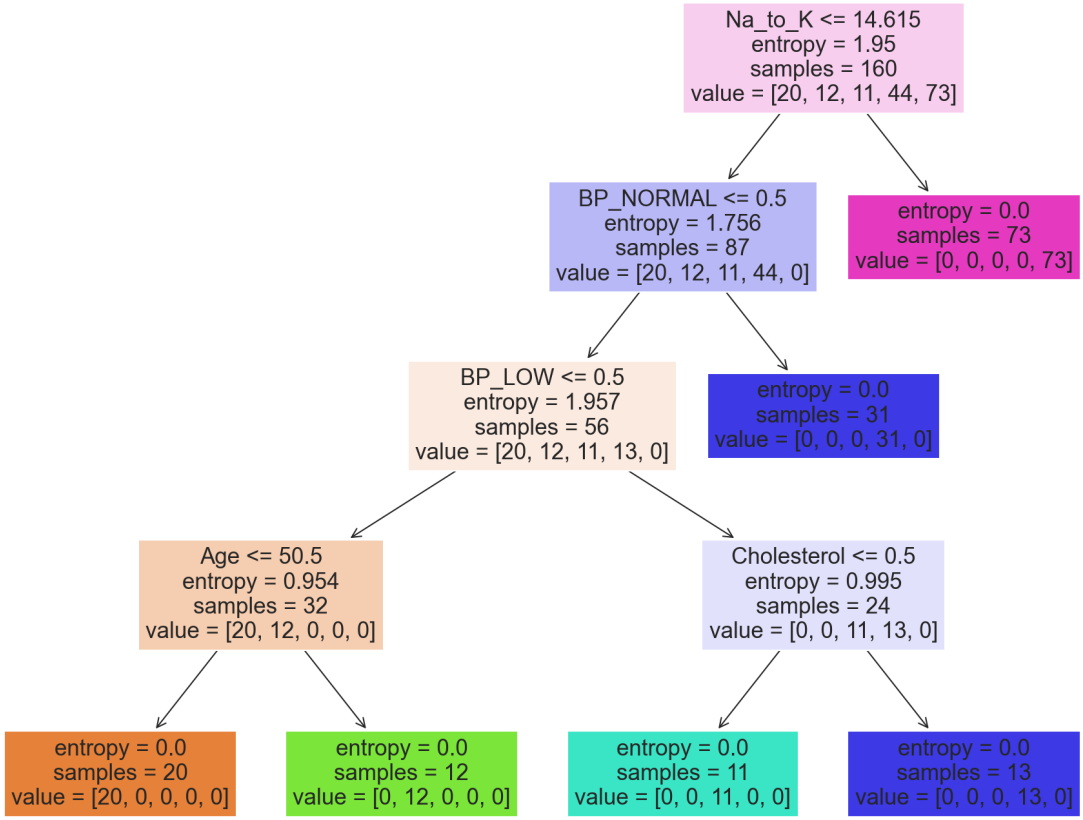
In [35]:

```
# Gini Visualization
from sklearn.tree import plot_tree
plt.figure(figsize=(15,12),dpi=150)
plot_tree(dt_gn,filled=True,feature_names=x.columns)
plt.show()
```



In [36]:

```
# Entropy Visualization
from sklearn.tree import plot_tree
plt.figure(figsize=(15,12),dpi=150)
plot_tree(dt_et,filled=True,feature_names=x.columns)
plt.show()
```



Overfitting problem & its solution

- Overfitting is a common problem that needs to be handled while training a decision tree model.
- It occurs when a model fits too closely to the training data and may become less accurate when encountering new data or predicting future outcomes.
- The solution to overfitting problem is Pruning. It is a technique that removes parts of the decision tree and prevents it from growing to its full depth. Pruning removes those parts of the decision tree that do not have the power to classify instances.
- Pruning can be of two types — Pre-Pruning and Post-Pruning. Pre-Pruning, also known as ‘Early Stopping’ or ‘Forward Pruning’, stops the growth of the decision tree — preventing it from reaching its full depth. Post-Pruning or ‘backward pruning’ is a technique that eliminates branches from a “completely grown” decision tree model to reduce its complexity and variance.

In [37]:

```
# Using Post prunning method to handle overfitting probelm
def report_model(model):
    model_preds=model.predict(x_test)
    print(classification_report(y_test,model_preds))
    print('\n')
    plt.figure(figsize=(15,12),dpi=150)
    plot_tree(model,filled=True,feature_names=x.columns)
plt.show()
```

In [38]:

```
# max depth at 3
prunned_dtree=DecisionTreeClassifier(max_depth=3)
prunned_dtree.fit(x_train,y_train)
```

Out[38]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

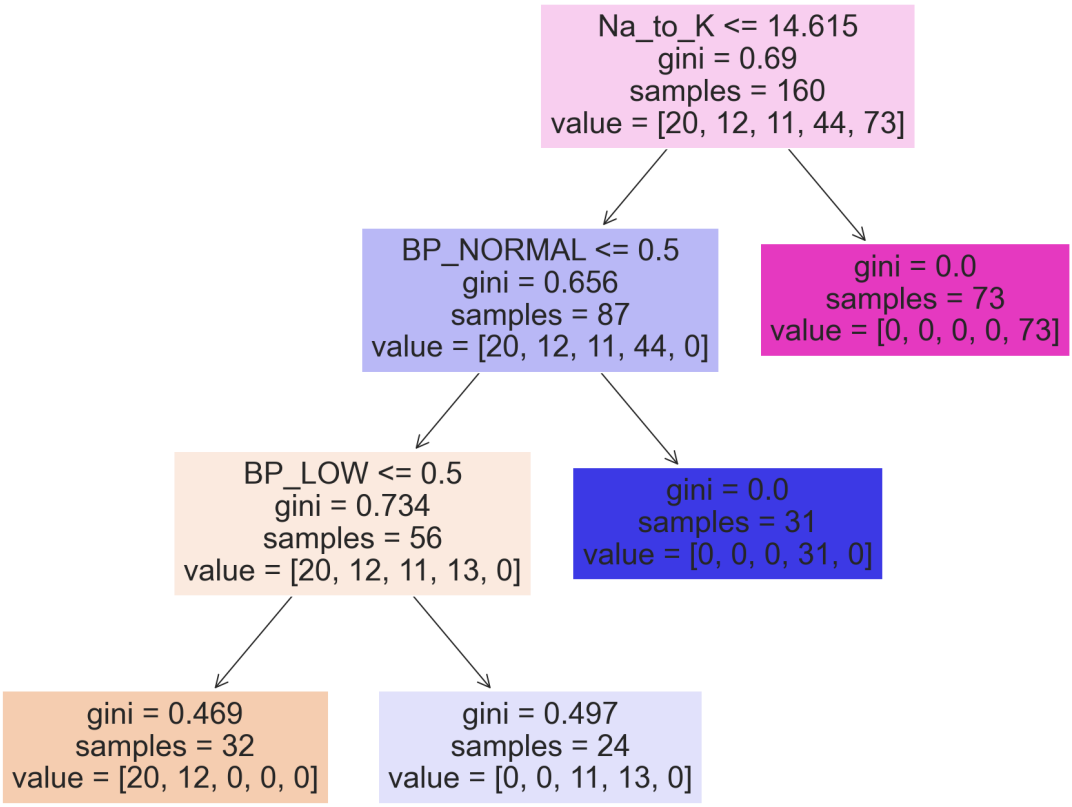
In [39]:

```
report_model(prunned_dtree)
```

```
              precision    recall  f1-score   support

           0       0.43      1.00      0.60         3
           1       0.00      0.00      0.00         4
           2       0.00      0.00      0.00         5
           3       0.64      0.90      0.75        10
           4       0.95      1.00      0.97        18

    accuracy                           0.75        40
   macro avg       0.40      0.58      0.46        40
weighted avg       0.62      0.75      0.67        40
```



In [40]:

```
y_pred_prunned_train=prunned_dtree.predict(x_train)
y_pred_prunned_test=prunned_dtree.predict(x_test)
```

In [41]:

```
print(accuracy_score(y_train,y_pred_prunned_train))
print()
print(accuracy_score(y_test,y_pred_prunned_test))
```

```
0.85625

0.75
```

Conclusion

- In this analysis, I constructed a decision tree model to predict the most suitable drug for future patients with similar conditions. I leveraged key features such as age, sex, blood pressure, cholesterol levels and the Sodium-Potassium ratio (Na_to_K).
- Using this plot, we can easily make decisions for what treatment might work for a future patient.
- Before Prunning we were getting train accuracy of 100% and test accuracy of 97.5%
- After giving max depth of 3, post prunning train data accuracy is of 85% and test data accuracy of 75%.

In [ ]:

```

```