Data Preprocessing

```
# Import Data
import pandas as pd
df = pd.read_csv("drug200_mod.csv")
data_vis=df
```

```
df.head()
```

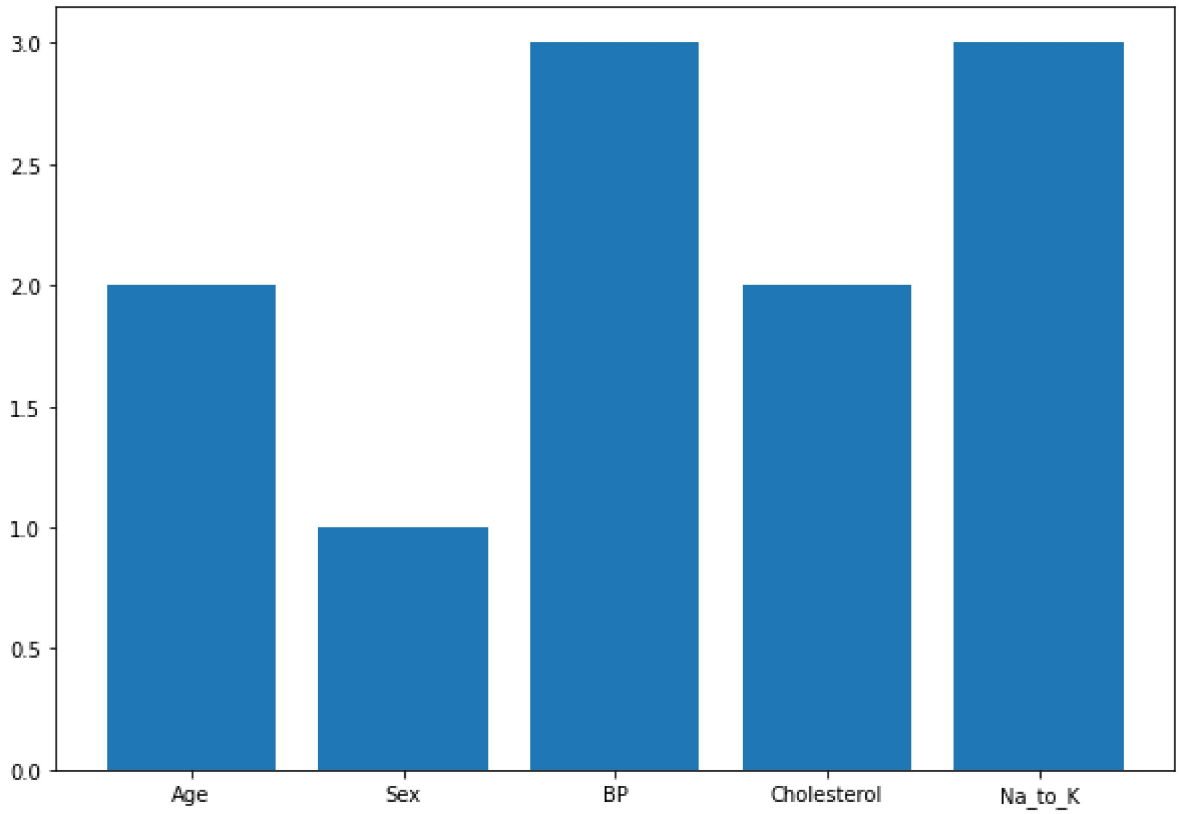
	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23.0	F	HIGH	HIGH	25.355	drugY
1	NaN	M	LOW	HIGH	13.093	drugC
2	47.0	M	LOW	HIGH	10.114	drugC
3	28.0	F	NORMAL	HIGH	NaN	drugY
4	61.0	F	LOW	HIGH	18.043	drugY

```
# Finding Count Of Null Values For Each Attribute
null_values = df.isna().sum()
null_values = null_values.to_frame().reset_index()
null_values = null_values.rename({'index': 'attribute', 0: 'count'}, axis=1)
null_values
```

	attribute	count
0	Age	2
1	Sex	1
2	BP	3
3	Cholesterol	2
4	Na_to_K	3
5	Drug	0

```
from matplotlib import pyplot as plt

attribute = null_values['attribute'].head(5)
count = null_values['count'].head(5)

fig = plt.figure(figsize =(10, 7))
plt.bar(attribute, count)
```

```
plt.show()
```




```
correlations = df.corr()
print(correlations)
```

```
                Age   Na_to_K
Age        1.000000 -0.062283
Na_to_K   -0.062283  1.000000
```

```
# Remove Rows With Null Values For Attribute Sex
df.dropna(subset = ["Sex"], inplace=True)
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23.0	F	HIGH	HIGH	25.355	drugY
1	NaN	M	LOW	HIGH	13.093	drugC
2	47.0	M	LOW	HIGH	10.114	drugC
3	28.0	F	NORMAL	HIGH	NaN	drugY
4	61.0	F	LOW	HIGH	18.043	drugY

```
df.isna().sum()
```

```
Age            2
Sex            0
BP             3
Cholesterol    2
Na_to_K        2
Drug           0
dtype: int64
```

```
# Remove Null Values From Na_to_K Attribute

#Finding The Mean Of The Column Having NaN
mean_value=df['Na_to_K'].mean()

# Fill Null Values in Na_to_K With Mean Values
df['Na_to_K'].fillna(value=mean_value, inplace=True)
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23.0	F	HIGH	HIGH	25.355000	drugY
1	NaN	M	LOW	HIGH	13.093000	drugC
2	47.0	M	LOW	HIGH	10.114000	drugC
3	28.0	F	NORMAL	HIGH	16.117355	drugY
4	61.0	F	LOW	HIGH	18.043000	drugY

```
# Mean Age Of Patients Having High Cholesterol
# Mean Age Of Patients Having Normal Cholesterol

high_cholesterol_mean = df[df['Cholesterol'] == 'HIGH']['Age'].mean(skipna=True)
normal_cholesterol_mean = df[df['Cholesterol'] == 'NORMAL']['Age'].mean(skipna=True)
print("Mean Age Of Patients Having High Cholesterol:", high_cholesterol_mean)
print("Mean Age Of Patients Having Normal Cholesterol:", normal_cholesterol_mean)
```

```
Mean Age Of Patients Having High Cholesterol: 45.23
Mean Age Of Patients Having Normal Cholesterol: 42.73684210526316
```

```
# Replace Null Cholesterol Values With Mean Age Of Patient With Respective Cholesterol Catego

df['Cholesterol'] = df['Cholesterol'].fillna('$')
for index in df.index:
  if df.at[index, 'Cholesterol'] == '$':
    age = df.at[index, 'Age']
    if abs(age-high_cholesterol_mean) <= abs(age-normal_cholesterol_mean):
      df.at[index, 'Cholesterol'] = "HIGH"
    else:
      df.at[index, 'Cholesterol'] = "NORMAL"
```

```
    print(df.at[index, 'Cholesterol'])
```

```
 HIGH
 HIGH
```

```
# Mean Age Of Patients Having High BP
# Mean Age Of Patients Having Normal BP
# Mean Age Of Patients Having Low BP

high_bp_mean = df[df['BP'] == 'HIGH']['Age'].mean(skipna=True)
low_bp_mean = df[df['BP'] == 'LOW']['Age'].mean(skipna=True)
normal_bp_mean = df[df['BP'] == 'NORMAL']['Age'].mean(skipna=True)

print("Mean Age Of Patients Having High BP:", high_bp_mean)
print("Mean Age Of Patients Having Normal BP:", normal_bp_mean)
print("Mean Age Of Patients Having Low BP:", low_bp_mean)
```

```
 Mean Age Of Patients Having High BP: 42.578947368421055
 Mean Age Of Patients Having Normal BP: 43.771929824561404
 Mean Age Of Patients Having Low BP: 46.90163934426229
```

```
# Replace Null BP Values With Category With Respect To Age Group

df['BP'] = df['BP'].fillna('$')
for index in df.index:
  if df.at[index, 'BP'] == '$':
    age = df.at[index, 'Age']
    if abs(age-high_bp_mean) <= abs(age-normal_bp_mean) and abs(age-high_bp_mean) <= abs(age-
      df.at[index, 'BP'] = "HIGH"
    elif abs(age-normal_bp_mean) <= abs(age-high_bp_mean) and abs(age-normal_bp_mean) <= abs(
      df.at[index, 'BP'] = "NORMAL"
    else:
      df.at[index, 'BP'] = "LOW"
    print(index, df.at[index, 'BP'])
```

```
 15 HIGH
 24 HIGH
 35 LOW
```

```
# Replace Null Cholesterol Values With Category With Respect To Age Group

df['Age'] = df['Age'].fillna('$')
for index in df.index:
  if df.at[index, 'Age'] == '$':
    Cholesterol = df.at[index, 'Cholesterol']
    if Cholesterol == "HIGH":
      df.at[index, 'Age'] = int(high_cholesterol_mean)
    else:
      df.at[index, 'Age'] = int(normal_cholesterol_mean)
    print(index, df.at[index, 'Age'])
```

```
1 45
16 42
```

```
# Convert Data Type Of A DataFrame Column From Float To Str
def change_dtype(value):
    try:
        return str(value)
    except ValueError:
        try:
            return str(value)
        except ValueError:
            return value

df.loc[:, 'Na_to_K'] = df['Na_to_K'].apply(change_dtype)


# Categorizing Age Values Into Respective Group
'''
For Age:
  less than 20
  20-40
  40-60
  Above 60
'''

for index in df.index:
  if df.at[index, 'Age'] < 20:
    df.at[index, 'Age'] = 'Less than 20'
  elif df.at[index, 'Age'] >= 20 and df.at[index, 'Age'] < 40:
    df.at[index, 'Age'] = '20-40'
  elif df.at[index, 'Age'] >= 40 and df.at[index, 'Age'] < 60:
    df.at[index, 'Age'] = '40-60'
  elif df.at[index, 'Age'] >=60:
    df.at[index, 'Age'] = 'More than 60'

df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	20-40	F	HIGH	HIGH	25.355	drugY
1	40-60	M	LOW	HIGH	13.093	drugC
2	40-60	M	LOW	HIGH	10.113999999999999	drugC
3	20-40	F	NORMAL	HIGH	16.11735532994923	drugY
4	More than 60	F	LOW	HIGH	18.043	drugY

```
# Categorize Na_To_K Into Respective Groups
```

```
'''
For NA_TO_K:
  LESS THAN 10
  10-15
  15-20
  20-25
  Above 25
'''

for index in df.index:
  try:
    if float(df.at[index, 'Na_to_K']) < 10:
      df.at[index, 'Na_to_K'] = 'Less than 10'
    elif float(df.at[index, 'Na_to_K']) >= 10 and float(df.at[index, 'Na_to_K']) < 15:
      df.at[index, 'Na_to_K'] = '10-15'
    elif float(df.at[index, 'Na_to_K']) >= 15 and float(df.at[index, 'Na_to_K']) < 20:
      df.at[index, 'Na_to_K'] = '15-20'
    elif float(df.at[index, 'Na_to_K']) >= 20 and float(df.at[index, 'Na_to_K']) < 25:
      df.at[index, 'Na_to_K'] = '20-25'
    elif float(df.at[index, 'Na_to_K']) >=25:
      df.at[index, 'Na_to_K'] = 'More than 25'
  except:
    #print(df.at[index, 'Na_to_K'])
    break
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	20-40	F	HIGH	HIGH	More than 25	drugY
1	40-60	M	LOW	HIGH	10-15	drugC
2	40-60	M	LOW	HIGH	10-15	drugC
3	20-40	F	NORMAL	HIGH	15-20	drugY
4	More than 60	F	LOW	HIGH	15-20	drugY

DECISION TREE PLOTTING

```
# Convert DataFrame Into List Of Dictionaries For Ease Of Data Manipulation
import math
import pandas as pd

data = df.to_dict('records')
count = 1
ROW_INDEX_ATTRIBUTE = 'INDEX'
for i in data:
  i[ROW_INDEX_ATTRIBUTE] = str(count)
  for j in i:
```

```
        i[j] = str(i[j])
      count += 1
    data[0:3]
```

```
[{'Age': '20-40',
  'BP': 'HIGH',
  'Cholesterol': 'HIGH',
  'Drug': 'drugY',
  'INDEX': '1',
  'Na_to_K': 'More than 25',
  'Sex': 'F'},
 {'Age': '40-60',
  'BP': 'LOW',
  'Cholesterol': 'HIGH',
  'Drug': 'drugC',
  'INDEX': '2',
  'Na_to_K': '10-15',
  'Sex': 'M'},
 {'Age': '40-60',
  'BP': 'LOW',
  'Cholesterol': 'HIGH',
  'Drug': 'drugC',
  'INDEX': '3',
  'Na_to_K': '10-15',
  'Sex': 'M'}]
```

```
# Define Target Attribute And Target Classes

TARGET_ATTRIBUTE = 'Drug'
TARGET_CLASSES = ['drugA', 'drugB', 'drugC', 'drugX', 'drugY']


# Define Functions For Building Decicion Tree, Finding Entropy, Finding Information Gain and

class Node:
    def __init__(self, data, parent = '', removed_attrib = []):
        self.data = data
        self.parent = parent
        self.removed_attrib = removed_attrib
        self.children = []
        self.name = ''
        self.target = TARGET_ATTRIBUTE

    def printData(self):
        header = self.data[0].keys()
        col_size = 15
        print("\n" + str('-')*((col_size+1)*len(header) + 1))

        print('|', end='')
        for i in header:
            if i == TARGET_ATTRIBUTE:
                print(TARGET_ATTRIBUTE.center(col_size) + '|', end='')
            else:
```

```
                print(i.center(col_size) + '|', end='')

        print("\n" + str('-')*((col_size+1)*len(header) + 1))

        for i in range(len(self.data)):
            output_str = '|'
            for j in header:
                output_str += (self.data[i][j]).center(col_size) + '|'
            print(output_str)

        print(str('-')*((col_size+1)*len(header) + 1))

    def findEntropy(self):
        targetClassesCount = [0 for i in range(len(TARGET_CLASSES))]
        for i in self.data:
            for j in range(len(TARGET_CLASSES)):
                if i[self.target] == TARGET_CLASSES[j]:
                    targetClassesCount[j] += 1
                    break
        totalVal = sum(targetClassesCount)
        entropy = 0

        for i in targetClassesCount:
            pi = i/totalVal
            if pi != 0:
                entropy += (-1)*pi* math.log(pi, 2)
        return entropy

    def findInfoGain(self, attribute):
        setVal = set() # Set contains all unique value of attribute
        for i in self.data:
            setVal.add(i[attribute])

        summation = 0
        for val in setVal: # find entropy of each unique value in setVal
            newData = [] # Contains only those data values with val
            for i in self.data:
                if i[attribute] == val:
                    newData.append(i)
            tempNode = Node(newData)
            e = tempNode.findEntropy()
            frac = len(newData)/len(self.data)  #Sv/S
            summation += frac * e
        infoGain = self.findEntropy() - summation
        return infoGain

    def findRoot(self):
        output_str = "Removed Attrib = " + str(self.removed_attrib)

        # Removing Previously Processed Attributes
        keys = list(self.data[0].keys())
        keys.remove(ROW_INDEX_ATTRIBUTE)
```

```
            keys.remove(TARGET_ATTRIBUTE)
            attributesList = []
            for i in keys:
                if i not in self.removed_attrib:
                    attributesList.append(i)
            maxInfoGain = -1
            maxAttr = ''
            for attr in attributesList:
                infoGain = Node(self.data[::]).findInfoGain(attr)
                output_str += "\n Info Gain of " + attr + ": " + str(infoGain)
                if infoGain > maxInfoGain:
                    maxInfoGain = infoGain
                    maxAttr = attr
            if float(maxInfoGain) == float(0):
                return self.data[0][self.target]
            print(output_str)
            return maxAttr

        def buildTree(self):
            self.name = self.findRoot()
            if self.name in TARGET_CLASSES:
                print('Data Classified')
                return
            print("Node Selected: ", self.name)
            setVal = set() # Set contains all unique value of attribute
            for i in self.data:
                setVal.add(i[self.name])

            for val in setVal: # find entropy of each unique value in setVal
                newData = [] # Contains only those data values with val
                for i in self.data:
                    if i[self.name] == val:
                        newData.append(i)
                newNode = Node(newData)
                print("\n" + ("BUILD TREE FOR (" + self.name + " = " + val + ")").center(100, '-'
                newNode.printData()
                newNode.parent = self.name + ' = ' + val
                newNode.removed_attrib = self.removed_attrib[:]
                newNode.removed_attrib.append(self.name)
                self.children.append([val, newNode])
                newNode.buildTree()

        def printTree(self, indent = ''):
            if self.name not in TARGET_CLASSES: # No need to print already classified nodes
                for i in self.children:
                    if i[1]:
                        print(indent + self.name + ': ' + i[0] + ' -> ' + i[1].name)
                        i[1].printTree(indent + '    ')


    # Building Decision Tree
    root = Node(data)
```

```
root.buildTree()
```

```
Node Selected:  Cholesterol

--------------------------------BUILD TREE FOR (Cholesterol = NORMAL)----------------

-------------------------------------------------------------------------------------
|      Age      |      Sex      |       BP      |  Cholesterol  |    Na_to_K    |
-------------------------------------------------------------------------------------
|     40-60     |       M       |      LOW      |     NORMAL    |  Less than 10 |
|     40-60     |       M       |      LOW      |     NORMAL    |  Less than 10 |
|     20-40     |       M       |      LOW      |     NORMAL    |  Less than 10 |
|     20-40     |       M       |      LOW      |     NORMAL    |  Less than 10 |
|  More than 60 |       F       |      LOW      |     NORMAL    |  Less than 10 |
-------------------------------------------------------------------------------------
Data Classified

--------------------------------BUILD TREE FOR (Cholesterol = HIGH)------------------

-------------------------------------------------------------------------------------
|      Age      |      Sex      |       BP      |  Cholesterol  |    Na_to_K    |
-------------------------------------------------------------------------------------
|     20-40     |       M       |      LOW      |      HIGH     |  Less than 10 |
|     20-40     |       F       |      LOW      |      HIGH     |  Less than 10 |
|     20-40     |       M       |      LOW      |      HIGH     |  Less than 10 |
|  More than 60 |       M       |      LOW      |      HIGH     |  Less than 10 |
-------------------------------------------------------------------------------------
Data Classified

------------------------------BUILD TREE FOR (Na_to_K = More than 25)----------------

-------------------------------------------------------------------------------------
|      Age      |      Sex      |       BP      |  Cholesterol  |    Na_to_K    |
-------------------------------------------------------------------------------------
|     20-40     |       F       |      HIGH     |      HIGH     |  More than 25 |
|     20-40     |       F       |      HIGH     |     NORMAL    |  More than 25 |
|  More than 60 |       M       |     NORMAL    |      HIGH     |  More than 25 |
|     40-60     |       M       |      LOW      |     NORMAL    |  More than 25 |
|     20-40     |       F       |      HIGH     |      HIGH     |  More than 25 |
|     20-40     |       M       |      HIGH     |      HIGH     |  More than 25 |
|  More than 60 |       F       |      HIGH     |     NORMAL    |  More than 25 |
|     20-40     |       M       |     NORMAL    |      HIGH     |  More than 25 |
|  More than 60 |       M       |      LOW      |     NORMAL    |  More than 25 |
|     40-60     |       M       |      HIGH     |      HIGH     |  More than 25 |
|     20-40     |       F       |      LOW      |     NORMAL    |  More than 25 |
|  More than 60 |       F       |     NORMAL    |     NORMAL    |  More than 25 |
|     20-40     |       F       |      HIGH     |      HIGH     |  More than 25 |
|     40-60     |       F       |      LOW      |     NORMAL    |  More than 25 |
|     40-60     |       F       |      LOW      |      HIGH     |  More than 25 |
|     40-60     |       F       |      HIGH     |      HIGH     |  More than 25 |

|     20-40     |       M       |      HIGH     |     NORMAL    |  More than 25 |
|     20-40     |       M       |      HIGH     |     NORMAL    |  More than 25 |
|  More than 60 |       F       |      HIGH     |      HIGH     |  More than 25 |
|     20-40     |       M       |     NORMAL    |      HIGH     |  More than 25 |
|  Less than 20 |       F       |      HIGH     |     NORMAL    |  More than 25 |
|     40-60     |       M       |      LOW      |     NORMAL    |  More than 25 |
|     40-60     |       M       |      LOW      |     NORMAL    |  More than 25 |
```

```
|     20-40      |       M       |     NORMAL     |      HIGH      |  More than 25 |
|  More than 60  |       F       |      LOW       |     NORMAL     |  More than 25 |
|     20-40      |       F       |      HIGH      |     NORMAL     |  More than 25 |
```

```
# Final Decision Tree

print("\n\n" + "Final Decision Tree".center(100, '-') + "\n")
root.printTree()
```

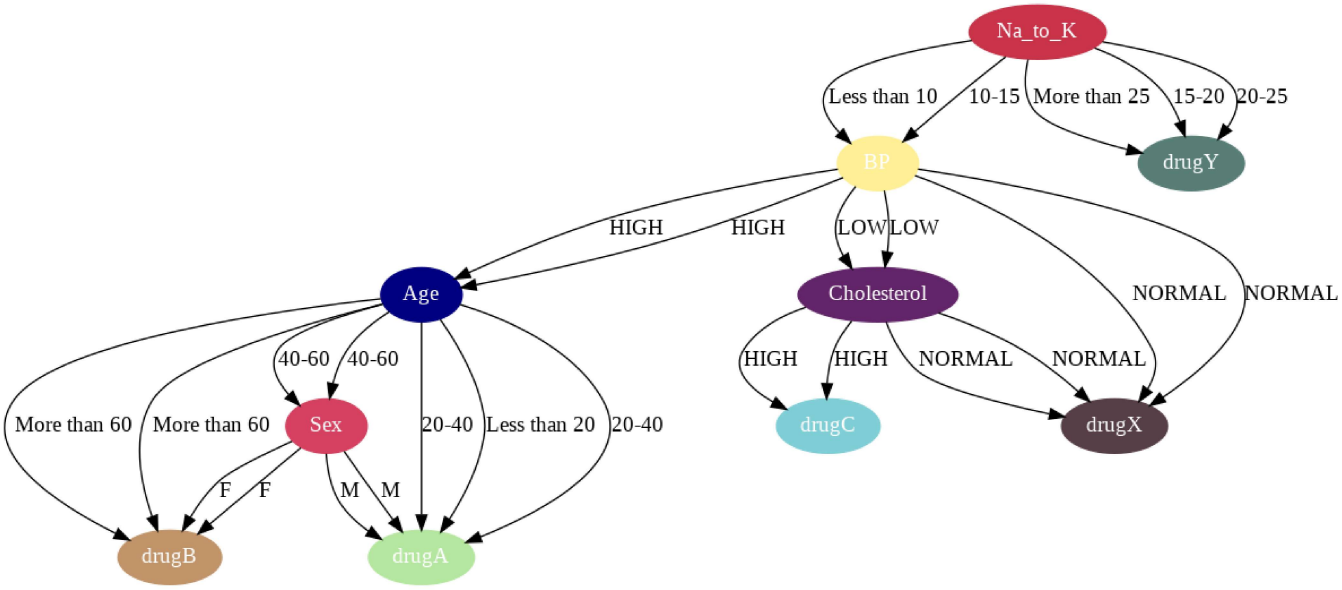
```
-----------------------------------------Final Decision Tree-----------------------------

Na_to_K: 15-20 -> drugY
Na_to_K: 20-25 -> drugY
Na_to_K: 10-15 -> BP
    BP: HIGH -> Age
        Age: More than 60 -> drugB
        Age: Less than 20 -> drugA
        Age: 20-40 -> drugA
        Age: 40-60 -> Sex
            Sex: F -> drugB
            Sex: M -> drugA
    BP: LOW -> Cholesterol
        Cholesterol: NORMAL -> drugX
        Cholesterol: HIGH -> drugC
    BP: NORMAL -> drugX
Na_to_K: Less than 10 -> BP
    BP: NORMAL -> drugX
    BP: HIGH -> Age
        Age: More than 60 -> drugB
        Age: 20-40 -> drugA
        Age: 40-60 -> Sex
            Sex: F -> drugB
            Sex: M -> drugA
    BP: LOW -> Cholesterol
        Cholesterol: NORMAL -> drugX
        Cholesterol: HIGH -> drugC
Na_to_K: More than 25 -> drugY
```

```
# Import GraphViz For Ploting Decision Tree

!pip install graphviz
import graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10
```

```
# Creating Graphviz Object And Defining Colors For Nodes Of Tree

treeObj = graphviz.Digraph(comment='Tree', filename='treeOutput')
colorMap = {
```

```
            'Age': '#000080',
            'Sex': '#d64161',
            'BP': '#ffef96',
            'Cholesterol': '#622569',
            'Na_to_K': '#c83349',
            'drugA': '#b5e7a0',
            'drugB': '#c1946a',
            'drugC': '#80ced6',
            'drugX': '#563f46',
            'drugY': '#587e76'
            }


# Creating Decision Tree And Exporting It To PNG File
queue = [root]
edges = []

for i in colorMap:
  treeObj.node(i, color=colorMap[i], fillcolor=colorMap[i],style='filled',fontcolor="white")

while len(queue) > 0:
  front = queue.pop(0)
  for i in front.children:
    queue.append(i[1])
    edges.append([front.name, i[1].name, i[0]])
    treeObj.edge(front.name, i[1].name, label=i[0])

import os
file_path = 'treeOutput.png'
if os.path.isfile(file_path):
    os.remove(file_path)
if os.path.isfile(file_path[0:-4]):
    os.remove(file_path[0:-4])

treeObj.format = 'png'
treeObj.render(view=True)

while not os.path.exists(file_path):
    time.sleep(1)


# Display The PNG Exported In Previous Step

from IPython.display import Image
Image(filename = "treeOutput.png")
```



Data Visualization

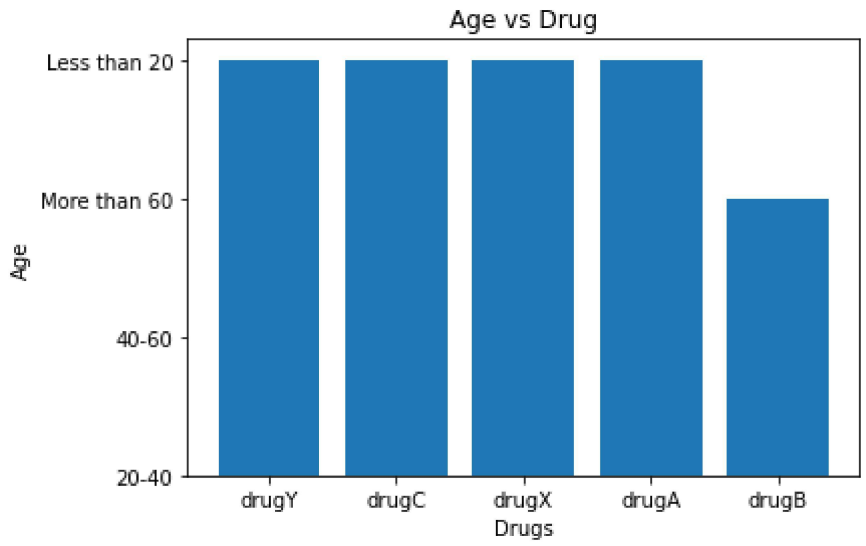
```
# Import libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

```
df.head()
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	20-40	F	HIGH	HIGH	More than 25	drugY
1	40-60	M	LOW	HIGH	10-15	drugC
2	40-60	M	LOW	HIGH	10-15	drugC
3	20-40	F	NORMAL	HIGH	15-20	drugY
4	More than 60	F	LOW	HIGH	15-20	drugY

Q1. Relation between age group and drugs consumed?

```
plt.bar(data_vis["Drug"],data_vis["Age"])
plt.xlabel('Drugs')
plt.ylabel('Age')
plt.title('Age vs Drug')
plt.show()
```



Q2. Visulaizing Drug Count

Countplot A countplot basically counts the categories and returns a count of their occurrences. It is one of the most simple plots provided by the seaborn library. Syntax:

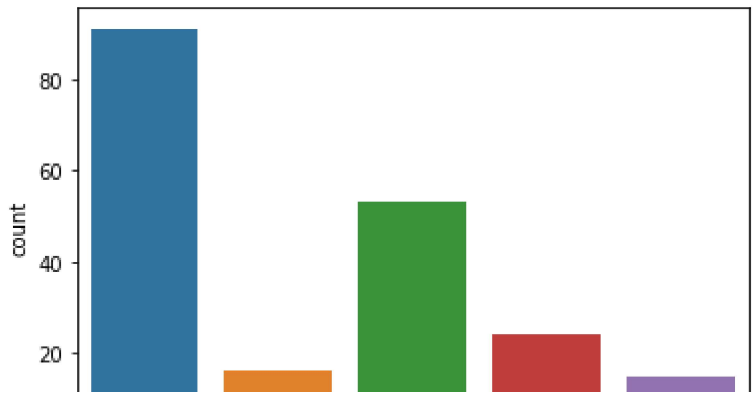
countplot([x, y, hue, data, order, ...])

```
drug_count={}
for i in df["Drug"]:
  if (i in drug_count):
    drug_count[i]+=1
  else:
    drug_count[i]=1

drug_cnt = pd.DataFrame(drug_count, index=[0])
drug_cnt
```

	drugY	drugC	drugX	drugA	drugB
0	91	16	53	24	15

```
sns.countplot(x ='Drug', data = data_vis)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29b736fa90>
```



Explanation: Order of occurence of drug in decreasing order: drugY > drugX > drugA > drugC > drugB

Q3. Relation between Drug and Na_to_K value

Violinplot

It is similar to the boxplot except that it provides a higher, more advanced visualization and uses the kernel density estimation to give a better description about the data distribution. Syntax:

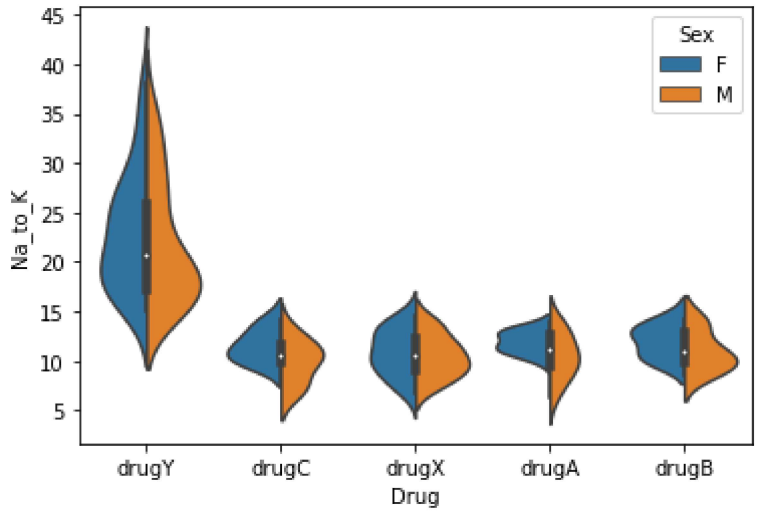
violinplot([x, y, hue, data, order, ...])

```
vis_data=pd.read_csv("drug200_mod.csv")
vis_data
```

Age	Sex	BP	Cholesterol	Na_to_K	Drug

```
sns.violinplot(x ='Drug', y ='Na_to_K', data = vis_data, hue ='Sex', split = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29b77efd90>
```

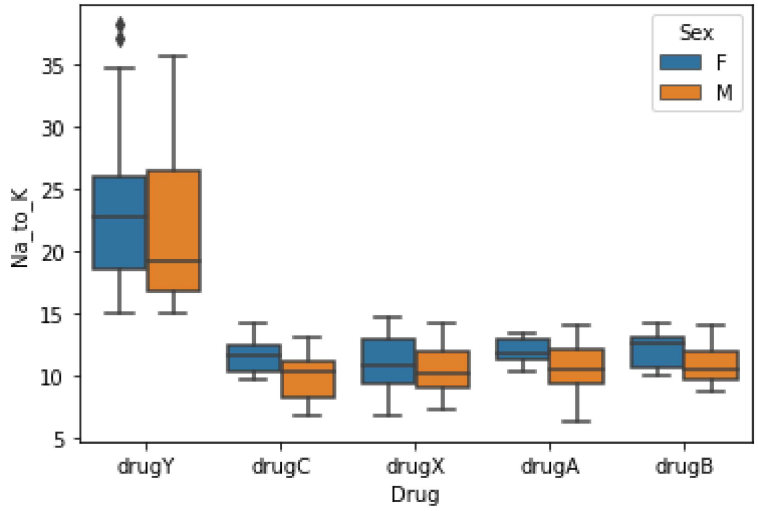


200 rows × 6 columns

Box Plot

```
sns.boxplot(x ='Drug', y ='Na_to_K', data = vis_data, hue ='Sex')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29b78ced50>
```



✓ 0s completed at 11:34 PM