import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

In [2]:

*# importing necessary module*

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

In [3]:

df = pd.read\_csv(r'/content/drug200.csv')

In [4]:

df.head()

Out[4]:

|  | 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 [5]:

df.shape

Out[5]:

(200, 6)

In [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

Columns are in correct format for analysis.

In [7]:

df.isnull().sum()

Out[7]:

Age 0

Sex 0

BP 0

Cholesterol 0

Na\_to\_K 0

Drug 0

dtype: int64

No missing value.

In [8]:

df.describe()

Out[8]:

|  | Age | Na\_to\_K |
| --- | --- | --- |
| count | 200.000000 | 200.000000 |
| mean | 44.315000 | 16.084485 |
| std | 16.544315 | 7.223956 |
| min | 15.000000 | 6.269000 |
| 25% | 31.000000 | 10.445500 |
| 50% | 45.000000 | 13.936500 |
| 75% | 58.000000 | 19.380000 |
| max | 74.000000 | 38.247000 |

In [9]:

df.drop\_duplicates()

Out[9]:

|  | 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

No duplicate value

EDA

Age

In [10]:

plt.figure(figsize=(6,2))

sns.boxplot(x=df['Age'], palette='Blues\_r')

plt.show()

In [ ]:

In [11]:

plt.figure(figsize=(5,3))

sns.histplot(x=df['Age'], bins=20,kde=True)

plt.show()

Na\_to\_K

In [12]:

plt.figure(figsize=(6,2))

sns.boxplot(x=df['Na\_to\_K'], palette='Blues\_r')

plt.show()

In [13]:

plt.figure(figsize=(5,3))

sns.histplot(x=df['Na\_to\_K'], bins=20,kde=True)

plt.show()

SEX

In [14]:

df.Sex.unique()

Out[14]:

array(['F', 'M'], dtype=object)

In [15]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['Sex'], palette='Blues\_r')

plt.show()

BP

In [16]:

df.BP.unique()

Out[16]:

array(['HIGH', 'LOW', 'NORMAL'], dtype=object)

In [17]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['BP'], palette='Blues\_r')

plt.show()

In [18]:

df.Cholesterol.unique()

Out[18]:

array(['HIGH', 'NORMAL'], dtype=object)

In [19]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['Cholesterol'], palette='Blues\_r')

plt.show()

Age - Drug

In [20]:

plt.figure(figsize=(5,3))

sns.scatterplot(y=df['Age'], x=df['Drug'], hue=df['Sex'])

plt.show()

Na\_to\_K - Drug

In [21]:

plt.figure(figsize=(5,3))

sns.scatterplot(y=df['Na\_to\_K'], x=df['Drug'])

plt.show()

Sex - Drug

In [22]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['Drug'], palette='Blues\_r', hue=df['Sex'])

plt.show()

BP - Drug

In [23]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['Drug'], palette='Blues\_r', hue=df['BP'])

plt.show()

In [ ]:

Cholesterol - Drug

In [24]:

plt.figure(figsize=(5,3))

sns.countplot(x=df['Drug'], palette='Blues\_r', hue=df['Cholesterol'])

plt.show()

FE

Data binning

age and na\_to\_k

In [25]:

cut\_labels = ['teenage', 'young adullt', 'mid-age adult', 'old']

cut\_bins = [0, 18, 35, 55, df.Age.max()]

df['age\_cat'] = pd.cut(df['Age'], bins=cut\_bins, labels=cut\_labels)

df.drop('Age', axis=1, inplace= True)

df

Out[25]:

|  | Sex | BP | Cholesterol | Na\_to\_K | Drug | age\_cat |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | F | HIGH | HIGH | 25.355 | DrugY | young adullt |
| 1 | M | LOW | HIGH | 13.093 | drugC | mid-age adult |
| 2 | M | LOW | HIGH | 10.114 | drugC | mid-age adult |
| 3 | F | NORMAL | HIGH | 7.798 | drugX | young adullt |
| 4 | F | LOW | HIGH | 18.043 | DrugY | old |
| ... | ... | ... | ... | ... | ... | ... |
| 195 | F | LOW | HIGH | 11.567 | drugC | old |
| 196 | M | LOW | HIGH | 12.006 | drugC | teenage |
| 197 | M | NORMAL | HIGH | 9.894 | drugX | mid-age adult |
| 198 | M | NORMAL | NORMAL | 14.020 | drugX | young adullt |
| 199 | F | LOW | NORMAL | 11.349 | drugX | mid-age adult |

200 rows × 6 columns

In [26]:

cut\_labels = ['Low','Normal','High']

df['Na\_to\_K\_bin'] = pd.qcut(df.Na\_to\_K, q=3,labels=cut\_labels)

df.drop('Na\_to\_K', axis=1, inplace= True)

df

Out[26]:

|  | Sex | BP | Cholesterol | Drug | age\_cat | Na\_to\_K\_bin |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | F | HIGH | HIGH | DrugY | young adullt | High |
| 1 | M | LOW | HIGH | drugC | mid-age adult | Normal |
| 2 | M | LOW | HIGH | drugC | mid-age adult | Low |
| 3 | F | NORMAL | HIGH | drugX | young adullt | Low |
| 4 | F | LOW | HIGH | DrugY | old | High |
| ... | ... | ... | ... | ... | ... | ... |
| 195 | F | LOW | HIGH | drugC | old | Normal |
| 196 | M | LOW | HIGH | drugC | teenage | Normal |
| 197 | M | NORMAL | HIGH | drugX | mid-age adult | Low |
| 198 | M | NORMAL | NORMAL | drugX | young adullt | Normal |
| 199 | F | LOW | NORMAL | drugX | mid-age adult | Low |

200 rows × 6 columns

In [27]:

input\_feature = df.drop(columns=['Drug'],axis=1)

target\_feature = df['Drug']

In [28]:

input\_feature

Out[28]:

|  | Sex | BP | Cholesterol | age\_cat | Na\_to\_K\_bin |
| --- | --- | --- | --- | --- | --- |
| 0 | F | HIGH | HIGH | young adullt | High |
| 1 | M | LOW | HIGH | mid-age adult | Normal |
| 2 | M | LOW | HIGH | mid-age adult | Low |
| 3 | F | NORMAL | HIGH | young adullt | Low |
| 4 | F | LOW | HIGH | old | High |
| ... | ... | ... | ... | ... | ... |
| 195 | F | LOW | HIGH | old | Normal |
| 196 | M | LOW | HIGH | teenage | Normal |
| 197 | M | NORMAL | HIGH | mid-age adult | Low |
| 198 | M | NORMAL | NORMAL | young adullt | Normal |
| 199 | F | LOW | NORMAL | mid-age adult | Low |

200 rows × 5 columns

OneHotEncoding

In [29]:

input\_feature = pd.get\_dummies(input\_feature)

input\_feature

Out[29]:

|  | Sex\_F | Sex\_M | BP\_HIGH | BP\_LOW | BP\_NORMAL | Cholesterol\_HIGH | Cholesterol\_NORMAL | age\_cat\_teenage | age\_cat\_young adullt | age\_cat\_mid-age adult | age\_cat\_old | Na\_to\_K\_bin\_Low | Na\_to\_K\_bin\_Normal | Na\_to\_K\_bin\_High |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 3 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 195 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 196 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 197 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 198 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 199 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

200 rows × 14 columns

Applying Model

In [30]:

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(input\_feature,target\_feature,test\_size=0.2)

In [31]:

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators = 100, random\_state = 0)

model.fit(x\_train, y\_train)

Out[31]:

RandomForestClassifier(random\_state=0)

Analysing Model Performance

1. Presicion
2. Recall
3. Accuracy

In [32]:

from sklearn import metrics

y\_pred = model.predict(x\_test)

print(metrics.classification\_report(y\_test, y\_pred))

precision recall f1-score support

DrugY 0.94 0.83 0.88 18

drugA 1.00 0.50 0.67 4

drugB 0.67 1.00 0.80 4

drugC 1.00 1.00 1.00 1

drugX 0.87 1.00 0.93 13

accuracy 0.88 40

macro avg 0.89 0.87 0.86 40

weighted avg 0.90 0.88 0.87 40

In [33]:

model.score(x\_test, y\_test)

Out[33]:

0.875