Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

Reading Data Set

In [2]:
```
#loading the dataset in pandas dataframe
df_drug = pd.read_csv("drug200.csv")
```

In [3]:
```
#check first five rows of the dataset
df_drug.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

In [4]:
```
#check last five rows of the dataset
df_drug.tail()
```

Out[4]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
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

In [5]:
```
#check shape of the dataset
df_drug.shape
```

Out[5]: (200, 6)

In [6]:
```
#check more infomation of the dataset
df_drug.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
```

In [7]:
```
#check mathamic realtion ship of the dataset
df_drug.describe()
```

Out[7]:

	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 [8]:
```
#check corr realtion of the dataset
df_drug.corr()
```

Out[8]:

	Age	Na_to_K
Age	1.000000	-0.063119
Na_to_K	-0.063119	1.000000

In [9]:
```
#check missing value of the dataset
df_drug.isnull().sum()
```

Out[9]:
```
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
```

Categorical Variables

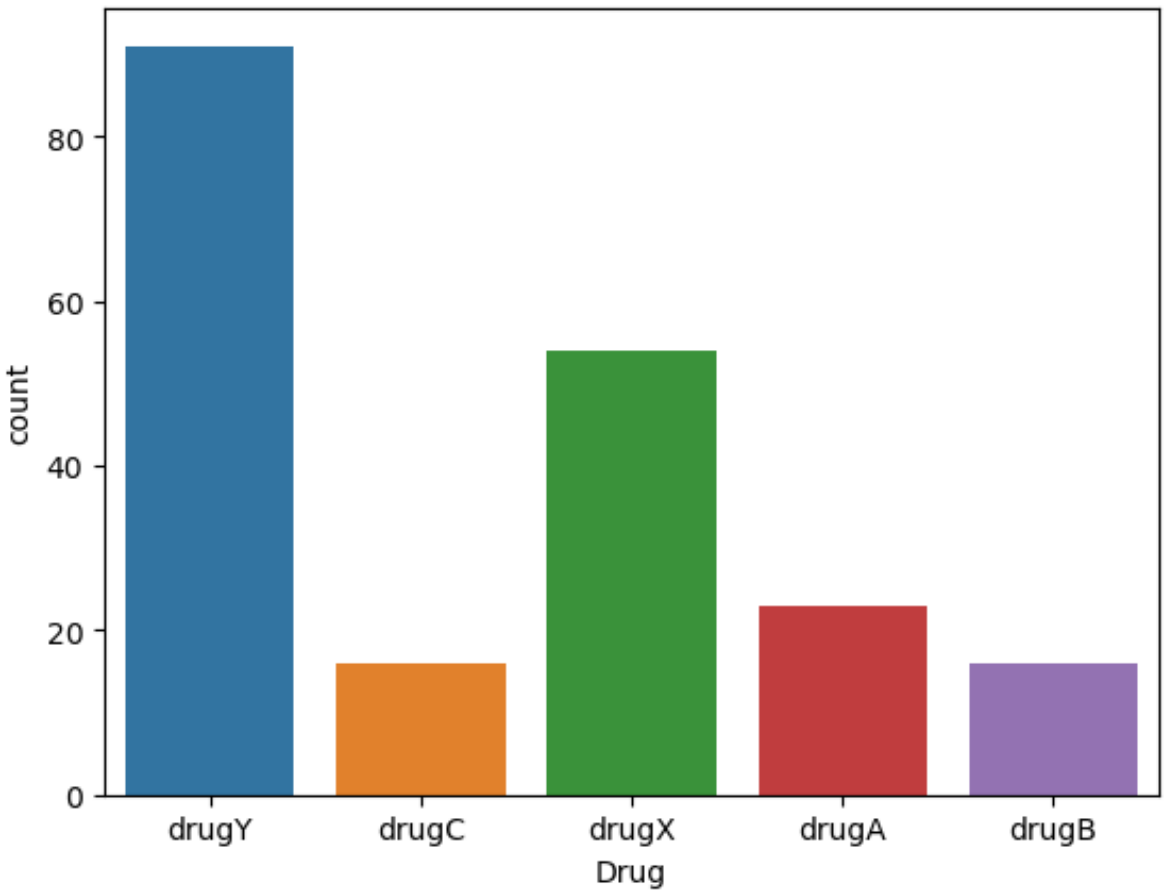
In [10]:
```
#count the value Drug
df_drug["Drug"].value_counts()
```

Out[10]:
```
drugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: Drug, dtype: int64
```

In [11]:
```
sns.countplot(df_drug["Drug"])
```

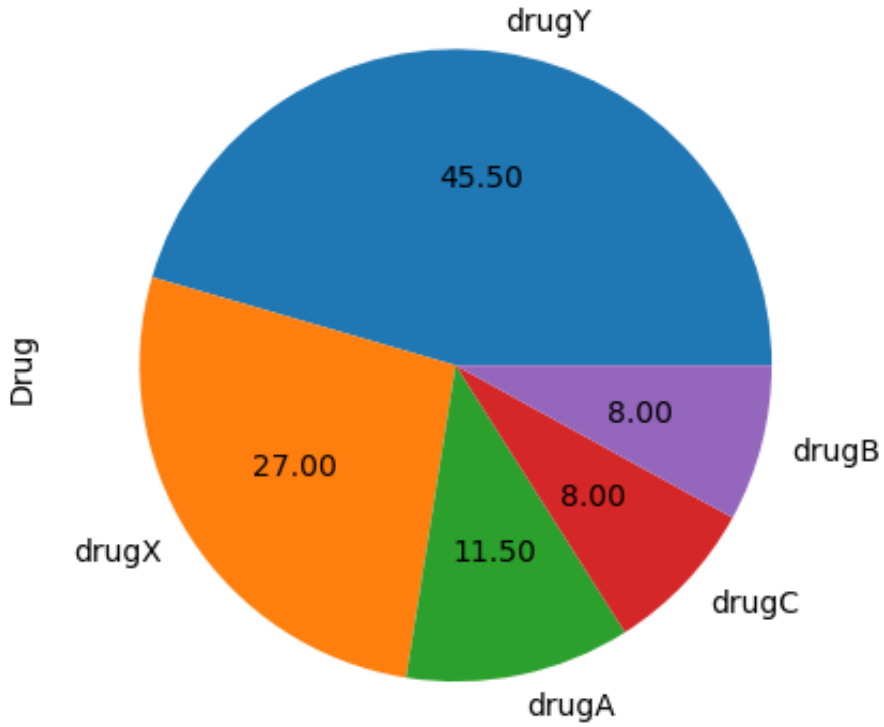
```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[11]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [12]:
```
df_drug["Drug"].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[12]: <AxesSubplot:ylabel='Drug'>



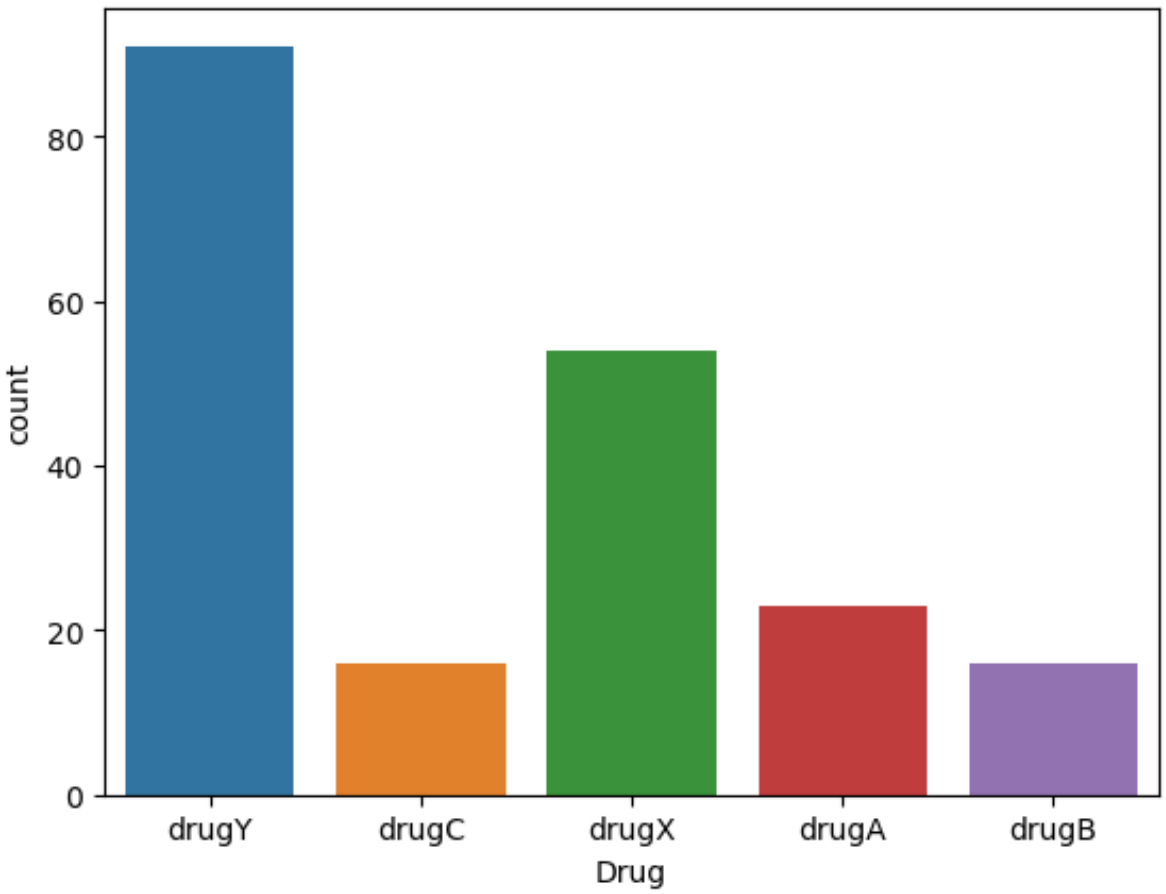
In [14]:
```
#count the value of sex
df_drug["Sex"].value_counts()
```

Out[14]:
```
M    104
F     96
Name: Sex, dtype: int64
```

In [15]:
```
sns.countplot(df_drug["Drug"])
```

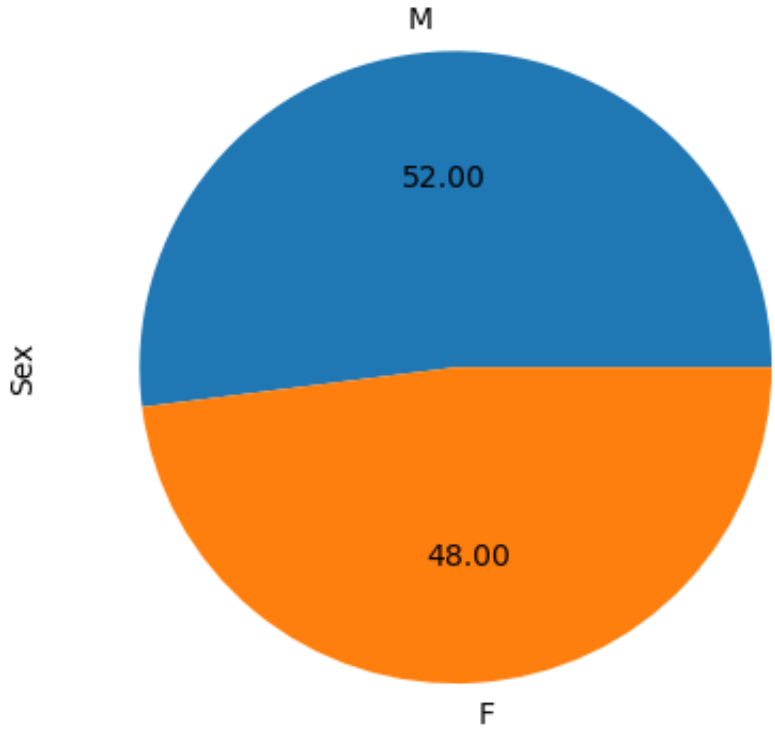
```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[15]: <AxesSubplot:xlabel='Drug', ylabel='count'>



In [16]:
```
df_drug["Sex"].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[16]: <AxesSubplot:ylabel='Sex'>



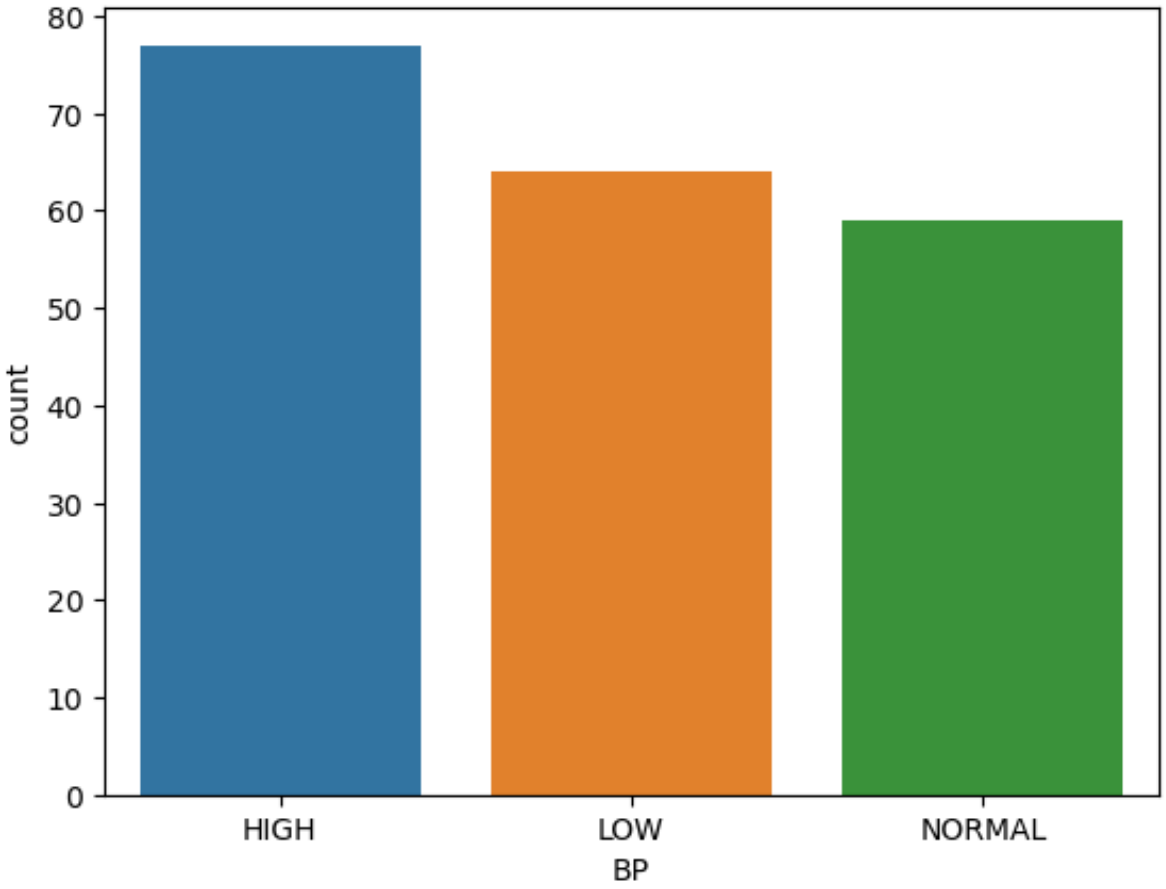
In [18]:
```
#cout the value of BP
df_drug["BP"].value_counts()
```

Out[18]:
```
HIGH      77
LOW       64
NORMAL    59
Name: BP, dtype: int64
```

In [19]:
```
sns.countplot(df_drug["BP"])
```

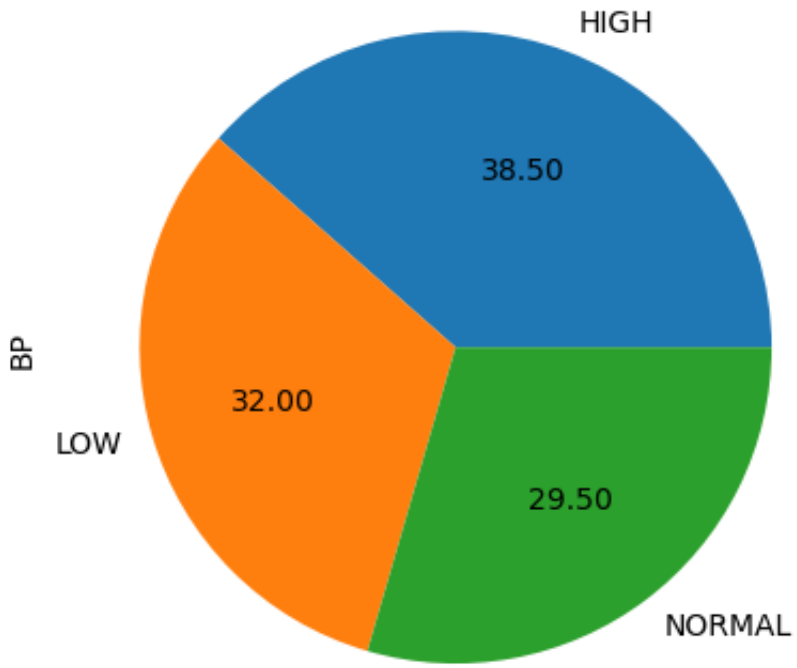
```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[19]: <AxesSubplot:xlabel='BP', ylabel='count'>



In [20]:
```
df_drug["BP"].value_counts().plot(kind='pie',autopct='%.2f')
```

Out[20]: <AxesSubplot:ylabel='BP'>



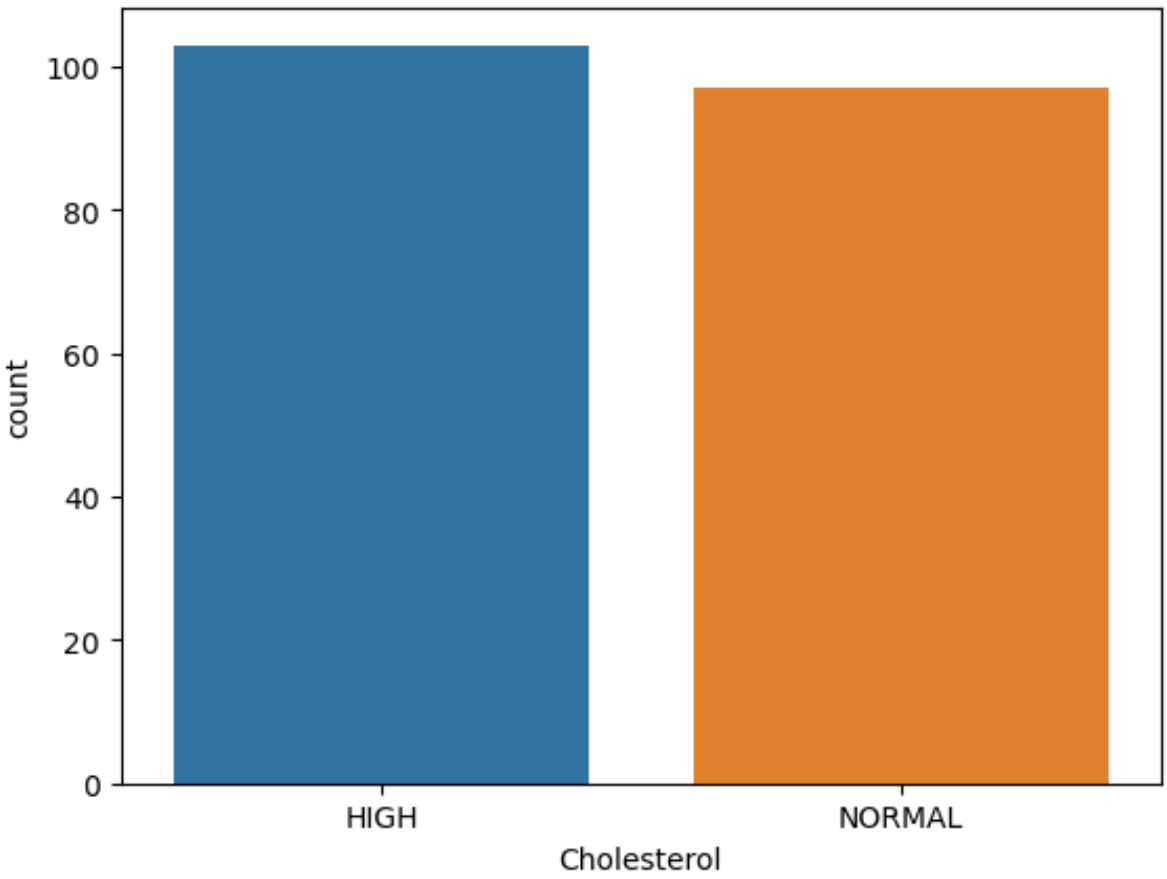
In [21]:
```
#count the value of cholesterol
df_drug["Cholesterol"].value_counts()
```

Out[21]:
```
HIGH      103
NORMAL     97
Name: Cholesterol, dtype: int64
```

In [22]:
```
sns.countplot(df_drug["Cholesterol"])
```

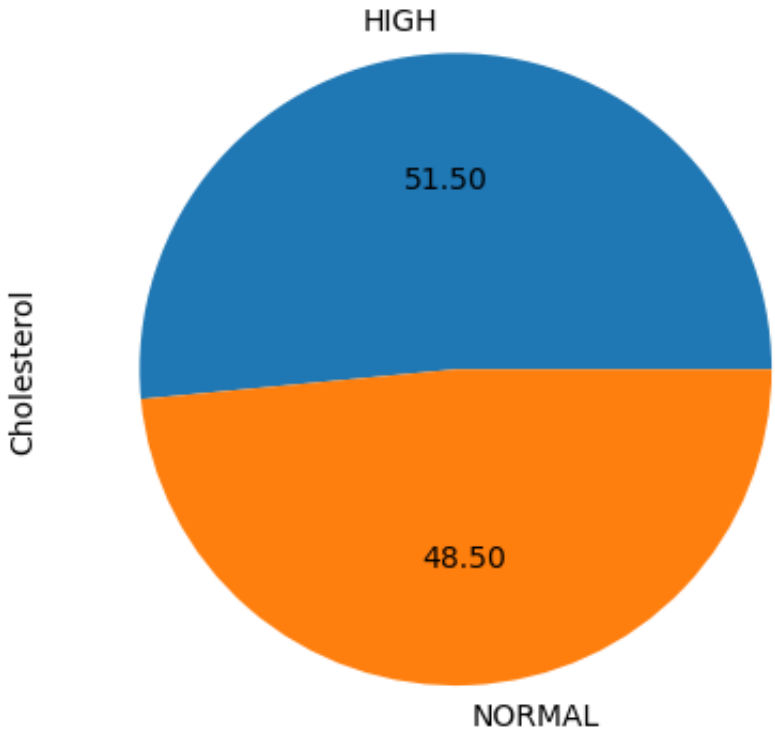
```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[22]: <AxesSubplot:xlabel='Cholesterol', ylabel='count'>



In [23]:
```
df_drug["Cholesterol"].value_counts().plot(kind='pie',autopct='%.2f
```

Out[23]: <AxesSubplot:ylabel='Cholesterol'>



Numerical Variables

In [24]:
```
df_drug.describe()
```

Out[24]:

	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 [25]:
```
skewAge = df_drug.Age.skew(axis = 0, skipna = True)
print('Age skewness: ', skewAge)
```

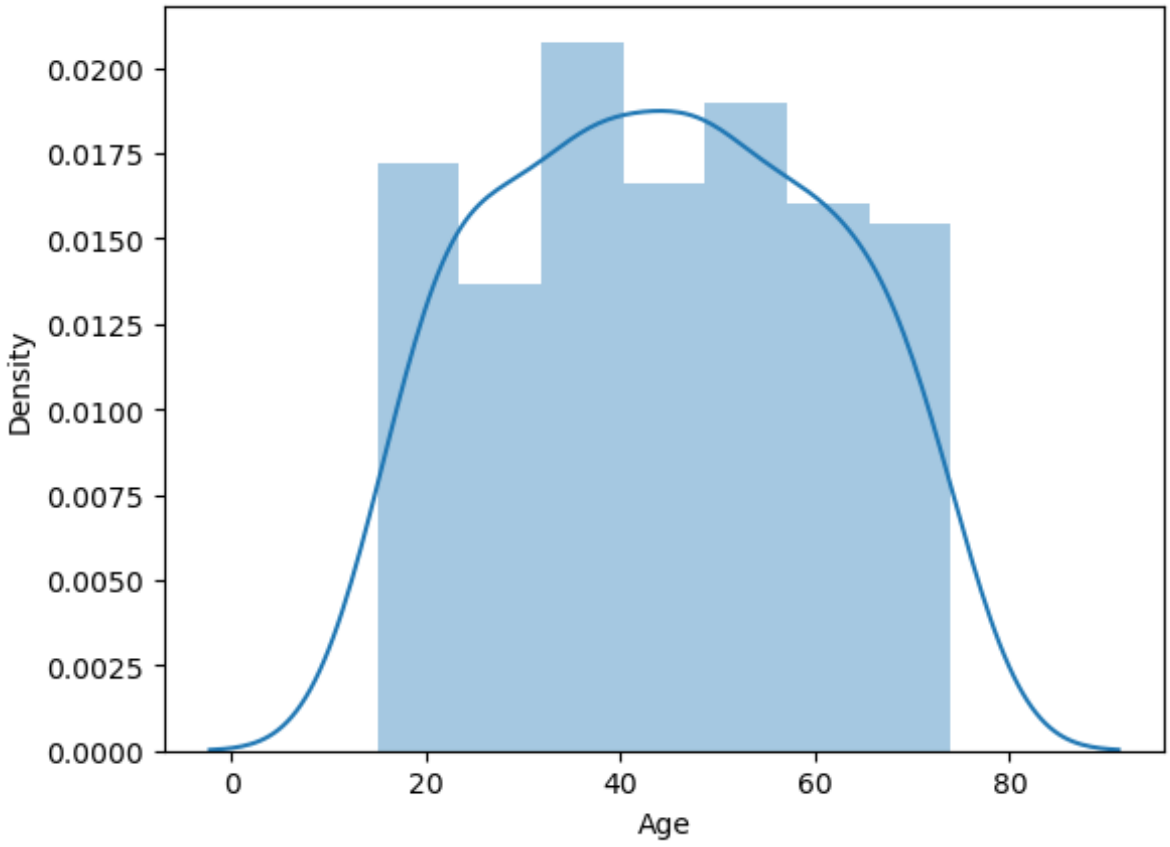
```
Age skewness:  0.03030835703000607
```

In [26]:
```
skewNatoK = df_drug.Na_to_K.skew(axis = 0, skipna = True)
print('Na to K skewness: ', skewNatoK)
```

```
Na to K skewness:  1.039341186028881
```

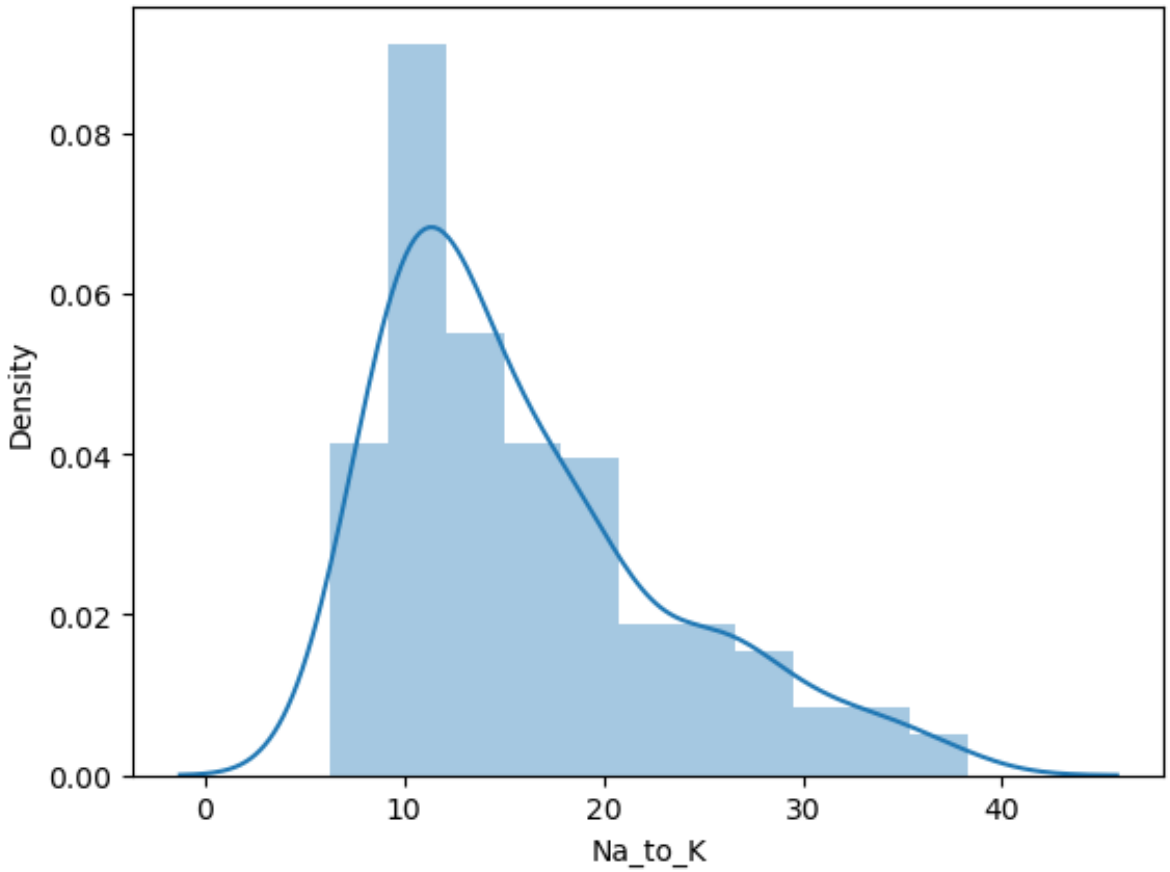
In [27]:
```
sns.distplot(df_drug['Age']);
```

```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [28]:
```
sns.distplot(df_drug['Na_to_K']);
```

```
/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

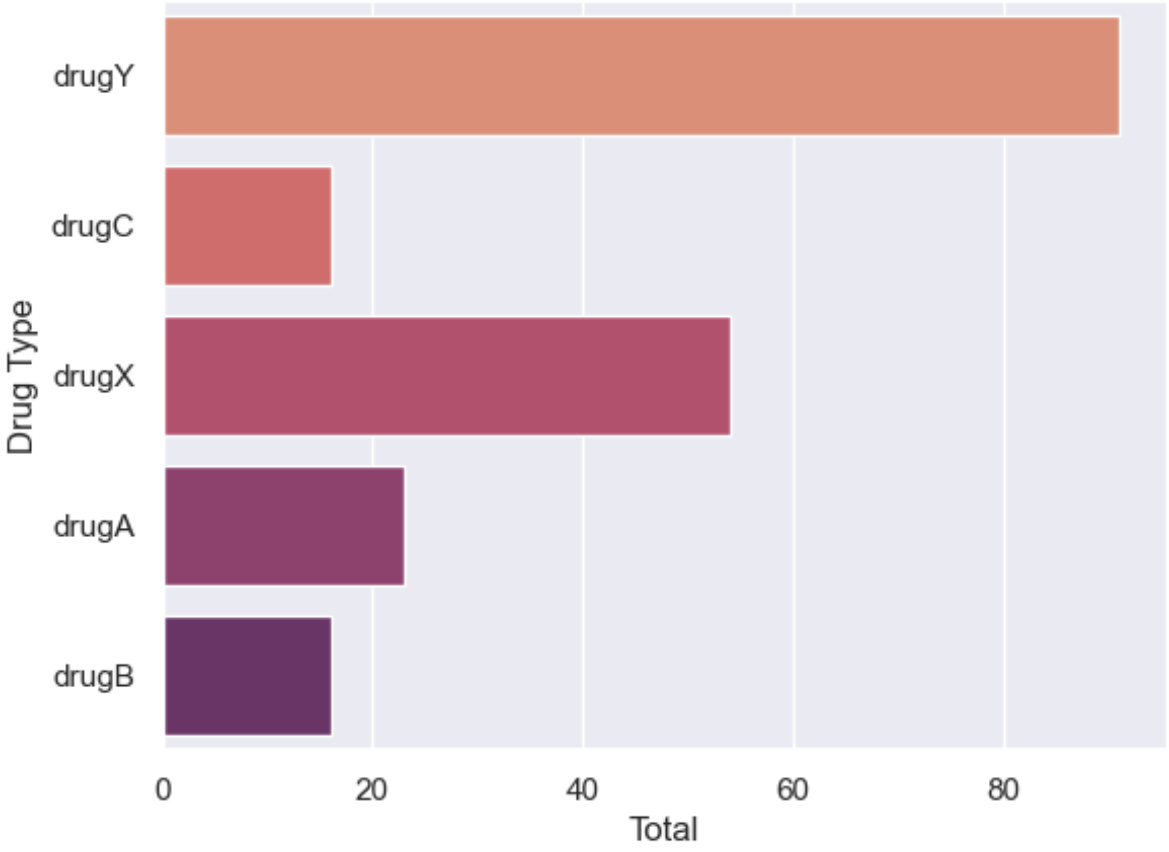


EDA

This section will explore variables in the dataset using different various plots/charts.

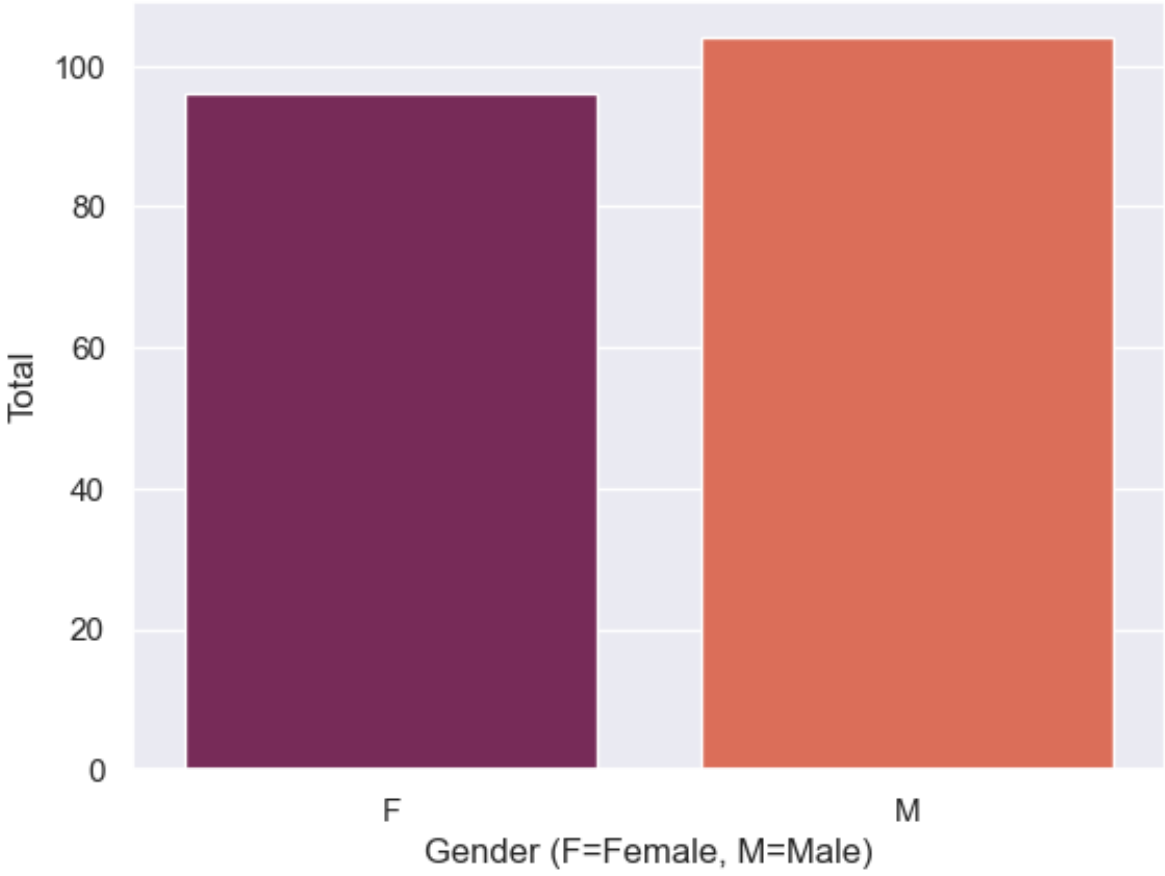
Drug Type Distribution 💊

In [29]:
```
sns.set_theme(style="darkgrid")
sns.countplot(y="Drug", data=df_drug, palette="flare")
plt.ylabel('Drug Type')
plt.xlabel('Total')
plt.show()
```



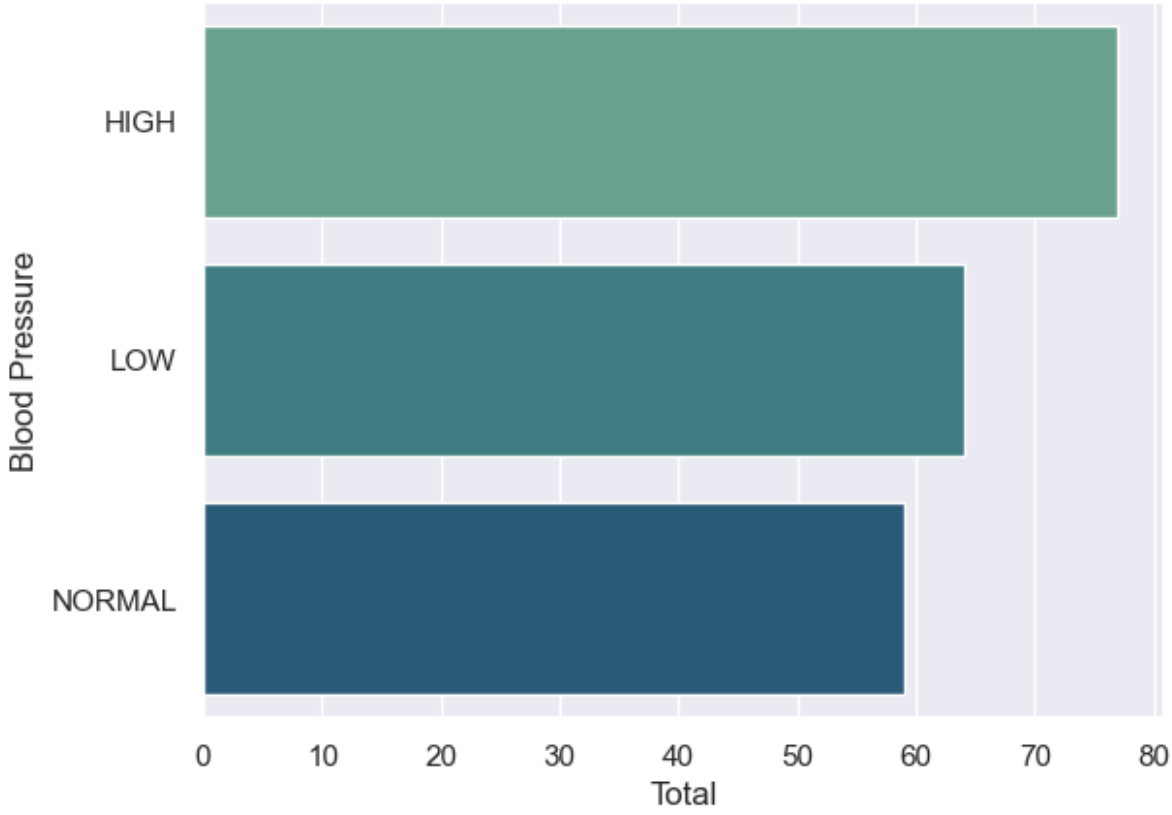
Gender Distribution

In [30]:
```
sns.set_theme(style="darkgrid")
sns.countplot(x="Sex", data=df_drug, palette="rocket")
plt.xlabel('Gender (F=Female, M=Male)')
plt.ylabel('Total')
plt.show()
```



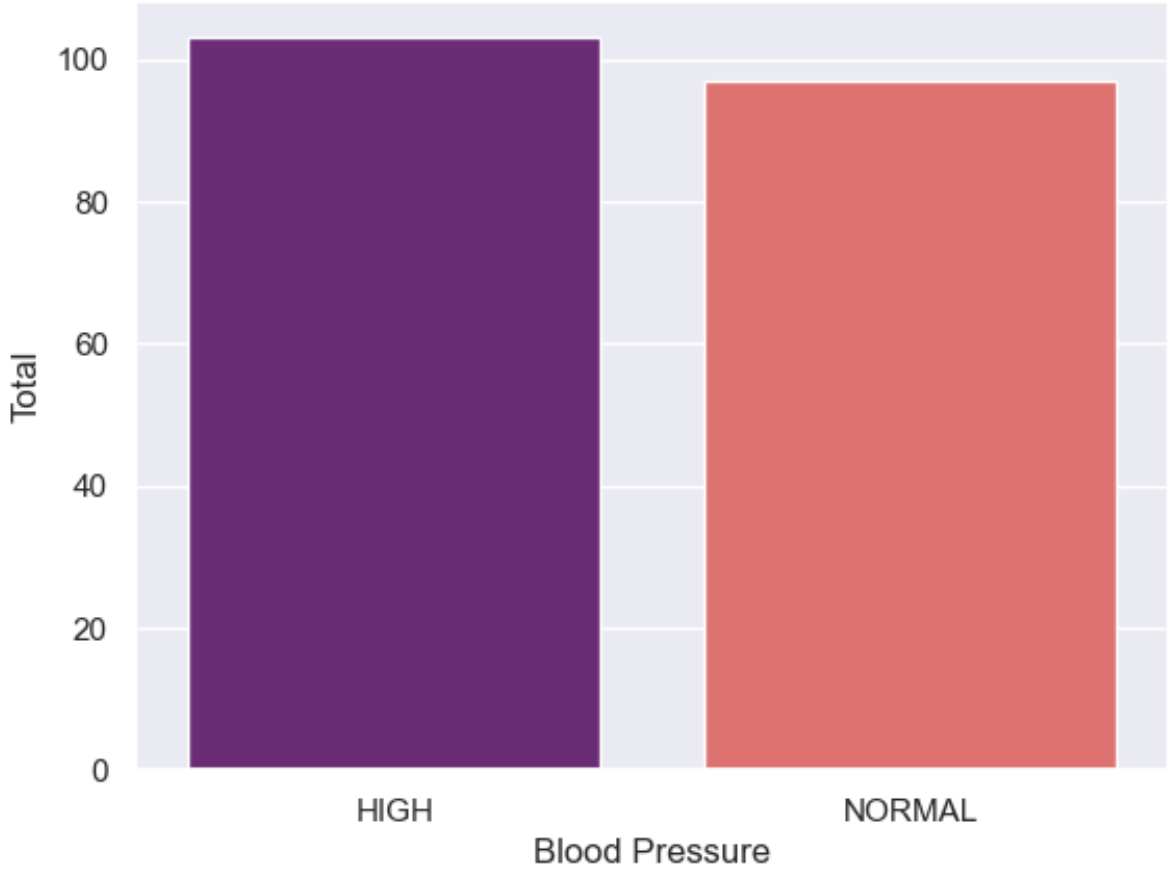
Blood Pressure Distribution 🩸

In [31]:
```
sns.set_theme(style="darkgrid")
sns.countplot(y="BP", data=df_drug, palette="crest")
plt.ylabel('Blood Pressure')
plt.xlabel('Total')
plt.show()
```



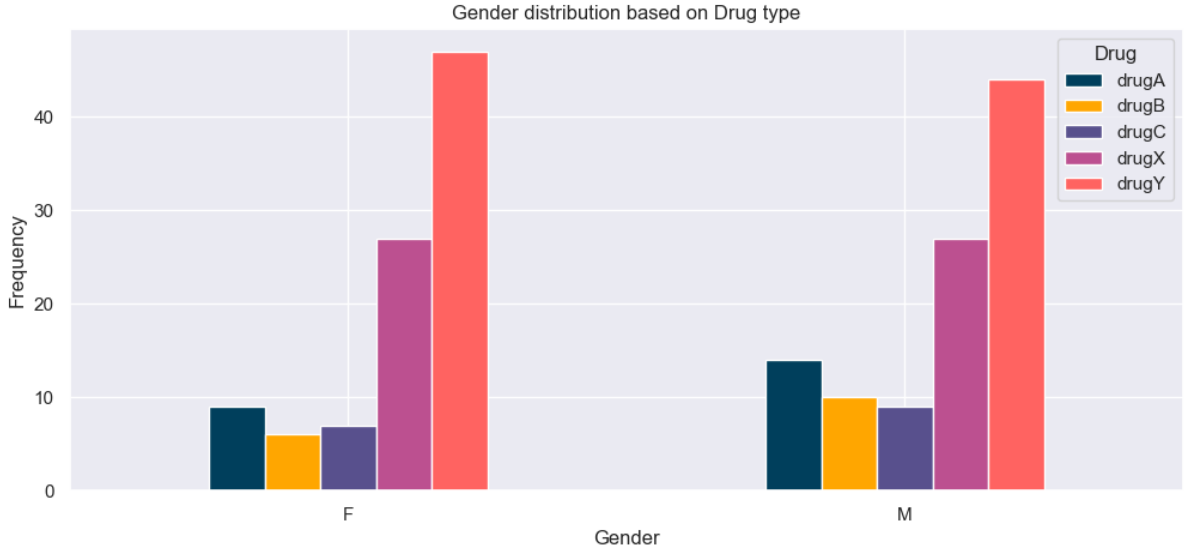
Cholesterol Distribution 🥛

In [32]:
```
sns.set_theme(style="darkgrid")
sns.countplot(x="Cholesterol", data=df_drug, palette="magma")
plt.xlabel('Blood Pressure')
plt.ylabel('Total')
plt.show()
```



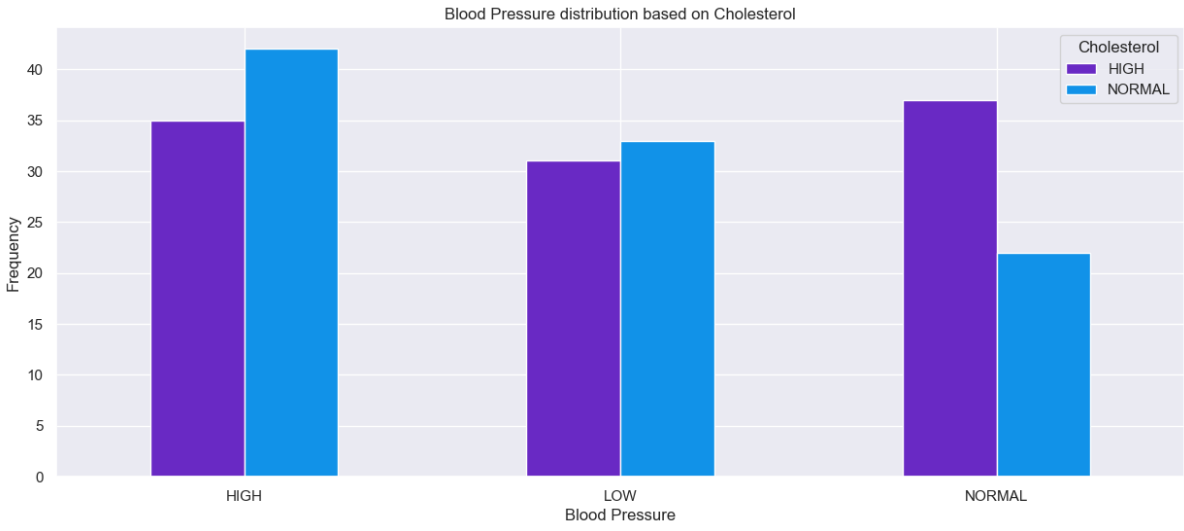
Gender Distribution based on Drug Type

In [33]:
```
pd.crosstab(df_drug.Sex,df_drug.Drug).plot(kind="bar",figsize=(12,5
plt.title('Gender distribution based on Drug type')
plt.xlabel('Gender')
plt.xticks(rotation=0)
plt.ylabel('Frequency')
plt.show()
```



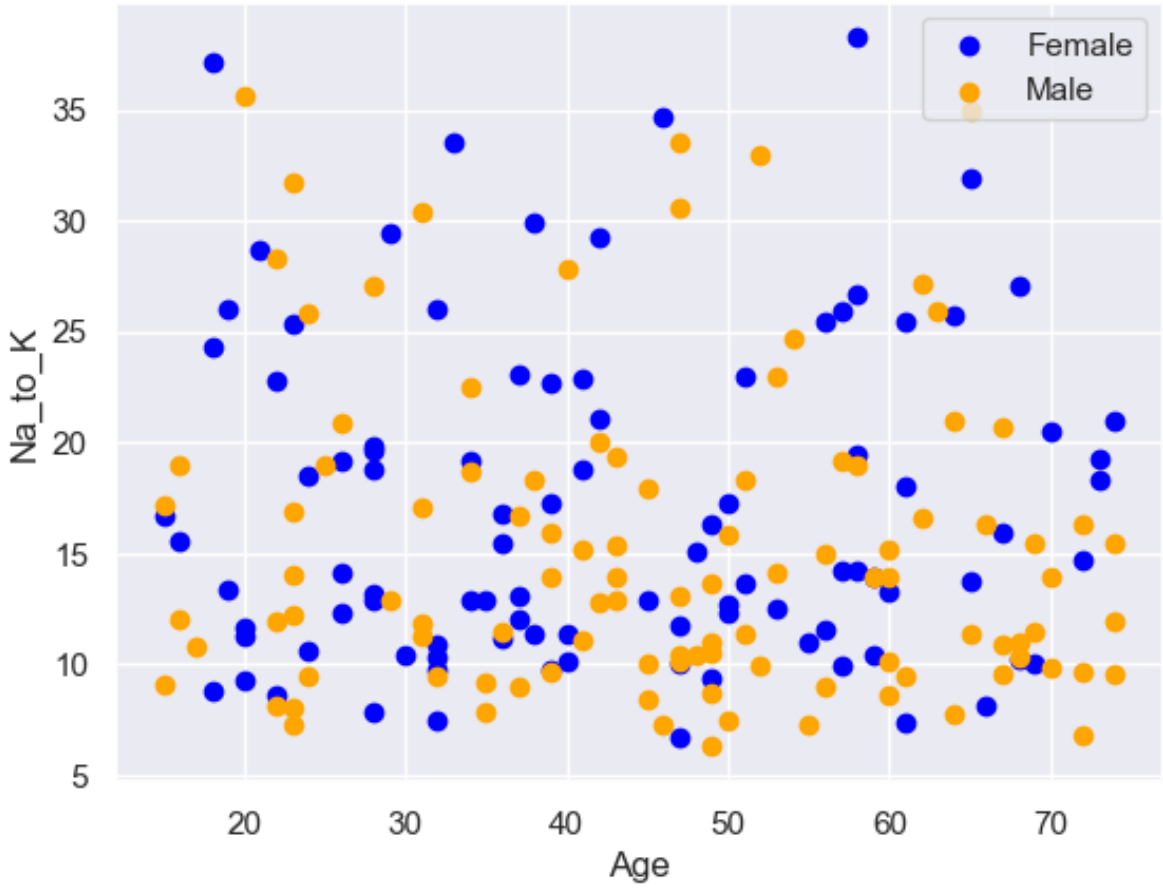
Blood Pressure Distribution based on Cholesetrol 🩸🥛

In [34]:
```
pd.crosstab(df_drug.BP,df_drug.Cholesterol).plot(kind="bar",figsize
plt.title('Blood Pressure distribution based on Cholesterol')
plt.xlabel('Blood Pressure')
plt.xticks(rotation=0)
plt.ylabel('Frequency')
plt.show()
```



Sodium to Potassium Distribution based on Gender and Age 🧪👫👴

In [35]:
```
plt.scatter(x=df_drug.Age[df_drug.Sex=='F'], y=df_drug.Na_to_K[(df_
plt.scatter(x=df_drug.Age[df_drug.Sex=='M'], y=df_drug.Na_to_K[(df_
plt.legend(["Female", "Male"])
plt.xlabel("Age")
plt.ylabel("Na_to_K")
plt.show()
```



Dataset Preparation

This section will prepare the dataset before building the machine learning models.

Data Binning

Age

The age will be divided into **7 age categories**:

- Below 20 y.o.
- 20 - 29 y.o.
- 30 - 39 y.o.
- 40 - 49 y.o.
- 50 - 59 y.o.
- 60 - 69 y.o.
- Above 70.

In [36]:
```
bin_age = [0, 19, 29, 39, 49, 59, 69, 80]
category_age = ['<20s', '20s', '30s', '40s', '50s', '60s', '>60s']
df_drug['Age_binned'] = pd.cut(df_drug['Age'], bins=bin_age, labels
df_drug = df_drug.drop(['Age'], axis = 1)
```

In [37]:
```
bin_NatoK = [0, 9, 19, 29, 50]
category_NatoK = ['<10', '10-20', '20-30', '>30']
df_drug['Na_to_K_binned'] = pd.cut(df_drug['Na_to_K'], bins=bin_Nat
df_drug = df_drug.drop(['Na_to_K'], axis = 1)
```

Splitting the dataset

In [38]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [39]:
```
X = df_drug.drop(["Drug"], axis=1)
y = df_drug["Drug"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

Feature Engineering

In [40]:
```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
```

In [41]:
```
X_train.head()
```

Out[41]:

	Sex_F	Sex_M	BP_HIGH	BP_LOW	BP_NORMAL	Cholesterol_HIGH	Cholesterol_NORM
131	0	1	0	1	0	0	
96	1	0	0	1	0	1	
181	1	0	0	0	1	1	
19	1	0	1	0	0	0	
153	1	0	0	1	0	0	

In [42]:
```
X_test.head()
```

Out[42]:

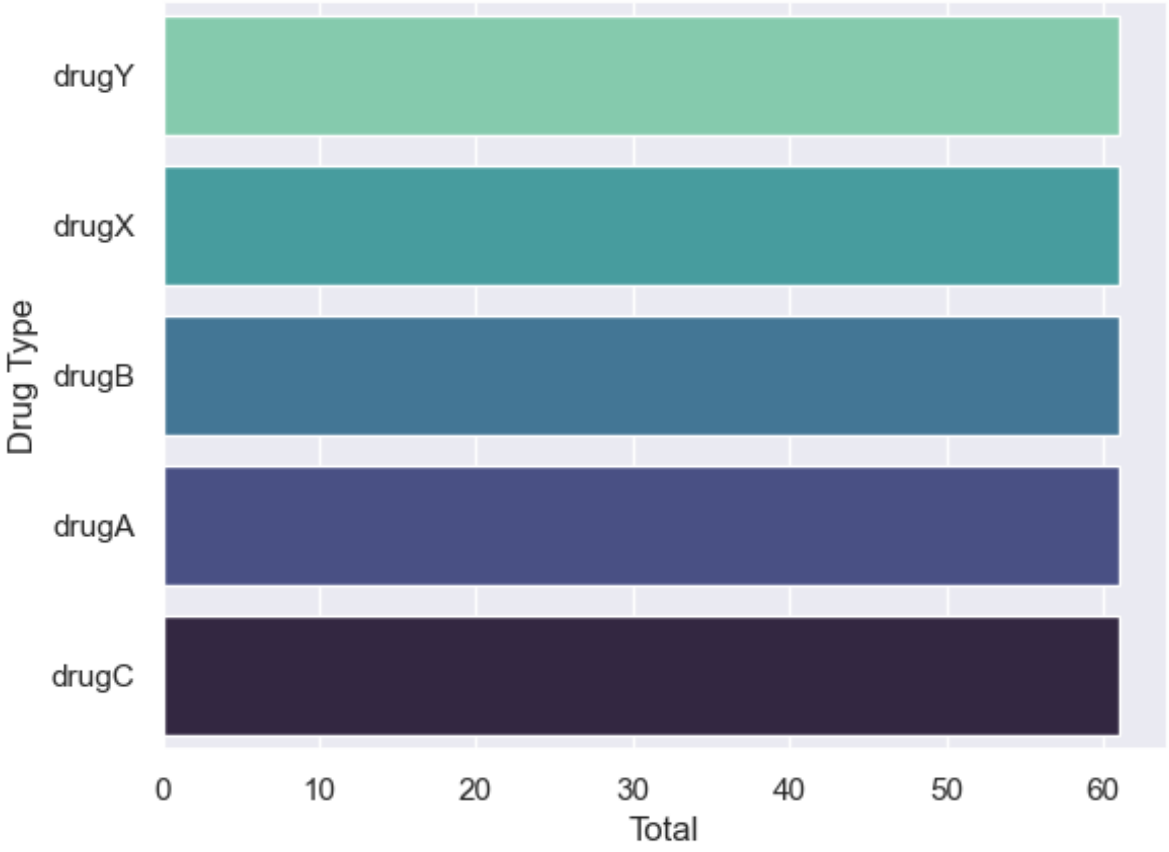
	Sex_F	Sex_M	BP_HIGH	BP_LOW	BP_NORMAL	Cholesterol_HIGH	Cholesterol_NORM
18	0	1	0	1	0	1	
170	1	0	0	0	1	1	
107	0	1	0	1	0	1	
98	0	1	1	0	0	0	
177	0	1	0	0	1	1	

SMOTE Technique ⚒

Since the number of 'DrugY' is more than other types of drugs, **oversampling is carried out to avoid overfitting**.

In [45]:
```
from imblearn.over_sampling import SMOTE
X_train, y_train = SMOTE().fit_resample(X_train, y_train)
```

In [46]:
```
sns.set_theme(style="darkgrid")
sns.countplot(y=y_train, data=df_drug, palette="mako_r")
plt.ylabel('Drug Type')
plt.xlabel('Total')
plt.show()
```



Models

Logistic Regression

In [47]:
```
from sklearn.linear_model import LogisticRegression
LRclassifier = LogisticRegression(solver='liblinear', max_iter=5000
LRclassifier.fit(X_train, y_train)

y_pred = LRclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
LRAcc = accuracy_score(y_pred,y_test)
print('Logistic Regression accuracy is: {:.2f}%'.format(LRAcc*100))
```

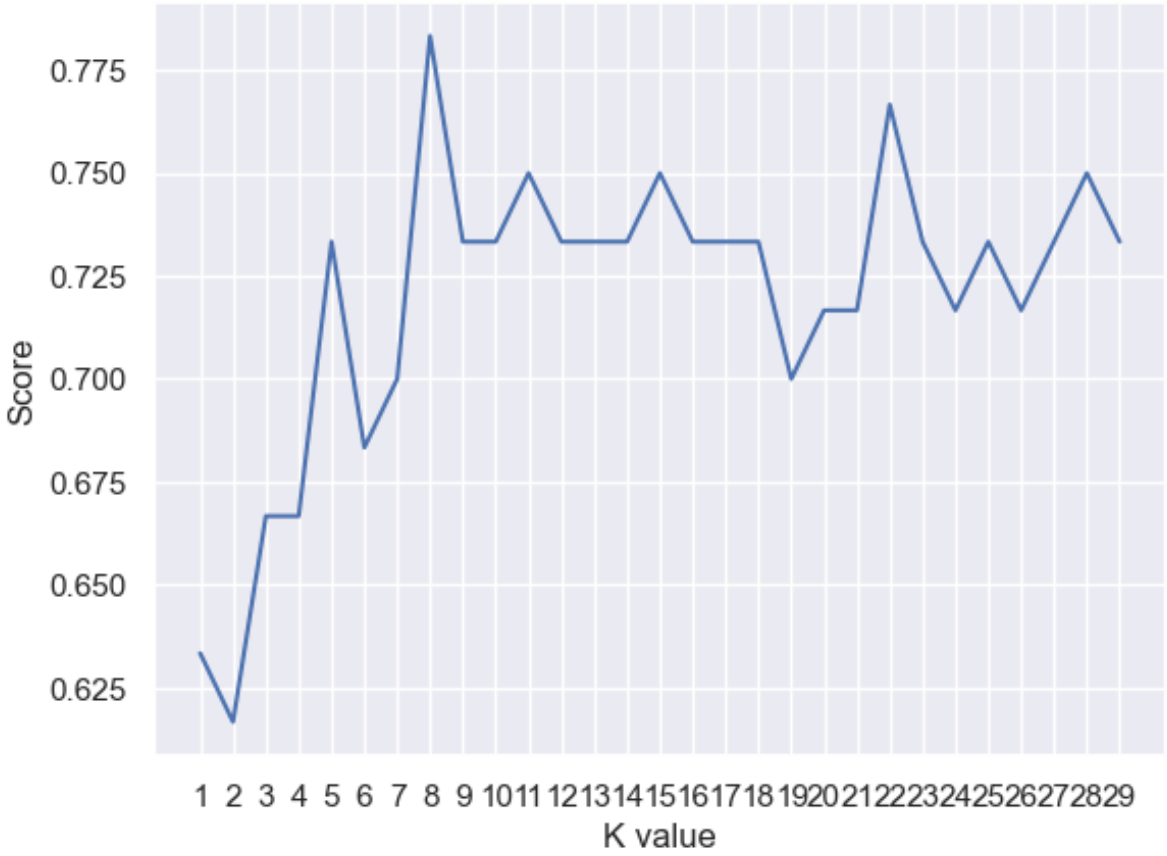
```
              precision    recall  f1-score   support

       drugA       0.71      1.00      0.83         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      0.78      0.80        18
       drugY       0.85      0.73      0.79        30

    accuracy                           0.80        60
   macro avg       0.76      0.90      0.82        60
weighted avg       0.81      0.80      0.80        60

[[ 5  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 14  4]
 [ 2  1  2  3 22]]
Logistic Regression accuracy is: 80.00%
```

K Neighbours

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
KNclassifier = KNeighborsClassifier(n_neighbors=20)
KNclassifier.fit(X_train, y_train)

y_pred = KNclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
KNAcc = accuracy_score(y_pred,y_test)
print('K Neighbours accuracy is: {:.2f}%'.format(KNAcc*100))
```

```
              precision    recall  f1-score   support

       drugA       0.40      0.80      0.53         5
       drugB       0.67      0.67      0.67         3
       drugC       0.57      1.00      0.73         4
       drugX       0.76      0.89      0.82        18
       drugY       0.89      0.57      0.69        30

    accuracy                           0.72        60
   macro avg       0.66      0.78      0.69        60
weighted avg       0.78      0.72      0.72        60

[[ 4  1  0  0  0]
 [ 1  2  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 16  2]
 [ 5  0  3  5 17]]
K Neighbours accuracy is: 71.67%
```

In [49]:
```
scoreListknn = []
for i in range(1,30):
    KNclassifier = KNeighborsClassifier(n_neighbors = i)
    KNclassifier.fit(X_train, y_train)
    scoreListknn.append(KNclassifier.score(X_test, y_test))

plt.plot(range(1,30), scoreListknn)
plt.xticks(np.arange(1,30,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
KNAccMax = max(scoreListknn)
print("KNN Acc Max {:.2f}%".format(KNAccMax*100))
```




```
KNN Acc Max 78.33%
```

Support Vector Machine (SVM)

In [50]:
```
from sklearn.svm import SVC
SVCclassifier = SVC(kernel='linear', max_iter=251)
SVCclassifier.fit(X_train, y_train)

y_pred = SVCclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
SVCAcc = accuracy_score(y_pred,y_test)
print('SVC accuracy is: {:.2f}%'.format(SVCAcc*100))
```

```
              precision    recall  f1-score   support

       drugA       0.71      1.00      0.83         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18
       drugY       1.00      0.70      0.82        30

    accuracy                           0.85        60
   macro avg       0.79      0.94      0.84        60
weighted avg       0.89      0.85      0.85        60

[[ 5  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 18  0]
 [ 2  1  2  4 21]]
SVC accuracy is: 85.00%

/Users/vikky/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm
/_base.py:299: ConvergenceWarning: Solver terminated early (max_it
er=251).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

Naive Bayes

Categorical NB

In [51]:
```
from sklearn.naive_bayes import CategoricalNB
NBclassifier1 = CategoricalNB()
NBclassifier1.fit(X_train, y_train)

y_pred = NBclassifier1.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
NBAcc1 = accuracy_score(y_pred,y_test)
print('Naive Bayes accuracy is: {:.2f}%'.format(NBAcc1*100))
```

```
              precision    recall  f1-score   support

       drugA       0.71      1.00      0.83         5
       drugB       0.75      1.00      0.86         3
       drugC       0.50      0.50      0.50         4
       drugX       0.70      0.78      0.74        18
       drugY       0.84      0.70      0.76        30

    accuracy                           0.75        60
   macro avg       0.70      0.80      0.74        60
weighted avg       0.76      0.75      0.75        60

[[ 5  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  2  2  0]
 [ 0  0  0 14  4]
 [ 2  1  2  4 21]]
Naive Bayes accuracy is: 75.00%
```

Gaussian NB

In [52]:
```
from sklearn.naive_bayes import GaussianNB
NBclassifier2 = GaussianNB()
NBclassifier2.fit(X_train, y_train)

y_pred = NBclassifier2.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
NBAcc2 = accuracy_score(y_pred,y_test)
print('Gaussian Naive Bayes accuracy is: {:.2f}%'.format(NBAcc2*100
```

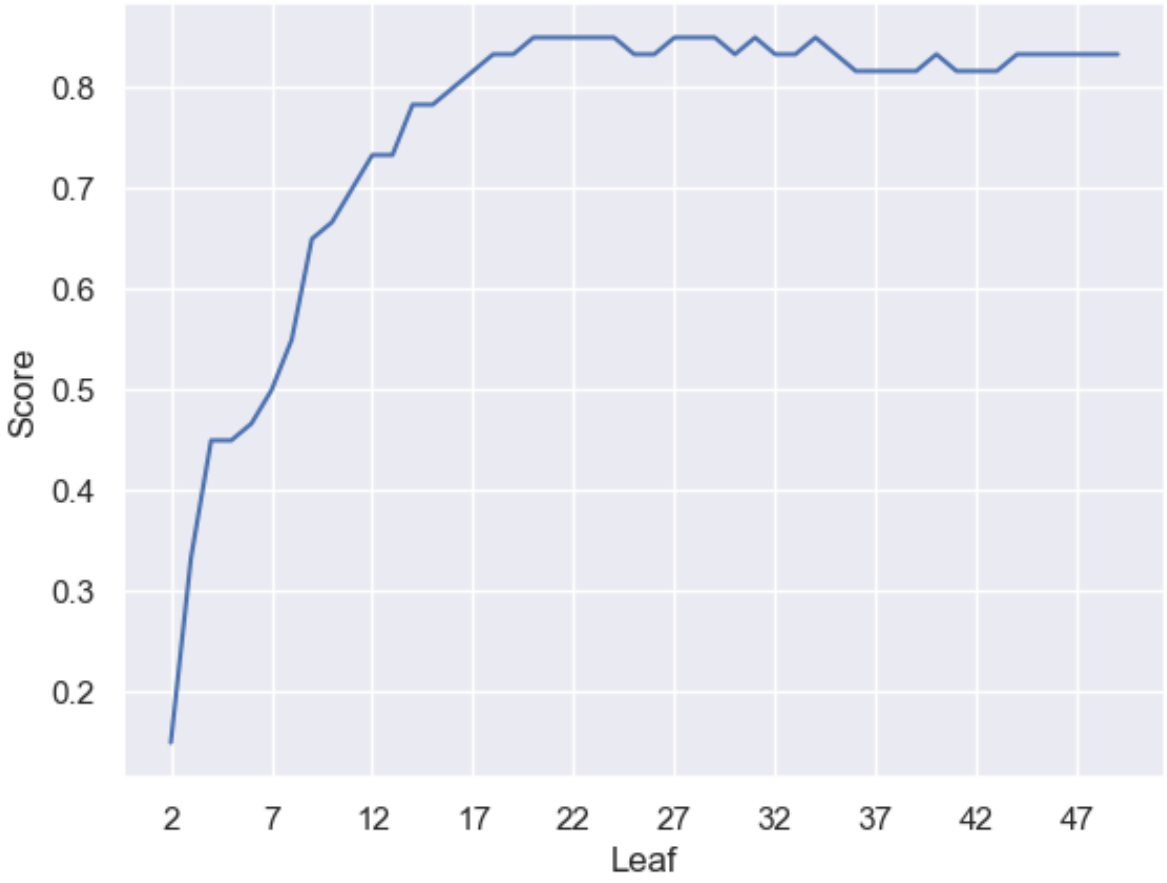
```
              precision    recall  f1-score   support

       drugA       0.60      0.60      0.60         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       1.00      0.39      0.56        18
       drugY       0.66      0.83      0.74        30

    accuracy                           0.70        60
   macro avg       0.73      0.76      0.71        60
weighted avg       0.76      0.70      0.68        60

[[ 3  0  0  0  2]
 [ 0  3  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0  7 11]
 [ 2  1  2  0 25]]
Gaussian Naive Bayes accuracy is: 70.00%
```

Decision Tree

In [53]:
```
from sklearn.tree import DecisionTreeClassifier
DTclassifier = DecisionTreeClassifier(max_leaf_nodes=20)
DTclassifier.fit(X_train, y_train)

y_pred = DTclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
DTAcc = accuracy_score(y_pred,y_test)
print('Decision Tree accuracy is: {:.2f}%'.format(DTAcc*100))
```

```
              precision    recall  f1-score   support

       drugA       0.71      1.00      0.83         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18
       drugY       1.00      0.70      0.82        30

    accuracy                           0.85        60
   macro avg       0.79      0.94      0.84        60
weighted avg       0.89      0.85      0.85        60

[[ 5  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 18  0]
 [ 2  1  2  4 21]]
Decision Tree accuracy is: 85.00%
```

In [54]:
```
scoreListDT = []
for i in range(2,50):
    DTclassifier = DecisionTreeClassifier(max_leaf_nodes=i)
    DTclassifier.fit(X_train, y_train)
    scoreListDT.append(DTclassifier.score(X_test, y_test))

plt.plot(range(2,50), scoreListDT)
plt.xticks(np.arange(2,50,5))
plt.xlabel("Leaf")
plt.ylabel("Score")
plt.show()
DTAccMax = max(scoreListDT)
print("DT Acc Max {:.2f}%".format(DTAccMax*100))
```




```
DT Acc Max 85.00%
```

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier

RFclassifier = RandomForestClassifier(max_leaf_nodes=30)
RFclassifier.fit(X_train, y_train)

y_pred = RFclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
RFAcc = accuracy_score(y_pred,y_test)
print('Random Forest accuracy is: {:.2f}%'.format(RFAcc*100))
```

```
              precision    recall  f1-score   support

       drugA       0.62      1.00      0.77         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18
       drugY       1.00      0.67      0.80        30

    accuracy                           0.83        60
   macro avg       0.77      0.93      0.83        60
weighted avg       0.88      0.83      0.83        60

[[ 5  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  4  0  0]
 [ 0  0  0 18  0]
 [ 3  1  2  4 20]]
Random Forest accuracy is: 83.33%
```

In [*]:
```
scoreListRF = []
for i in range(2,50):
    RFclassifier = RandomForestClassifier(n_estimators = 1000, rand
    RFclassifier.fit(X_train, y_train)
    scoreListRF.append(RFclassifier.score(X_test, y_test))

plt.plot(range(2,50), scoreListRF)
plt.xticks(np.arange(2,50,5))
plt.xlabel("RF Value")
plt.ylabel("Score")
plt.show()
RFAccMax = max(scoreListRF)
print("RF Acc Max {:.2f}%".format(RFAccMax*100))
```

Model Comparison

In [*]:
```
compare = pd.DataFrame({'Model': ['Logistic Regression', 'K Neighbo
                        'Accuracy': [LRAcc*100, KNAcc*100, KNAccMax
compare.sort_values(by='Accuracy', ascending=False)
```

Output

The next step will make output results in csv file.

Transforming prediction in appropriate output format

In [*]:
```
pred_lr = NBclassifier1.predict(X_test)
prediction = pd.DataFrame({'Sex_F': X_test.loc[:,"Sex_F"],
                           'Sex_M': X_test.loc[:,"Sex_M"],
                           'BP_HIGH': X_test.loc[:,"BP_HIGH"],
                           'BP_LOW': X_test.loc[:,"BP_LOW"],
                           'BP_NORMAL': X_test.loc[:,"BP_NORMAL"],
                           'Cholesterol_HIGH': X_test.loc[:,"Choles
                           'Cholesterol_NORMAL': X_test.loc[:,"Chol
                           'Age_binned_<20s': X_test.loc[:,"Age_bin
                           'Age_binned_20s': X_test.loc[:,"Age_binn
                           'Age_binned_30s': X_test.loc[:,"Age_binn
                           'Age_binned_40s': X_test.loc[:,"Age_binn
                           'Age_binned_50s': X_test.loc[:,"Age_binn
                           'Age_binned_60s': X_test.loc[:,"Age_binn
                           'Age_binned_>60s': X_test.loc[:,"Age_bin
                           'Na_to_K_binned_<10': X_test.loc[:,"Na_t
                           'Na_to_K_binned_10-20': X_test.loc[:,"Na
                           'Na_to_K_binned_20-30': X_test.loc[:,"Na
                           'Na_to_K_binned_>30': X_test.loc[:,"Na_t
```

In [*]:
```
 Sex
rediction['Sex_F'] = prediction['Sex_F'].replace([1, 0],['Female', '

BP
rediction['BP_HIGH'] = prediction['BP_HIGH'].replace([1, 0],['High',
rediction['BP_LOW'] = prediction['BP_LOW'].replace([1, 0],['Low', ''
rediction['BP_NORMAL'] = prediction['BP_NORMAL'].replace([1, 0],['No

rediction['BP_HIGH'] = np.where((prediction['BP_HIGH'] == ''), predi
rediction['BP_HIGH'] = np.where((prediction['BP_HIGH'] == ''), predi

Cholestrol
rediction['Cholesterol_HIGH'] = prediction['Cholesterol_HIGH'].repla

Age_binned
rediction['Age_binned_<20s'] = prediction['Age_binned_<20s'].replace
rediction['Age_binned_20s'] = prediction['Age_binned_20s'].replace([
rediction['Age_binned_30s'] = prediction['Age_binned_30s'].replace([
rediction['Age_binned_40s'] = prediction['Age_binned_40s'].replace([
rediction['Age_binned_50s'] = prediction['Age_binned_50s'].replace([
rediction['Age_binned_60s'] = prediction['Age_binned_60s'].replace([
rediction['Age_binned_>60s'] = prediction['Age_binned_>60s'].replace

rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s
rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s
rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s
rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s
rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s
rediction['Age_binned_<20s'] = np.where((prediction['Age_binned_<20s

Na to K
rediction['Na_to_K_binned_<10'] = prediction['Na_to_K_binned_<10'].r
rediction['Na_to_K_binned_10-20'] = prediction['Na_to_K_binned_10-20
rediction['Na_to_K_binned_20-30'] = prediction['Na_to_K_binned_20-30
rediction['Na_to_K_binned_>30'] = prediction['Na_to_K_binned_>30'].r

rediction['Na_to_K_binned_<10'] = np.where((prediction['Na_to_K_binn
rediction['Na_to_K_binned_<10'] = np.where((prediction['Na_to_K_binn
rediction['Na_to_K_binned_<10'] = np.where((prediction['Na_to_K_binn

 Drop columns
rediction = prediction.drop(['Sex_M', 'BP_LOW', 'BP_NORMAL', 'Choles
                'Age_binned_40s', 'Age_binned_50s', 'Age_binned_60s'
               'Na_to_K_binned_10-20', 'Na_to_K_binned_20-30', 'Na_t
```

In [*]:
```
# Rename columns name
new_name = {'Sex_F': 'Sex', 'BP_HIGH': 'BP', 'Cholesterol_HIGH': 'C
          'Na_to_K_binned_<10': 'Na_to_K_binned'}
prediction.rename(columns=new_name, inplace=True)
```

Generating output file (CSV)

In [*]:
```
prediction.to_csv('prediction.csv', index=False)
predictioncsv = pd.read_csv('./prediction.csv')
predictioncsv.head()
```

In [ ]:

In [ ]:

In [ ]: