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

In [2]:
```
d=pd.read_csv(r"C:\Users\user\Downloads\4_drug200 - 4_drug200.csv")
d
```

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]:
```
d.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]:
```
d.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 [5]:
```
d.describe()
```

Out[5]:

	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 [6]:
```
d.columns
```

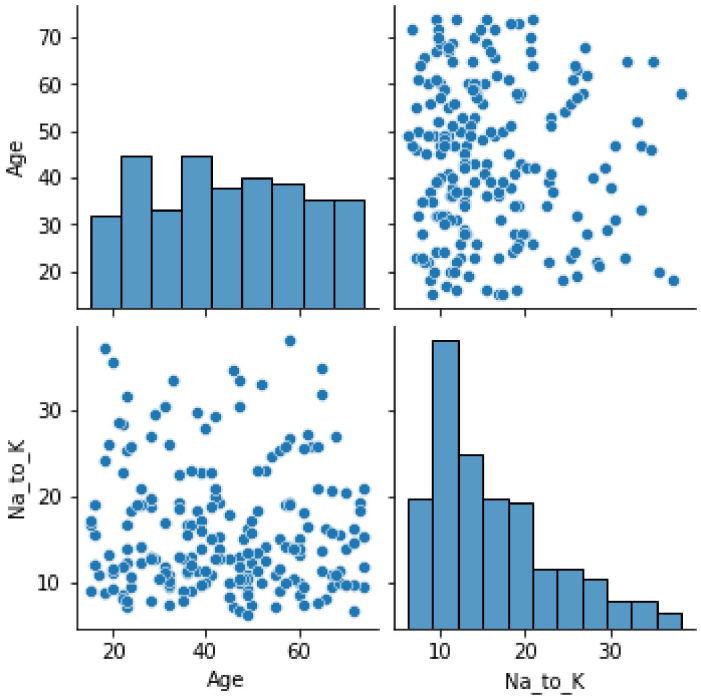
Out[6]:
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

In [7]:
```
d.index
```

Out[7]:
```
RangeIndex(start=0, stop=200, step=1)
```

In [8]:
```
sns.pairplot(d)
```

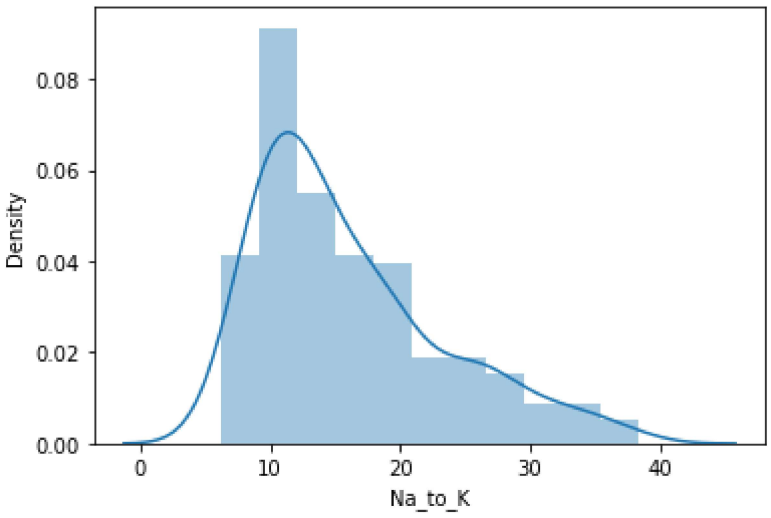
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x225e15692e0>
```



In [9]:
```
sns.distplot(d['Na_to_K'])
```

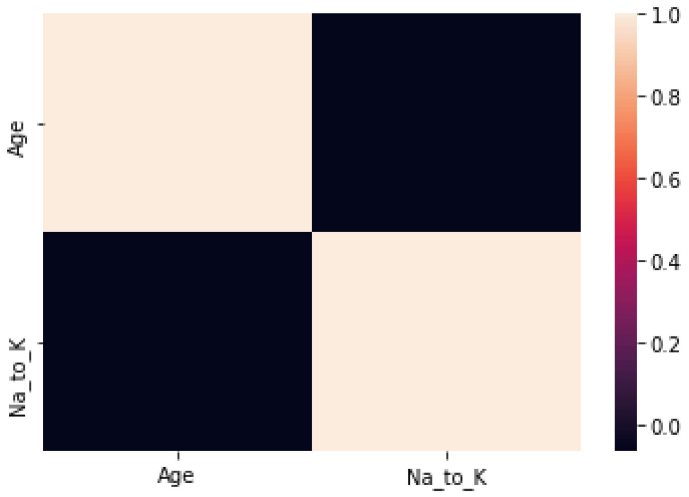
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]: <AxesSubplot:xlabel='Na_to_K', ylabel='Density'>



In [10]:
```
d1=d[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug']]
sns.heatmap(d1.corr())
```

Out[10]: <AxesSubplot:>



In [11]:
```
x=d1[['Age']]
y=d1['Na_to_K']
```

In [12]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [13]:
```
from sklearn.linear_model import LinearRegression
```

In [14]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
18.06313725901312
```

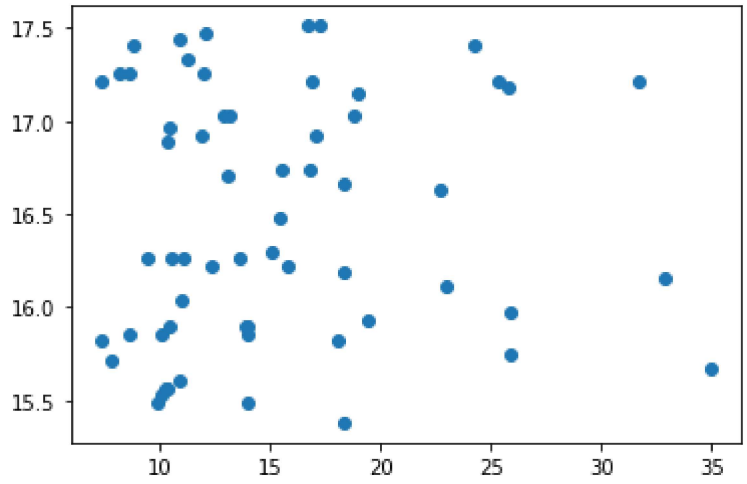
In [16]:
```
coeff =pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[16]:

	Co-efficient
Age	-0.036752

In [17]:
```
prediction =lr.predict(x_test)
py.scatter(y_test,prediction)
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x225e3829a00>
```



In [18]:
```
print(lr.score(x_test,y_test))
```

```
-0.034782243272657665
```

In [19]:
```
print(lr.score(x_train,y_train))
```

```
0.006025264809294328
```

In [20]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [21]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[21]:
```
Ridge(alpha=10)
```

In [22]:
```
rr.score(x_test,y_test)
```

Out[22]:
```
-0.03477771827031728
```

In [23]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[23]:
```
Lasso(alpha=10)
```

In [24]:
```
la.score(x_test,y_test)
```

Out[24]:
```
-0.02953410856989236
```

elasticnet

In [25]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[25]:
```
ElasticNet()
```

In [26]:
```
print(en.coef_)
```

```
[-0.03468343]
```

In [27]:
```
print(en.intercept_)
```

```
17.970456123681092
```

In [28]:
```
print(en.predict(x_test))
```

```
[16.89526971 15.43856553 16.86058627 17.2074206  15.75071643 15.54261583
 15.92413359 17.17273717 16.27096792 15.95881702 15.61198269 17.13805374
 17.17273717 16.27096792 16.65248568 17.4155212  15.78539986 17.17273717
 16.92995314 15.88945016 17.17273717 15.85476672 15.92413359 16.61780224
 15.85476672 16.23628448 15.71603299 17.34615433 15.64666613 16.99932001
 15.88945016 16.99932001 17.2074206  17.1033703  15.54261583 16.27096792
 16.20160105 16.16691762 16.72185254 16.89526971 17.2074206  15.61198269
 17.45020463 16.30565135 16.27096792 16.47906851 15.57729926 16.13223419
 16.68716911 16.06286732 15.88945016 15.99350045 15.92413359 16.23628448
 17.27678747 17.38083777 17.34615433 17.45020463 16.99932001 16.72185254]
```

In [29]:
```
print(en.score(x_test,y_test))
```

```
-0.03392364376621759
```

evaluation metrics

In [34]:
```
from sklearn import metrics
```

In [35]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 5.279829897166966
```

In [36]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 42.82162776866871
```

In [37]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction))
```

```
Root Mean Squared Error: 6.5438236352050865
```

In [ ]:
```

```