Problem statement

predicting the house price in USA.To create a model to help him estimate of what the house would sell for.

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

In [2]:
```
df=pd.read_csv("drug")
```

To display top 10 rows

In [3]:
```
df.head(10)
```

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
5	22	F	NORMAL	HIGH	8.607	drugX
6	49	F	NORMAL	HIGH	16.275	drugY
7	41	M	LOW	HIGH	11.037	drugC
8	60	M	NORMAL	HIGH	15.171	drugY
9	43	M	LOW	NORMAL	19.368	drugY

Data Cleaning And Pre-Processing

In [4]:
```
1 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
```

In [5]:
```
1 # Display the statistical summary
2 df.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]:
```
# To display the col headings
df.columns
```

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

In [7]:
```
cols=df.dropna(axis=1)
```

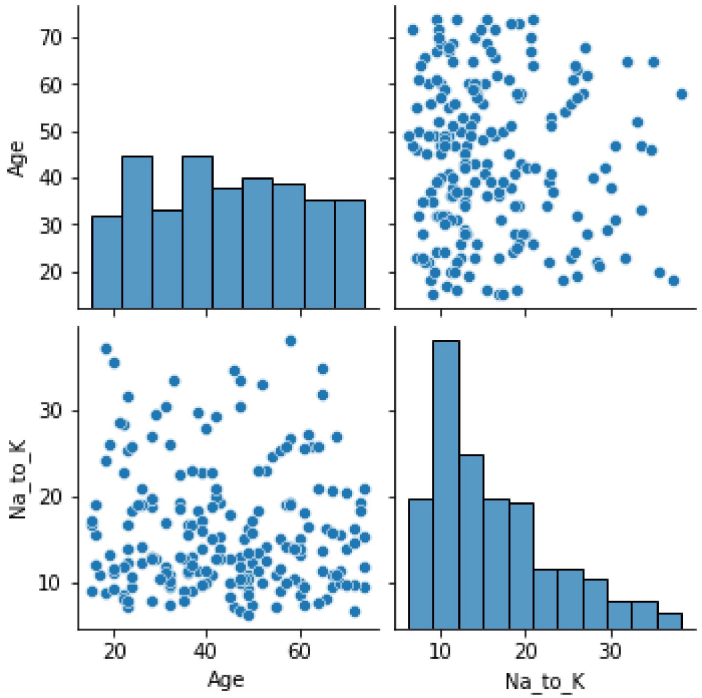
In [8]:
```
cols.columns
```

Out[8]:
```
Index(['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K', 'Drug'], dtype='object')
```

EDA and Visualization

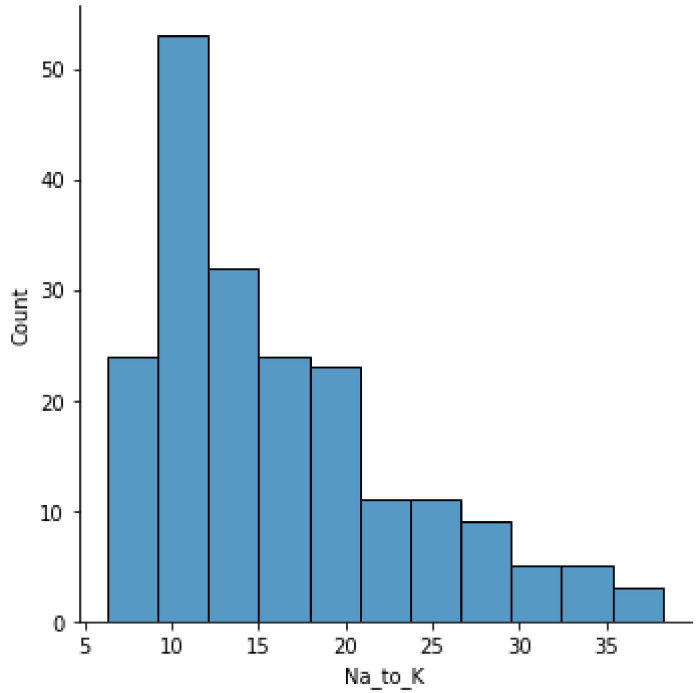
In [9]:
```
sns.pairplot(cols)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1f150e726d0>
```



In [10]:
```
1 sns.displot(df['Na_to_K'])
```

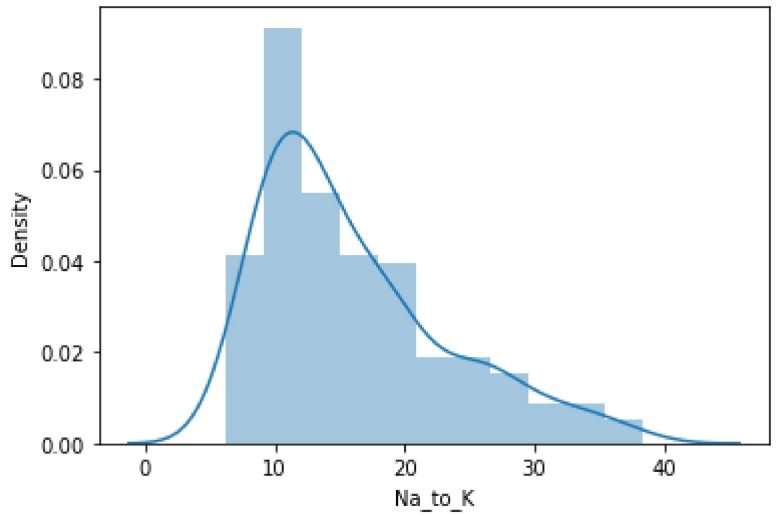
Out[10]: <seaborn.axisgrid.FacetGrid at 0x1f151768310>



In [11]:
```
# We use displot in older version we get distplot use displot
sns.distplot(df['Na_to_K'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a dep
recated 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)
```

Out[11]: <AxesSubplot:xlabel='Na_to_K', ylabel='Density'>



In [12]:
```
df1=cols[['Age', 'Na_to_K']]
df1
```

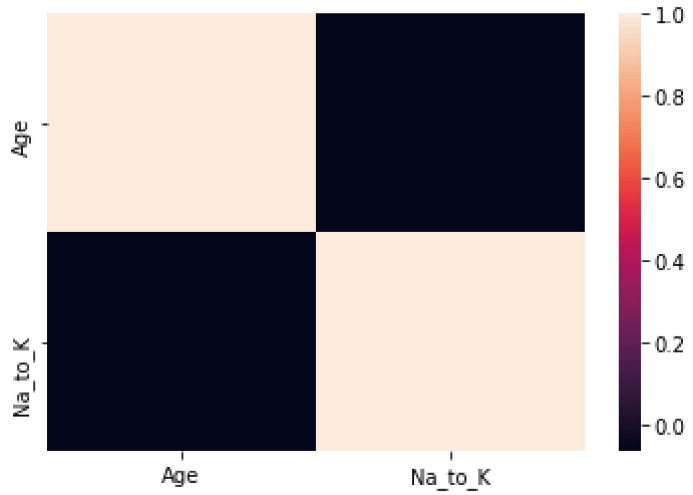
Out[12]:

	Age	Na_to_K
0	23	25.355
1	47	13.093
2	47	10.114
3	28	7.798
4	61	18.043
...	...	...
195	56	11.567
196	16	12.006
197	52	9.894
198	23	14.020
199	40	11.349

200 rows × 2 columns

In [13]:
```
1 sns.heatmap(df1.corr())
```

Out[13]: <AxesSubplot:>



To train the model - MODEL BUILD

Going to train linear regression model;We split our data into 2 variables x and y where x is independent var(input) and y is dependent on x(output), we could ignore address col as it is not required for our model

In [14]:
```
1 x=df1[['Age', 'Na_to_K']]
2 y=df1[['Na_to_K']]
```

To split the dataset into test data

In [15]:
```
1 # importing lib for splitting test data
2 from sklearn.model_selection import train_test_split
```

In [16]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

In [17]:
```
from sklearn.linear_model import LinearRegression

lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[17]:
```
LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
[7.10542736e-15]
```

In [19]:
```
print(lr.score(x_test,y_test))
```

```
1.0
```

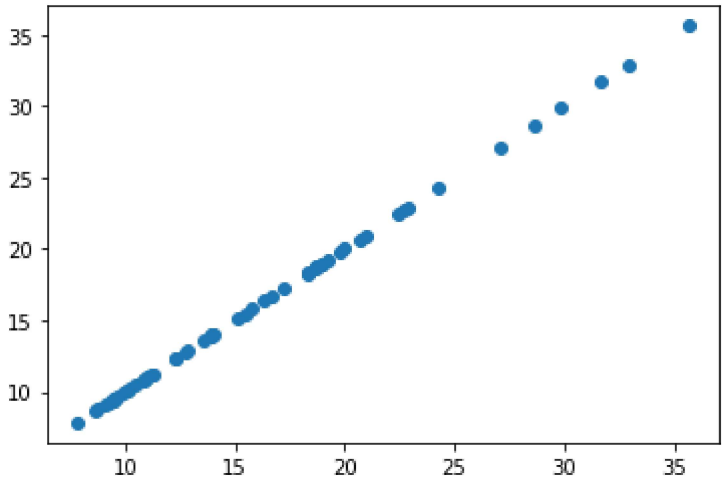
In [20]:
```
coeff=pd.DataFrame(lr.coef_)
coeff
```

Out[20]:

	0	1
0	-4.687824e-18	1.0

In [21]:
```
pred = lr.predict(x_test)
plt.scatter(y_test,pred)
```

Out[21]: <matplotlib.collections.PathCollection at 0x1f1531c2760>



In [22]:
```
from sklearn.linear_model import Ridge,Lasso
```

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

Out[23]: Ridge(alpha=10)

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

Out[24]: 0.9999983447574403

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]: Lasso(alpha=10)

In [26]:
```
la.score(x_test,y_test)
```

Out[26]: 0.9673495603384751

ELASTICNET

In [27]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[27]:
```
ElasticNet()
```

In [28]:
```
print(en.coef_)
```

```
[-0.          0.98209656]
```

In [29]:
```
print(en.intercept_)
```

```
[0.288742]
```

In [30]:
```
prediction=en.predict(x_test)
prediction
```

Out[30]:
```
array([11.10064299, 10.93466867, 18.25619851, 26.86820322, 13.92417059,
       35.28968119,  9.40358014, 18.69224938,  9.59410687,  9.68445976,
       18.30824963, 14.05773573, 31.4074535 ,  8.74164706,  9.56464398,
       14.00568461, 29.62887664, 16.34307441, 18.6568939 , 10.54477634,
       12.951895  , 17.20535519, 19.73032544, 10.99163027, 14.01059509,
       12.82618664,  9.56267978, 20.85580809, 12.36361917, 20.61126605,
       22.34270228, 22.78366364, 11.0692159 , 10.17550804, 18.93973771,
       10.05569226,  8.88208687,  7.99328949, 10.22166657, 18.95545126,
       16.71332482, 28.40813062, 15.79604663, 13.97425752, 18.76099614,
       24.13011802, 15.17339741, 10.9268119 , 13.64329098,  9.21010712,
       10.29532382, 11.34911342, 19.16561992, 11.28625924, 19.94344039,
       22.57938755,  9.77972312, 32.62132485, 12.37540432, 15.50141766])
```

In [31]:
```
print(en.score(x_test,y_test))
```

```
0.9996793149923281
```

EVALUATION METRICS

In [32]:
```
from sklearn import metrics
```

In [33]:
```
print("Mean Absolute Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolute Error: 0.09616337750064127
```

In [34]:
```
print("Mean Squared Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Squared Error: 0.014079799306715247
```

In [35]:
```
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
Root Mean Squared Error: 0.11865833011936097
```