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

In [28]:
```
var=pd.read_csv("C:/Users/Gopi/Desktop/machine learning/csv files/home.csv")
var
```

Out[28]:

	town	area	price
0	monre township	2600	550000
1	monre township	3000	565000
2	monre township	3200	610000
3	monre township	3600	680000
4	monre township	4000	725000
5	west windsor	2600	585000
6	west windsor	2800	615000
7	west windsor	3300	650000
8	west windsor	3600	710000
9	robbinsville	2600	575000
10	robbinsville	2900	600000
11	robbinsville	3100	620000
12	robbinsville	3600	695000

creating dummie variables

In [3]:
```
dummies=pd.get_dummies(var.town)
dummies
```

Out[3]:

	monre township	robbinsville	west windsor
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	0	0	1
6	0	0	1
7	0	0	1
8	0	0	1
9	0	1	0
10	0	1	0
11	0	1	0
12	0	1	0

In [4]:
```
varu=pd.concat([var,dummies],axis='columns')
varu
```

Out[4]:

	town	area	price	monre township	robbinsville	west windsor
0	monre township	2600	550000	1	0	0
1	monre township	3000	565000	1	0	0
2	monre township	3200	610000	1	0	0
3	monre township	3600	680000	1	0	0
4	monre township	4000	725000	1	0	0
5	west windsor	2600	585000	0	0	1
6	west windsor	2800	615000	0	0	1
7	west windsor	3300	650000	0	0	1
8	west windsor	3600	710000	0	0	1
9	robbinsville	2600	575000	0	1	0
10	robbinsville	2900	600000	0	1	0
11	robbinsville	3100	620000	0	1	0
12	robbinsville	3600	695000	0	1	0

In [5]:
```
varun=varu.drop(['town'],axis='columns')
varun
```

Out[5]:

	area	price	monre township	robbinsville	west windsor
0	2600	550000	1	0	0
1	3000	565000	1	0	0
2	3200	610000	1	0	0
3	3600	680000	1	0	0
4	4000	725000	1	0	0
5	2600	585000	0	0	1
6	2800	615000	0	0	1
7	3300	650000	0	0	1
8	3600	710000	0	0	1
9	2600	575000	0	1	0
10	2900	600000	0	1	0
11	3100	620000	0	1	0
12	3600	695000	0	1	0

In [6]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

In [8]:
```
X=varun.drop(['price'],axis='columns')
X
```

Out[8]:

	area	monre township	robbinsville	west windsor
0	2600	1	0	0
1	3000	1	0	0
2	3200	1	0	0
3	3600	1	0	0
4	4000	1	0	0
5	2600	0	0	1
6	2800	0	0	1
7	3300	0	0	1
8	3600	0	0	1
9	2600	0	1	0
10	2900	0	1	0
11	3100	0	1	0
12	3600	0	1	0

In [9]:
```
y=varun.price
y
```

Out[9]:
```
0     550000
1     565000
2     610000
3     680000
4     725000
5     585000
6     615000
7     650000
8     710000
9     575000
10    600000
11    620000
12    695000
Name: price, dtype: int64
```

In [10]:
```
model.fit(X,y)
```

Out[10]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [24]:
```
model.predict([[2600,0,0]])
```

Out[24]:
```
array([669344.35562038])
```

In [12]:
```
model.score(X,y)
```

Out[12]:
```
0.9573929037221873
```

using one hat encoding

In [13]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [14]:
```
var.town=le.fit_transform(var.town)
var
```

Out[14]:

	town	area	price
0	0	2600	550000
1	0	3000	565000
2	0	3200	610000
3	0	3600	680000
4	0	4000	725000
5	2	2600	585000
6	2	2800	615000
7	2	3300	650000
8	2	3600	710000
9	1	2600	575000
10	1	2900	600000
11	1	3100	620000
12	1	3600	695000

In [15]:
```
y=var.price
y
```

Out[15]:
```
0     550000
1     565000
2     610000
3     680000
4     725000
5     585000
6     615000
7     650000
8     710000
9     575000
10    600000
11    620000
12    695000
Name: price, dtype: int64
```

In [16]:
```
X=var[['town','area']].values
X
```

Out[16]:
```
array([[   0, 2600],
       [   0, 3000],
       [   0, 3200],
       [   0, 3600],
       [   0, 4000],
       [   2, 2600],
       [   2, 2800],
       [   2, 3300],
       [   2, 3600],
       [   1, 2600],
       [   1, 2900],
       [   1, 3100],
       [   1, 3600]], dtype=int64)
```

In [17]:
```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder(categorical_features=[0])
```

In [18]:
```
X=ohe.fit_transform(X).toarray()
X
```

```
C:\Users\Gopi\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:415: FutureWarning:
The handling of integer data will change in version 0.22. Currently, the categories are determin
ed based on the range [0, max(values)], while in the future they will be determined based on the
unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers,
then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\Users\Gopi\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:451: DeprecationWar
ning: The 'categorical_features' keyword is deprecated in version 0.20 and will be removed in 0.
22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

Out[18]:
```
array([[1.0e+00, 0.0e+00, 0.0e+00, 2.6e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.0e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.2e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.6e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 4.0e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 2.6e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 2.8e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 3.3e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 3.6e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 2.6e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 2.9e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 3.1e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 3.6e+03]])
```

In [19]:
```
X=X[:,1:]
X
```

Out[19]:
```
array([[0.0e+00, 0.0e+00, 2.6e+03],
       [0.0e+00, 0.0e+00, 3.0e+03],
       [0.0e+00, 0.0e+00, 3.2e+03],
       [0.0e+00, 0.0e+00, 3.6e+03],
       [0.0e+00, 0.0e+00, 4.0e+03],
       [0.0e+00, 1.0e+00, 2.6e+03],
       [0.0e+00, 1.0e+00, 2.8e+03],
       [0.0e+00, 1.0e+00, 3.3e+03],
       [0.0e+00, 1.0e+00, 3.6e+03],
       [1.0e+00, 0.0e+00, 2.6e+03],
       [1.0e+00, 0.0e+00, 2.9e+03],
       [1.0e+00, 0.0e+00, 3.1e+03],
       [1.0e+00, 0.0e+00, 3.6e+03]])
```

In [29]:
```
from sklearn.linear_model import LinearRegression
model1=LinearRegression()
```

In [21]:
```
model.fit(X,y)
```

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [22]:
```
model.predict([[1,0,2800]])
```

Out[22]:
```
array([590775.63964739])
```

In [23]:
```
model.score(X,y)
```

Out[23]:
```
0.9573929037221873
```