In [3]:

```
import pandas as pd

df = pd.read_csv("homeprice.csv")
df
```

Out[3]:

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

In [6]:

```
dummies = pd.get_dummies(df.town)
dummies
```

Out[6]:

	monroe township	robinsville	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 [9]:
```
merged = pd.concat([df,dummies], axis = "columns")
merged
```

Out[9]:

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

In [12]:
```
new = merged.drop("Unnamed: 3", axis = "columns")
new
```

Out[12]:

	town	area	price	monroe township	robinsville	west windsor
0	monroe township	2600	550000	1	0	0
1	monroe township	3000	565000	1	0	0
2	monroe township	3200	610000	1	0	0
3	monroe township	3600	680000	1	0	0
4	monroe 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	robinsville	2600	575000	0	1	0
10	robinsville	2900	600000	0	1	0
11	robinsville	3100	620000	0	1	0
12	robinsville	3600	695000	0	1	0

In [14]:
```
final = new.drop(["town","west windsor"], axis = "columns")
final
```

Out[14]:

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

In [15]:
```
from sklearn import linear_model
```

In [17]:
```
model = linear_model.LinearRegression()
```

In [19]:

```
x = final.drop('price', axis = 'columns')
x
```

Out[19]:

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

In [20]:

```
y = final.price
y
```

Out[20]:

```
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 [21]:

```
model.fit(x,y)
```

Out[21]:

```
LinearRegression()
```

In [23]:

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

Out[23]:

```
array([590775.63964739])
```

In [25]:

```
from sklearn.preprocessing import LabelEncoder
```

In [33]:

```
model.score(x,y)
```

Out[33]:

```
0.9573929037221873
```

In [26]:

```
le = LabelEncoder()
```

In [31]:

```
dfle = df
```

In [32]:

```
dfle.town = le.fit_transform(dfle.town)
dfle.town
```

Out[32]:

```
0     0
1     0
2     0
3     0
4     0
5     2
6     2
7     2
8     2
9     1
10    1
11    1
12    1
Name: town, dtype: int32
```

In [58]:

```
X = df[['town','area']].values
X
Y = dfle.price
```

In [65]:

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
ohe
```

Out[65]:

```
OneHotEncoder()
```

In [ ]:

```
```

In [54]:

```
X = ohe.fit_transform(x).toarray()
X
```

Out[54]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]])
```

In [66]:

```
X
```

Out[66]:

```
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 [56]:

In [59]:

```
model.fit(X,Y)
```

Out[59]:

```
LinearRegression()
```

In [69]:
```
model.predict([[ 0,1,2800]])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11788/2442174381.py in <module>
----> 1 model.predict([[ 0,1,2800]])

D:\ashwa\ana\lib\site-packages\sklearn\linear_model\_base.py in predict(self,
 X)
    236             Returns predicted values.
    237         """
--> 238         return self._decision_function(X)
    239 
    240     _preprocess_data = staticmethod(_preprocess_data)

D:\ashwa\ana\lib\site-packages\sklearn\linear_model\_base.py in _decision_funct
ion(self, X)
    219 
    220         X = check_array(X, accept_sparse=['csr', 'csc', 'coo'])
--> 221         return safe_sparse_dot(X, self.coef_.T,
    222                                dense_output=True) + self.intercept_
    223 

D:\ashwa\ana\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **
kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

D:\ashwa\ana\lib\site-packages\sklearn\utils\extmath.py in safe_sparse_dot(a,
 b, dense_output)
    150             ret = np.dot(a, b)
    151     else:
--> 152         ret = a @ b
    153 
    154     if (sparse.issparse(a) and sparse.issparse(b)

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, wit
h gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 3)
```

In [ ]:
```

```