To deal with Text data column you can either do Integer encoding/Label encoding where you convert the text data to as specific integer number, however, the problem with this approach is that it will assume the orders, for example if you convert apple to 1 and banana to 2 and kiwi to 3, it will assume :

that 2 is greater than 1 or banana is greater than apple or

apple + banana = kiwi

which ofcourse doesn't make any sense

These are called Categorical Variables

Categorical Variables is of two types:

1. Nominal Variables : where the categories doesn't have any numeric ordering between each other like male,female or apple,banana,kiwi or monroe town,robbinsville,west windsor

2. Ordinal Variables : where the categories have some sort of numerical ordering between them for example if you have graduate, masters and phd, so you could say that graduate is less than master and master is less than phd or another example is customer satisfaction survey such as satisfied, neutral and dissatisfied where satisfied is more than dissatisfied, etc

Our data here belongs to Nominal, which means its not a good idea to use the Integer encoding method,

Instead we could use a technique called One Hot Encoding

The way one hot encoding works is that you create a new column for each of your categories and assign binary values of 1 or 0, these extra variable column that we create is also called as dummy variables

## Importing Libraries and Loading the Data

In [2]:
```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib
import joblib
```

In [5]:
```
data = pd.read_csv('homeprices3.csv')
data
```

Out[5]:

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

## Preprocessing the Data

First we will create those dummy variable columns,

For this, we could use Pandas method get_dummies() which will return the dummy var columns,

if you have more than 3 towns like 10, then it would create 10 different columns

In [9]:
```
dummies = pd.get_dummies(data.town).astype(int)
dummies
```

Out[9]:

| | 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 |

Now, we will concatenate these dummy vairables column with our original data using Pandas concat

In [12]:
```
merged_data = pd.concat([data, dummies], axis=1)
merged_data
```

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 |

Now, we need to drop the town column since we dont need it anymore cuz we have the dummy variables column

Second thing is, we also need to drop one of the dummy variables column due to 'dummy variable trap'

Just search up what dummy variable trapm is and you will know, but the rule of a thumb is that we have to drop any 1 of the dummy var columns, say you have 5 dummy var columns, drop 1 and you will be left with 4

In our case we have 3 dummy var columns so we will drop 1, you can pick any column you want to drop

In [15]:
```
finaldata = merged_data.drop(['town', 'robinsville'], axis=1)
finaldata
```

Out[15]:

| | area | price | monroe township | west windsor |
|---|---|---|---|---|
| **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 | 1 |
| **6** | 2800 | 615000 | 0 | 1 |
| **7** | 3300 | 650000 | 0 | 1 |
| **8** | 3600 | 710000 | 0 | 1 |
| **9** | 2600 | 575000 | 0 | 0 |
| **10** | 2900 | 600000 | 0 | 0 |
| **11** | 3100 | 620000 | 0 | 0 |
| **12** | 3600 | 695000 | 0 | 0 |

Now, our data looks good and we have all numeric columns, also we have dropped the town and one of the dummy variable column.

Remember, when we use LinearRegression model, it will work even if you dont drop it because LinearRegression model is aware of the dummy variable trap and would drop it automatically but it is always good to drop it yourself

## Machine Learning part

In [16]:
```
model = LinearRegression()
```

In [17]:
```
X = finaldata.drop('price', axis=1)
y = finaldata.price
```

In [18]:
```
model.fit(X, y)
```

Out[18]:
```
▼ LinearRegression

LinearRegression()
```

Now, lets try predicting, for example if you want to check 2800 sqr feet home in monroe township you can type in [2800, 1, 0]

Similarly for west windsor type [2800, 0, 1]

and for Robinsville type [2800, 0, 0]

In [23]:
```
model.predict([[2800, 0, 0], [3400, 0, 1]])
```

```
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[23]:
```
array([590775.6396474 , 681241.66845839])
```

In [25]:
```
model.score(X, y)
```

Out[25]:
```
0.9573929037221872
```

# Sklearn's One hot encoder

Now, lets do the same thing but this time using SKlearn's One hot encoder as previously we used Pandas

First we need to do Label encoding on the town column

In [95]:
```
from sklearn.preprocessing import LabelEncoder
```

In [96]:
```
le = LabelEncoder()
dfle = data
```

fit_transform will take the town column as input and return the numeric label

but we dont want the array so just assign it back to the town column

Now, if you display dfle data, the town column will have the encoded label

In [97]:
```
dfle.town = le.fit_transform(dfle.town)
```

Out[97]:

| | 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 [98]:
```
X = dfle.drop(['price'], axis=1).values # using values cuz we want a 2D arra
y = data.price
```

Now, lets create the dummy variable, this time using SKLearn onehotencoder and Column transformer

In [99]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct = ColumnTransformer([("town", OneHotEncoder(), [0])], remainder='passthro
```

In [100…
```
X = ct.fit_transform(X)
X
```

Out[100]:
```
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 [101…
```
X = X[:, 1:]
X
```

Out[101]:
```
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 [102…
```
model.fit(X, y)
```

Out[102]:
```
▼ LinearRegression
LinearRegression()
```

In [105…
```
model.predict([[1, 0, 2800]])
```

Out[105]:
```
array([590775.63964739])
```