In [67]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import torch
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [68]:
```
housing = pd.read_csv('Housing.csv')
housing
```

Out[68]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheat
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	
...	...	...	...	...	...	...	...	...	
540	1820000	3000	2	1	1	yes	no	yes	
541	1767150	2400	3	1	1	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

545 rows × 13 columns

In [69]:
```
num_vars = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
df_new = housing[num_vars]
```

In [70]:
```
scaler = MinMaxScaler()
df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
df_new.head()
```

```
C:\Users\tareq\.conda\envs\ML-5105\lib\site-packages\pandas\core\frame.py:3678: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[col] = igetitem(value, i)
```

Out[70]:

	price	area	bedrooms	bathrooms	stories	parking
0	1.000000	0.396564	0.6	0.333333	0.666667	0.666667
1	0.909091	0.502405	0.6	1.000000	1.000000	1.000000
2	0.909091	0.571134	0.4	0.333333	0.333333	0.666667
3	0.906061	0.402062	0.6	0.333333	0.333333	1.000000
4	0.836364	0.396564	0.6	0.000000	0.333333	0.666667

In [71]:
```
X = df_new.iloc[:, 1:6].values
Y = df_new.iloc[:, 0].values
```

In [72]:
```
X = torch.tensor(X)
Y = torch.tensor(Y)
```

In [73]:
```
def model(X, W1, W2, W3, W4, W5, B):
    return W5*X[:,4] + W4*X[:,3] + W3*X[:,2] + W2*X[:,1] + W1*X[:,0] + B
```

In [74]:
```
def loss_fn(Y_p, Y):
    squared_diffs = (Y_p - Y)**2
    return squared_diffs.mean()
```

In [75]:
```
W1 = torch.ones(())
W2 = torch.ones(())
W3 = torch.ones(())
W4 = torch.ones(())
W5 = torch.ones(())
B = torch.zeros(())
```

In [76]:
```
n_samples = X.shape[0]
n_val = int(0.2 * n_samples)
shuffled_indices = torch.randperm(n_samples)
train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]
```

In [77]:
```
train_X = X[train_indices]
train_Y = Y[train_indices]
val_X = X[val_indices]
val_Y = Y[val_indices]
```

In [78]:
```
def training_loop(n_epochs, optimizer, params, train_X, val_X, train_Y, val_Y):

    for epoch in range(1, n_epochs + 1):
        train_Y_p = model(train_X, *params)
        train_loss = loss_fn(train_Y_p, train_Y)

        val_Y_p = model(val_X, *params)
        val_loss = loss_fn(val_Y_p, val_Y)

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        if epoch % 500 == 0:
```

```
            print(f"Epoch {epoch}, Training loss {train_loss.item():.4f},"
                  f" Validation loss {val_loss.item():.4f}")

    return params
```

In [79]:
```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.1
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.0118, Validation loss 0.0104
Epoch 1000, Training loss 0.0118, Validation loss 0.0104
Epoch 1500, Training loss 0.0118, Validation loss 0.0104
Epoch 2000, Training loss 0.0118, Validation loss 0.0104
Epoch 2500, Training loss 0.0118, Validation loss 0.0104
Epoch 3000, Training loss 0.0118, Validation loss 0.0104
Epoch 3500, Training loss 0.0118, Validation loss 0.0104
Epoch 4000, Training loss 0.0118, Validation loss 0.0104
Epoch 4500, Training loss 0.0118, Validation loss 0.0104
Epoch 5000, Training loss 0.0118, Validation loss 0.0104
```

Out[79]:
```
tensor([0.4426, 0.0927, 0.2979, 0.1320, 0.0960, 0.0339], requires_grad=True)
```

In [80]:
```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.01
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.0136, Validation loss 0.0144
Epoch 1000, Training loss 0.0120, Validation loss 0.0111
Epoch 1500, Training loss 0.0118, Validation loss 0.0105
Epoch 2000, Training loss 0.0118, Validation loss 0.0104
Epoch 2500, Training loss 0.0118, Validation loss 0.0104
Epoch 3000, Training loss 0.0118, Validation loss 0.0104
Epoch 3500, Training loss 0.0118, Validation loss 0.0104
Epoch 4000, Training loss 0.0118, Validation loss 0.0104
Epoch 4500, Training loss 0.0118, Validation loss 0.0104
Epoch 5000, Training loss 0.0118, Validation loss 0.0104
```

Out[80]:
```
tensor([0.4426, 0.0927, 0.2979, 0.1320, 0.0960, 0.0339], requires_grad=True)
```

In [81]:
```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.001
optimizer = optim.Adam([params], lr=learning_rate)
```

```
training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 0.1296, Validation loss 0.1134
Epoch 1000, Training loss 0.0519, Validation loss 0.0480
Epoch 1500, Training loss 0.0343, Validation loss 0.0332
Epoch 2000, Training loss 0.0230, Validation loss 0.0235
Epoch 2500, Training loss 0.0172, Validation loss 0.0184
Epoch 3000, Training loss 0.0148, Validation loss 0.0159
Epoch 3500, Training loss 0.0137, Validation loss 0.0146
Epoch 4000, Training loss 0.0130, Validation loss 0.0135
Epoch 4500, Training loss 0.0126, Validation loss 0.0125
Epoch 5000, Training loss 0.0122, Validation loss 0.0117
```

Out[81]:
```
tensor([ 0.4557,  0.2596,  0.2528,  0.1079,  0.0900, -0.0241],
       requires_grad=True)
```

In [82]:
```
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 0.0001
optimizer = optim.Adam([params], lr=learning_rate)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    params = params,
    train_X = train_X,
    val_X = val_X,
    train_Y = train_Y,
    val_Y = val_Y)
```

```
Epoch 500, Training loss 1.0156, Validation loss 0.9287
Epoch 1000, Training loss 0.8172, Validation loss 0.7443
Epoch 1500, Training loss 0.6499, Validation loss 0.5892
Epoch 2000, Training loss 0.5103, Validation loss 0.4601
Epoch 2500, Training loss 0.3951, Validation loss 0.3541
Epoch 3000, Training loss 0.3019, Validation loss 0.2687
Epoch 3500, Training loss 0.2283, Validation loss 0.2017
Epoch 4000, Training loss 0.1719, Validation loss 0.1509
Epoch 4500, Training loss 0.1305, Validation loss 0.1141
Epoch 5000, Training loss 0.1016, Validation loss 0.0888
```

Out[82]:
```
tensor([ 0.6147,  0.6150,  0.5916,  0.5953,  0.5937, -0.3704],
       requires_grad=True)
```