In [1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import torch
#torch.set_printoptions(edgeitems=2, linewidth=75)
```

In [2]:
```
t_c = [0.5,  14.0, 15.0, 28.0, 11.0,  8.0,  3.0, -4.0,  6.0, 13.0, 21.0]
t_u = [35.7, 55.9, 58.2, 81.9, 56.3, 48.9, 33.9, 21.8, 48.4, 60.4, 68.4]
t_c = torch.tensor(t_c)
t_u = torch.tensor(t_u)
```

In [3]:
```
def model(t_u, w1, w2, b):
    return w2*t_u**2 + w1*t_u + b
```

In [4]:
```
def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()
```

In [5]:
```
n_samples = t_u.shape[0]
n_val = int(0.2 * n_samples)
```

In [6]:
```
shuffled_indices = torch.randperm(n_samples)
```

In [7]:
```
train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]
print(train_indices, val_indices)
```

```
tensor([ 4,  5,  2,  6,  8, 10,  9,  0,  7]) tensor([1, 3])
```

In [8]:
```
train_t_u = t_u[train_indices]
train_t_c = t_c[train_indices]
```

In [9]:
```
val_t_u = t_u[val_indices]
val_t_c = t_c[val_indices]
```

In [10]:
```
train_t_un = 0.1 * train_t_u
val_t_un = 0.1 * val_t_u
```

In [11]:
```
train_loss_list = []
val_loss_list = []
#epoch_list = []
```

In [12]:
```
def training_loop(n_epochs, optimizer, params, train_t_u, train_t_c, val_t_u, val_t_c):
    for epoch in range(1, n_epochs + 1):
        train_t_p = model(train_t_u, *params)
        train_loss = loss_fn(train_t_p, train_t_c)

        val_t_p = model(val_t_u, *params)
        val_loss = loss_fn(val_t_p, val_t_c)
        val_loss_list.append(val_loss.item())

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()
        train_loss_list.append(train_loss.item())
```

```
        if epoch <= 1 or epoch % 500 ==0:
            #epoch_list.append(epoch)
            print(f"Epoch {epoch}, Training loss {train_loss.item():.4f}, "f"Validation

    return params
```

In [13]:
```
import torch.optim as optim

dir(optim)
```

Out[13]:
```
['ASGD',
 'Adadelta',
 'Adagrad',
 'Adam',
 'AdamW',
 'Adamax',
 'LBFGS',
 'NAdam',
 'Optimizer',
 'RAdam',
 'RMSprop',
 'Rprop',
 'SGD',
 'SparseAdam',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_functional',
 '_multi_tensor',
 'lr_scheduler',
 'swa_utils']
```

In [14]:
```
t_un = 0.1 * t_u
```

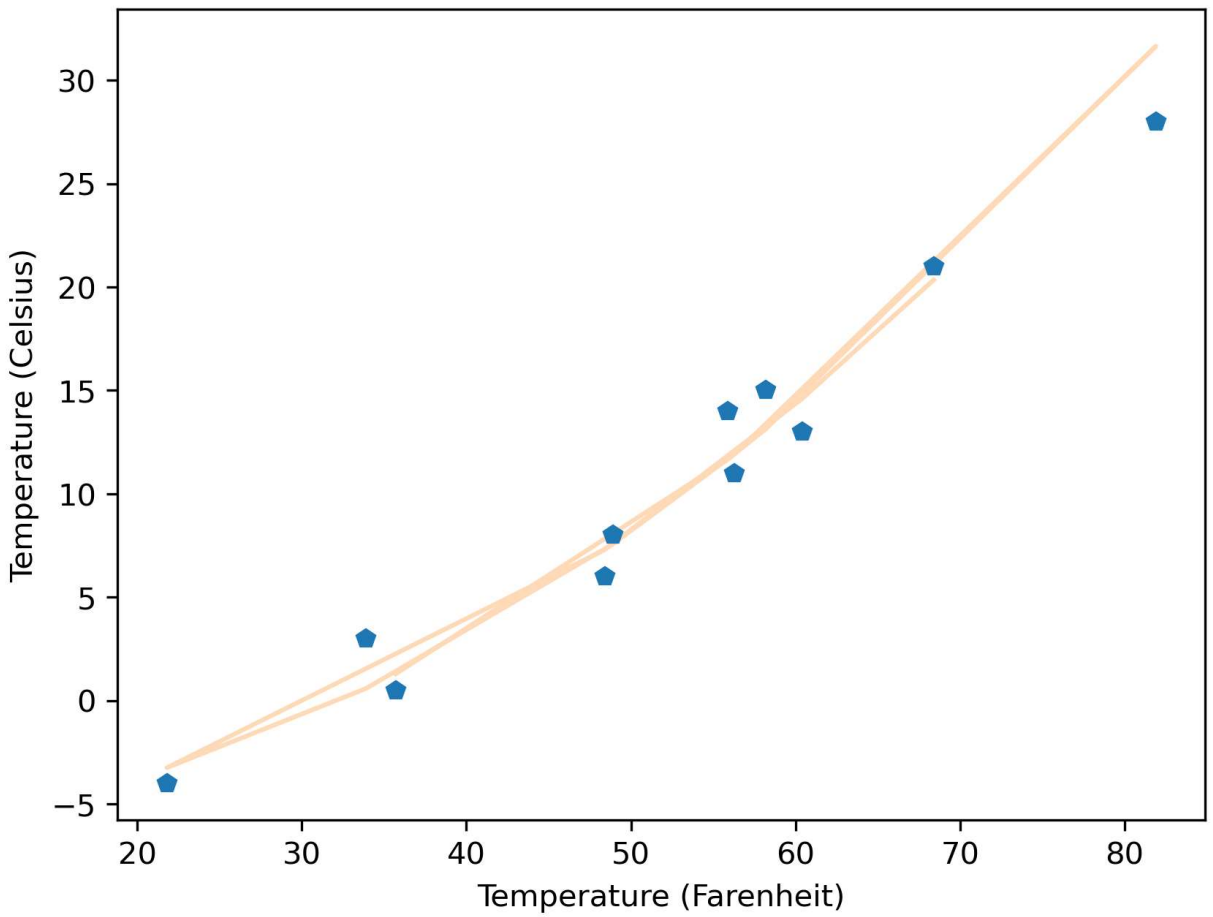
In [15]:
```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.1
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 1.8331, Validation loss 13.6555
Epoch 1000, Training loss 1.8231, Validation loss 12.9343
Epoch 1500, Training loss 1.8133, Validation loss 12.0619
Epoch 2000, Training loss 1.8059, Validation loss 11.2070
Epoch 2500, Training loss 1.8015, Validation loss 10.4798
Epoch 3000, Training loss 1.7996, Validation loss 9.9441
Epoch 3500, Training loss 1.7989, Validation loss 9.6099
Epoch 4000, Training loss 1.7988, Validation loss 9.4400
Epoch 4500, Training loss 1.7988, Validation loss 9.3736
Epoch 5000, Training loss 1.7988, Validation loss 9.3551
```

Out[15]:
```
tensor([ 0.1012,  0.5501, -6.0823], requires_grad=True)
```

In [16]:
```
from matplotlib import pyplot as plt
#t_un = 0.1 * t_u
t_p = model(t_un, *params)
fig = plt.figure(dpi=300)
plt.xlabel('Temperature (Farenheit)')
plt.ylabel('Temperature (Celsius)')
plt.plot(t_u.numpy(), t_p.detach().numpy(), color='peachpuff',)
plt.plot(t_u.numpy(), t_c.numpy(), 'p')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x1d98b128a00>]
```



In [17]:
```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.01
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
```

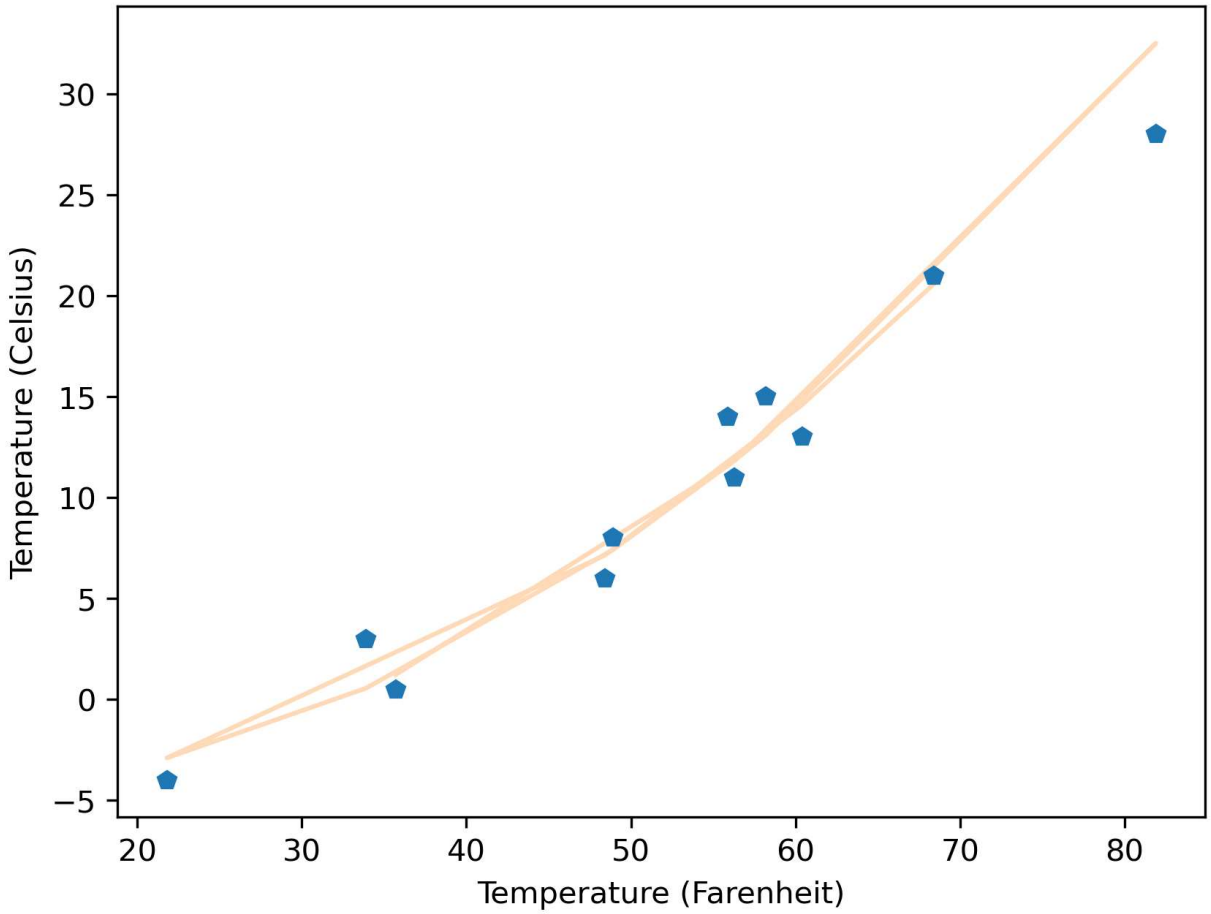
```
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 6.4291, Validation loss 4.5225
Epoch 1000, Training loss 3.5718, Validation loss 3.1108
Epoch 1500, Training loss 2.2457, Validation loss 6.7859
Epoch 2000, Training loss 1.8943, Validation loss 10.9993
Epoch 2500, Training loss 1.8424, Validation loss 13.2007
Epoch 3000, Training loss 1.8371, Validation loss 13.7966
Epoch 3500, Training loss 1.8348, Validation loss 13.7756
Epoch 4000, Training loss 1.8320, Validation loss 13.5993
Epoch 4500, Training loss 1.8289, Validation loss 13.3682
Epoch 5000, Training loss 1.8252, Validation loss 13.0897
```

Out[17]:
```
tensor([-0.6664,  0.6324, -4.4482], requires_grad=True)
```

In [18]:
```
from matplotlib import pyplot as plt
#t_un = 0.1 * t_u
t_p = model(t_un, *params)
fig = plt.figure(dpi=300)
plt.xlabel('Temperature (Farenheit)')
plt.ylabel('Temperature (Celsius)')
plt.plot(t_u.numpy(), t_p.detach().numpy(), color='peachpuff',)
plt.plot(t_u.numpy(), t_c.numpy(), 'p')
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x1d99226b160>]
```



In [19]:
```
params = torch.tensor([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_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 87.5958, Validation loss 162.9495
Epoch 1000, Training loss 13.4346, Validation loss 0.9132
Epoch 1500, Training loss 8.7090, Validation loss 5.7738
Epoch 2000, Training loss 8.2820, Validation loss 7.3488
Epoch 2500, Training loss 7.8330, Validation loss 6.6663
Epoch 3000, Training loss 7.2956, Validation loss 5.7663
Epoch 3500, Training loss 6.6758, Validation loss 4.8303
Epoch 4000, Training loss 5.9894, Validation loss 3.9517
Epoch 4500, Training loss 5.2624, Validation loss 3.2530
Epoch 5000, Training loss 4.5309, Validation loss 2.8763
```

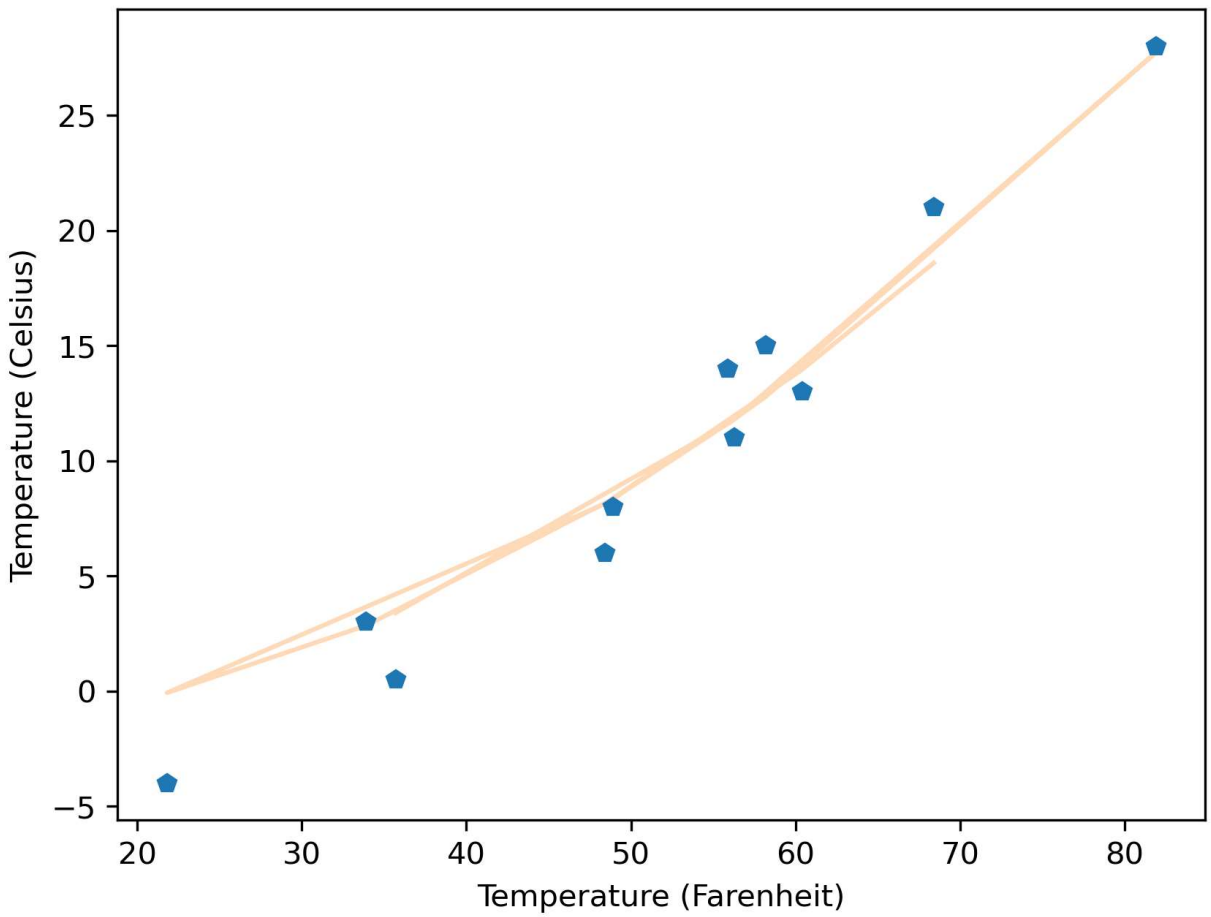
Out[19]:
```
tensor([-0.1546,  0.4611, -1.9273], requires_grad=True)
```

In [20]:
```
from matplotlib import pyplot as plt
#t_un = 0.1 * t_u
t_p = model(t_un, *params)
```

```
fig = plt.figure(dpi=300)
plt.xlabel('Temperature (Farenheit)')
plt.ylabel('Temperature (Celsius)')
plt.plot(t_u.numpy(), t_p.detach().numpy(), color='peachpuff',)
plt.plot(t_u.numpy(), t_c.numpy(), 'p')
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x1d992365130>]
```



In [21]:

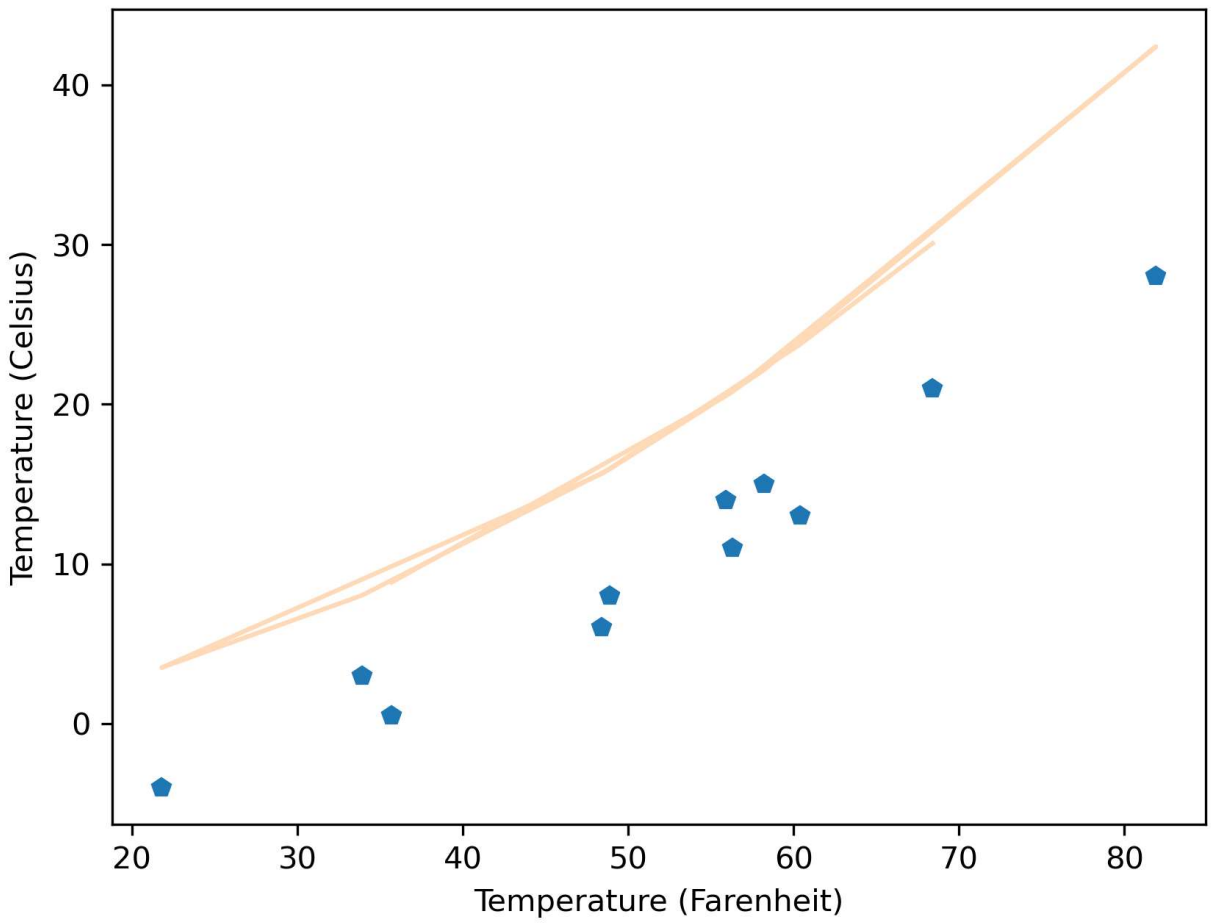
```
params = torch.tensor([1.0,1.0,0.0], requires_grad=True)
learning_rate = 0.0001
optimizer = optim.Adam([params], lr = learning_rate)

training_loop(
    n_epochs=5000,
    optimizer = optimizer,
    params = params,
    train_t_u = train_t_un,
    train_t_c = train_t_c,
    val_t_u = val_t_un,
    val_t_c = val_t_c)
```

```
Epoch 1, Training loss 519.7859, Validation loss 1377.8323
Epoch 500, Training loss 447.2988, Validation loss 1167.2705
Epoch 1000, Training loss 382.3910, Validation loss 980.0179
Epoch 1500, Training loss 324.4459, Validation loss 814.1627
Epoch 2000, Training loss 272.8312, Validation loss 667.7673
Epoch 2500, Training loss 227.0313, Validation loss 539.2454
Epoch 3000, Training loss 186.6240, Validation loss 427.2912
Epoch 3500, Training loss 151.2583, Validation loss 330.8061
Epoch 4000, Training loss 120.6321, Validation loss 248.8293
Epoch 4500, Training loss 94.4727, Validation loss 180.4769
Epoch 5000, Training loss 72.5197, Validation loss 124.8842
```

Out[21]:
```
tensor([ 0.5668,  0.5693, -0.4369], requires_grad=True)
```

In [22]:
```
from matplotlib import pyplot as plt
#t_un = 0.1 * t_u
t_p = model(t_un, *params)
fig = plt.figure(dpi=300)
plt.xlabel('Temperature (Farenheit)')
plt.ylabel('Temperature (Celsius)')
plt.plot(t_u.numpy(), t_p.detach().numpy(), color='peachpuff',)
plt.plot(t_u.numpy(), t_c.numpy(), 'p')
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x1d992c91310>]
```



In [ ]:

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)
```

In [1]:
```
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import torch
import time
import torch.optim as optim
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [2]:
```
housing = pd.read_csv('Housing.csv')
housing
```

Out[2]:

	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 [3]:
```
num_vars = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']
df_new = housing[num_vars]
df_new
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2
...	...	...	...	...	...	...
540	1820000	3000	2	1	1	2
541	1767150	2400	3	1	1	0
542	1750000	3620	2	1	1	0
543	1750000	2910	3	1	1	0
544	1750000	3850	3	1	2	0

545 rows × 6 columns

In [4]:
```
df_new.shape
```

Out[4]:
```
(545, 6)
```

In [5]:
```
scaler = MinMaxScaler()
df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
df_new.head()
```

```
C:\Users\shenc\AppData\Local\Temp\ipykernel_46180\2943118898.py:2: SettingWithCopyWarni
ng:
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
  df_new[num_vars] = scaler.fit_transform(df_new[num_vars])
```

Out[5]:

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

In [7]:
```
X = torch.tensor(X)
Y = torch.tensor(Y)
```

In [8]:
```
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 [9]:
```
train_X = X[train_indices]
train_Y = Y[train_indices]
train_Y = train_Y.unsqueeze(1)

val_X = X[val_indices]
val_Y = Y[val_indices]
val_Y = val_Y.unsqueeze(1)
```

In [10]:
```
import torch
import torch.nn as nn
import torch.optim as optim

model = nn.Sequential(
            nn.Linear(5,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1

optimizer = optim.SGD(model.parameters(), lr=learning_rate)

out = model(train_X.float())


loss_fn = nn.MSELoss()

n_epochs = 200
correct = 0
total = 0

for epoch in range(n_epochs+1):
        training_start_time = time.time()
        out = model(train_X.float())
        loss = loss_fn(out,train_Y.float())

        out_v= model(val_X.float()) # <1>
        val_loss = loss_fn(out_v, val_Y.float())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch==0 or epoch % 20 ==0:
            print("Epoch: %d, Training Loss: %f, Validation Loss: %f" % (epoch, float(l
        _, predicted = torch.max(out, dim=1)
        total += val_Y.shape[0]
        correct += int((predicted == train_Y).sum())

print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```

In [11]:

```
import torch
import torch.nn as nn
import torch.optim as optim

model2 = nn.Sequential(
            nn.Linear(5,16),
            nn.ReLU(),
            nn.Linear(16,8),
            nn.ReLU(),
            nn.Linear(8,1),
            nn.ReLU(),
            nn.LogSoftmax(dim=1))
model = model.float()
learning_rate = 0.1

optimizer = optim.SGD(model.parameters(), lr=learning_rate)

out = model2(train_X.float())


loss_fn = nn.MSELoss()

n_epochs = 200
correct = 0
total = 0

for epoch in range(n_epochs+1):
        training_start_time = time.time()
        out = model2(train_X.float())
        loss = loss_fn(out,train_Y.float())

        out_v= model2(val_X.float()) # <1>
        val_loss = loss_fn(out_v, val_Y.float())

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch==0 or epoch % 20 ==0:
            print("Epoch: %d, Training Loss: %f, Validation Loss: %f" % (epoch, float(l
        _, predicted = torch.max(out, dim=1)
        total += val_Y.shape[0]
        correct += int((predicted == train_Y).sum())

print("Accuracy: %f" % (correct / total))
print('Time {:.2f}s'.format(time.time() - training_start_time))
```

```
Epoch: 0, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 20, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 40, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 60, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 80, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 100, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 120, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 140, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 160, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 180, Training Loss: 0.094215, Validation Loss: 0.095123
Epoch: 200, Training Loss: 0.094215, Validation Loss: 0.095123
Accuracy: 8.000000
Time 0.00s
```