In [44]:
```
#PART 1
from PIL import Image
green = Image.open("./greenapples.jpg")
red = Image.open("./redApple.jpg")
blue = Image.open("./bluewhale.jpg")
```

In [45]:
```
import torch

from torchvision import transforms

transformer = transforms.ToTensor()

green_t = transformer(green)
blue_t = transformer(blue)
red_t = transformer(red)
```

In [ ]:
```

```

In [46]:
```
print("Green average brightness is: ")
print(green_t.mean())
print("Blue average brightness is: ")
print(blue_t.mean())
print("Red average brightness is: ")
print(red_t.mean())
```

```
Green average brightness is:
tensor(0.4459)
Blue average brightness is:
tensor(0.4289)
Red average brightness is:
tensor(0.3086)
```

In [47]:
```
print(green_t[0].mean())
print(green_t[1].mean())
print(green_t[2].mean())
```

```
tensor(0.4645)
tensor(0.6375)
tensor(0.2357)
```

In [48]:
```
print(blue_t[0].mean())
print(blue_t[1].mean())
print(blue_t[2].mean())
```

```
tensor(0.1249)
tensor(0.4690)
tensor(0.6929)
```

In [49]:
```
print(red_t[0].mean())
print(red_t[1].mean())
print(red_t[2].mean())
```

```
tensor(0.5616)
tensor(0.1776)
tensor(0.1867)
```

In [50]:
```
import torch
```

```
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)
t_un = 0.1 * t_u
```

In [51]:

```
#PART 2

def model(t_u, w1, w2, b):
    return w2 * t_u ** 2 + w1 * t_u + b

def loss_fn(t_p, t_c):
    return ((t_p-t_c)**2).mean()

def dloss_fn(t_p, t_c):
    return 2 * (t_p - t_c)/t_p.size(0)
#I dont underwstand how we are deriving
def dmodel_dw1(t_u, w1, w2, b):
    return t_u
def dmodel_dw2(t_u, w1, w2, b):
    return t_u**2
def dmodel_db(t_u, w1, w2, b):
    return 1.0

def grad_fn(t_u, t_c, t_p, w1, w2, b):
    dloss_dtp = dloss_fn(t_p, t_c)
    dloss_dw1 = dloss_dtp * dmodel_dw1(t_u, w1, w2, b)
    dloss_dw2 = dloss_dtp * dmodel_dw2(t_u, w1, w2, b)
    dloss_db = dloss_dtp * dmodel_db(t_u, w1, w2, b)
    return torch.stack([dloss_dw1.sum(), dloss_dw2.sum(), dloss_db.sum()])

def training_loop(n_epochs, learning_rate, params, t_u, t_c):
    for epoch in range(1, n_epochs+1):
        w1, w2, b = params

        t_p = model(t_u, w1, w2, b)
        loss = loss_fn(t_p, t_c)
        grad = grad_fn(t_u, t_c, t_p, w1, w2, b)

        params = params - learning_rate * grad

        if epoch % 500 == 0 or epoch < 2:
            print('Epoch %d, Loss %f' % (epoch, float(loss)))
    return params
```

In [52]:

```
training_loop(
n_epochs = 5000,
learning_rate = 1e-1,
params = torch.tensor([1.0, 1.0, 0.0]),
t_u = t_un,
t_c = t_c)
training_loop(
n_epochs = 5000,
learning_rate = 1e-2,
params = torch.tensor([1.0, 1.0, 0.0]),
t_u = t_un,
t_c = t_c)
training_loop(
n_epochs = 5000,
learning_rate = 1e-3,
params = torch.tensor([1.0, 1.0, 0.0]),
t_u = t_un,
t_c = t_c)
```

```
params = training_loop(
n_epochs = 5000,
learning_rate = 1e-4,
params = torch.tensor([1.0, 1.0, 0.0]),
t_u = t_un,
t_c = t_c)
```

```
Epoch 1, Loss 675.794373
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan
Epoch 1, Loss 675.794373
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan
Epoch 1, Loss 675.794373
Epoch 500, Loss nan
Epoch 1000, Loss nan
Epoch 1500, Loss nan
Epoch 2000, Loss nan
Epoch 2500, Loss nan
Epoch 3000, Loss nan
Epoch 3500, Loss nan
Epoch 4000, Loss nan
Epoch 4500, Loss nan
Epoch 5000, Loss nan
Epoch 1, Loss 675.794373
Epoch 500, Loss 10.708597
Epoch 1000, Loss 8.642083
Epoch 1500, Loss 7.171005
Epoch 2000, Loss 6.123476
Epoch 2500, Loss 5.377228
Epoch 3000, Loss 4.845286
Epoch 3500, Loss 4.465787
Epoch 4000, Loss 4.194724
Epoch 4500, Loss 4.000802
Epoch 5000, Loss 3.861744
```

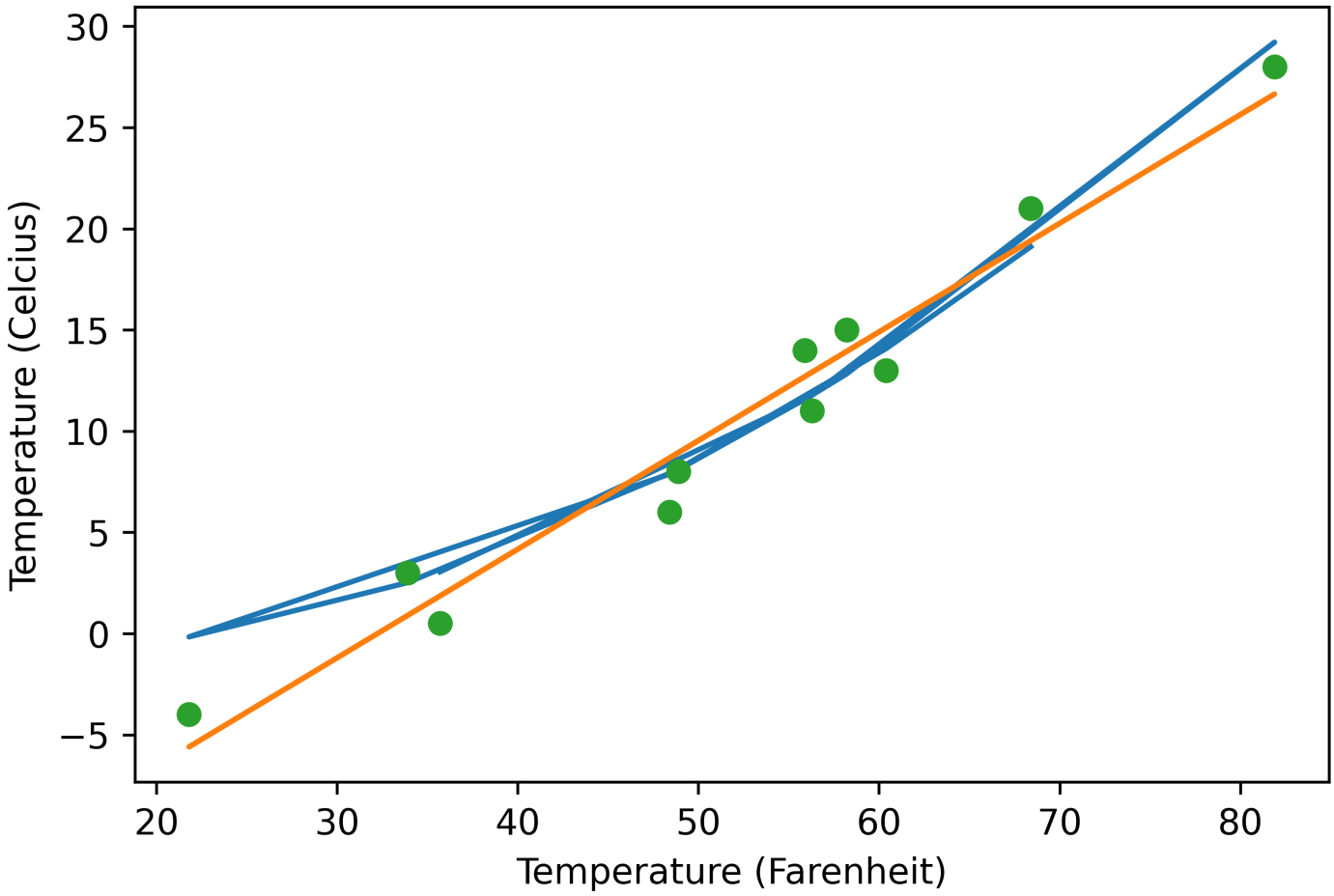
In [56]:

```
%matplotlib inline
from matplotlib import pyplot as plt




t_pp = model(t_un, *params)
t_pl = 5.3671 * t_un -17.3012
fig = plt.figure(dpi=600)
plt.xlabel("Temperature (Farenheit)")
plt.ylabel("Temperature (Celcius)")
plt.plot(t_u.numpy(), t_pp.detach().numpy())
```

```
plt.plot(t_u.numpy(), t_pl.detach().numpy())
plt.plot(t_u.numpy(), t_c.numpy(), 'o')
```

Out[56]:

```
[<matplotlib.lines.Line2D at 0x2708dba6ee0>]
```



In [57]:

```
#PART 3


import numpy as np
import pandas as pd

# Data Visualisation
import matplotlib.pyplot as plt

df = pd.DataFrame(pd.read_csv("Housing.csv"))
df.replace(('yes', 'no'), (1, 0), inplace=True)
df.replace(('furnished', 'semi-furnished', 'unfurnished'), (1, .5, 0), inplace=True)
housing_N=(df-df.min())/(df.max()-df.min())
housing_N = housing_N[['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking']]
#print(housing_N.head())

t_un = housing_N.drop('price', axis=1)
print(type(t_un))
t_c = housing_N['price']
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [58]:

```
def loss_fn(t_p, t_c):
    return ((t_p-t_c)**2).mean()

def model(tu_1, tu_2, tu_3, tu_4, tu_5, w1, w2, w3, w4, w5, b):
    return w1*tu_1+w2*tu_2+w3*tu_3+w4*tu_4+w5*tu_5 + b
```

```
def training_loop(n_epochs, optimizer, params, t_u, t_c):
    for epoch in range(1, n_epochs+1):
            t_p = model(torch.from_numpy(t_u['area'].to_numpy()), torch.from_numpy(t_u['be
                        torch.from_numpy(t_u['bathrooms'].to_numpy()), torch.from_numpy(t_
            loss = loss_fn(t_p, torch.tensor(t_c))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if epoch % 500 == 0:
                print('Epoch %d, Loss %f' % (epoch, float(loss)))
    return params
```

In [60]:
```
import torch
import torch.optim as optim
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
learning_rate = 1e-1

optimizer = optim.SGD([params], lr=learning_rate)
optimizer.zero_grad()
training_loop(n_epochs=5000, optimizer=optimizer, params = params, t_u = t_un, t_c = t_c)
```

```
Epoch 500, Loss 0.011730
Epoch 1000, Loss 0.011490
Epoch 1500, Loss 0.011477
Epoch 2000, Loss 0.011477
Epoch 2500, Loss 0.011477
Epoch 3000, Loss 0.011477
Epoch 3500, Loss 0.011477
Epoch 4000, Loss 0.011477
Epoch 4500, Loss 0.011477
Epoch 5000, Loss 0.011477
```

Out[60]:
```
tensor([0.4171, 0.0726, 0.2945, 0.1423, 0.0981, 0.0433], requires_grad=True)
```

In [61]:
```
import torch.optim as optim
learning_rate = 1e-2

params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
optimizer.zero_grad()
optimizer1 = optim.SGD([params], lr=learning_rate)
training_loop(n_epochs=5000, optimizer=optimizer1, params = params, t_u = t_un, t_c = t_c)
```

```
Epoch 500, Loss 0.049834
Epoch 1000, Loss 0.022525
Epoch 1500, Loss 0.015945
Epoch 2000, Loss 0.013894
Epoch 2500, Loss 0.013000
Epoch 3000, Loss 0.012499
Epoch 3500, Loss 0.012183
Epoch 4000, Loss 0.011972
Epoch 4500, Loss 0.011829
Epoch 5000, Loss 0.011730
```

Out[61]:
```
tensor([0.4581, 0.1827, 0.2842, 0.1164, 0.0832, 0.0009], requires_grad=True)
```

In [62]:
```
import torch.optim as optim
learning_rate = 1e-3
optimizer.zero_grad()
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
optimizer2 = optim.SGD([params], lr=learning_rate)
training_loop(n_epochs=5000, optimizer=optimizer2, params = params, t_u = t_un, t_c = t_c)
```

```
Epoch 500, Loss 0.222480
Epoch 1000, Loss 0.138441
Epoch 1500, Loss 0.117175
Epoch 2000, Loss 0.102285
Epoch 2500, Loss 0.089779
Epoch 3000, Loss 0.079105
Epoch 3500, Loss 0.069981
Epoch 4000, Loss 0.062177
Epoch 4500, Loss 0.055497
Epoch 5000, Loss 0.049776
```

Out[62]:
```
tensor([ 0.6986,  0.5646,  0.6522,  0.3475,  0.3725, -0.3941],
       requires_grad=True)
```

In [63]:
```
import torch.optim as optim
learning_rate = 1e-4
optimizer.zero_grad()
params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
optimizer3 = optim.SGD([params], lr=learning_rate)
training_loop(n_epochs=5000, optimizer=optimizer3, params = params, t_u = t_un, t_c = t_c)
```

```
Epoch 500, Loss 0.975083
Epoch 1000, Loss 0.781201
Epoch 1500, Loss 0.633147
Epoch 2000, Loss 0.519966
Epoch 2500, Loss 0.433324
Epoch 3000, Loss 0.366881
Epoch 3500, Loss 0.315813
Epoch 4000, Loss 0.276449
Epoch 4500, Loss 0.245998
Epoch 5000, Loss 0.222335
```

Out[63]:
```
tensor([ 0.8470,  0.7589,  0.9057,  0.7747,  0.7979, -0.5066],
       requires_grad=True)
```

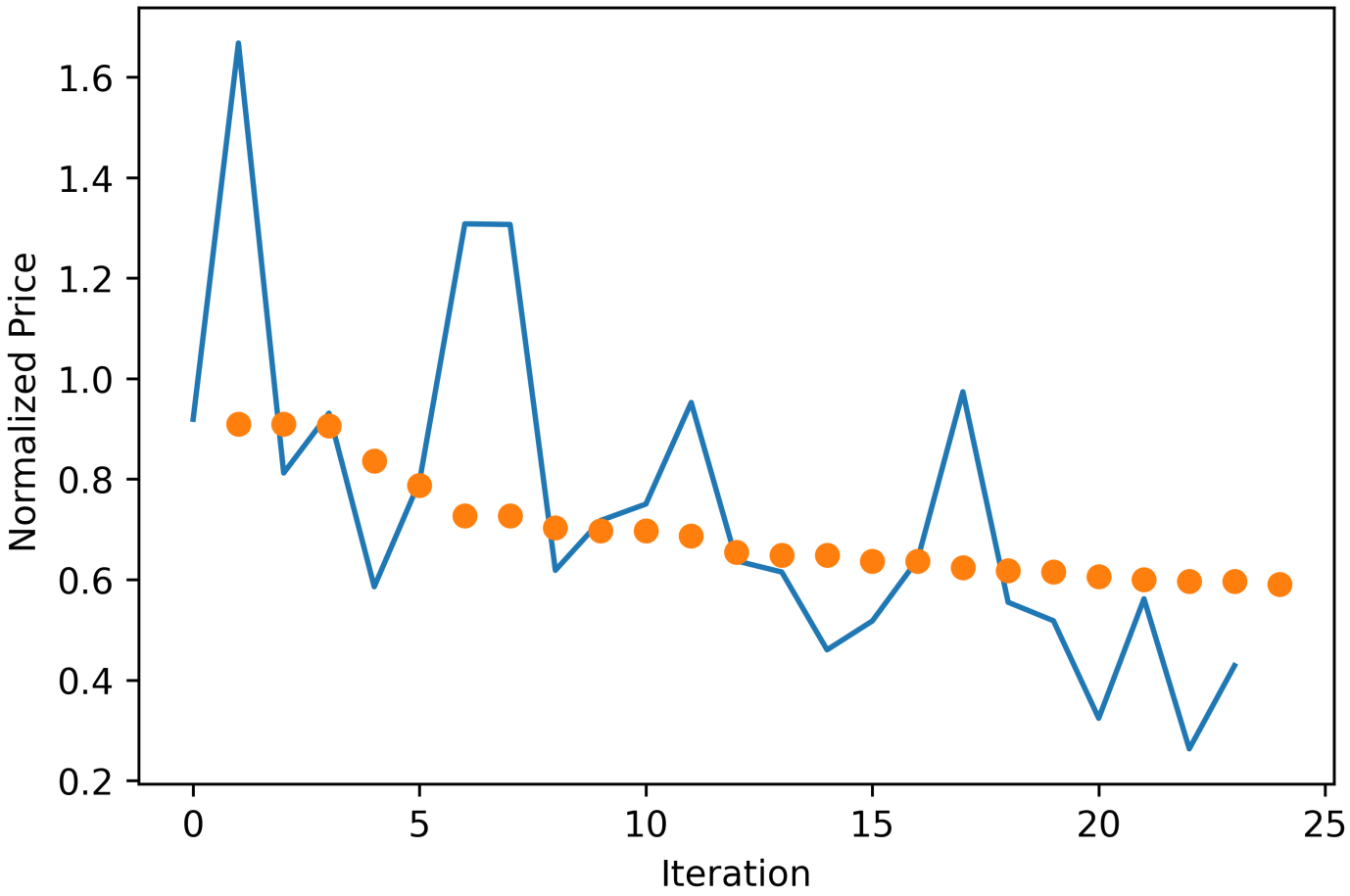
In [64]:
```
import torch
import torch.optim as optim
learning_rate = 1e-3

params = torch.tensor([1.0, 1.0, 1.0, 1.0, 1.0, 0.0], requires_grad=True)
optimizer2 = optim.SGD([params], lr=learning_rate)
optimizer2.zero_grad()
weights = training_loop(n_epochs=5000, optimizer=optimizer2, params = params, t_u = t_un,
```

```
Epoch 500, Loss 0.222480
Epoch 1000, Loss 0.138441
Epoch 1500, Loss 0.117175
Epoch 2000, Loss 0.102285
Epoch 2500, Loss 0.089779
Epoch 3000, Loss 0.079105
Epoch 3500, Loss 0.069981
Epoch 4000, Loss 0.062177
Epoch 4500, Loss 0.055497
Epoch 5000, Loss 0.049776
```

In [65]:
```
from matplotlib import pyplot as plt

t_ppp = [5]
t_u = t_un
with torch.no_grad():
    for i in range(0, 50):
        t_ppp.append(model(t_u['area'].to_numpy()[i], t_u['bedrooms'].to_numpy()[i],
                           t_u['bathrooms'].to_numpy()[i], t_u['stories'].to_numpy()[i], t_u

fig = plt.figure(dpi=600)
```

```
plt.xlabel("Iteration")
plt.ylabel("Normalized Price ")
plt.plot(t_ppp[1:25])
plt.plot(t_c[1:25], 'o')
```

Out[65]: [<matplotlib.lines.Line2D at 0x2708dc1ac70>]



In [ ]:

In [ ]: