```python
"""
Example 2.1 Linear Regression
@author: Austin Downey
"""

import IPython as IP
IP.get_ipython().run_line_magic('reset', '-sf')

import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
from sklearn import datasets
from sklearn.linear_model import LinearRegression

plt.close('all')

#%% load data

ames = sk.datasets.fetch_openml(name="house_prices", as_frame=True,parser='auto')
target = ames['target'].values
data = ames['data']
YrSold = data['YrSold'].values  # Year Sold (YYYY)
MoSold = data['MoSold'].values  # Month Sold (MM)
OverallCond = data['OverallCond'].values # OverallCond: Rates the overall condition of the house
GrLivArea = data['GrLivArea'].values # Above grade (ground) living area square feet
BedroomAbvGr = data['BedroomAbvGr'].values # Bedrooms above grade (does NOT include basement bedrooms)

# Ask a home buyer to describe their dream house, and they probably won't begin
# with the height of the basement ceiling or the proximity to an east-west railroad.
# But this playground competition's dataset proves that much more influences price
# negotiations than the number of bedrooms or a white-picket fence.

# With 79 explanatory variables describing (almost) every aspect of residential
# homes in Ames, Iowa, this competition challenges you to predict the final price
# of each home.

# Plot a few of the interesting features vs the target (price). In particular,
# let's plot the number of rooms vs. the price.
plt.figure()
plt.plot(GrLivArea,target,'o',markersize=2)
plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('price (USD)')
plt.grid(True)
plt.tight_layout()

#%% Build a model for the data
X = GrLivArea
Y = target
model_X = np.linspace(0,5000)

theta_1 = 0
theta_2 = 100
model_Y_manual = theta_1 + theta_2*model_X

plt.figure()
plt.plot(X,Y,'o',markersize=2,label='data')
plt.plot(model_X,model_Y_manual,'--',label='manual fit')
plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('price (USD)')
plt.grid(True)
#plt.xlim([3.5,9])
#plt.ylim([0,50000])
plt.legend(framealpha=1)
plt.tight_layout()
```

```
# add a dimension to the data as math is easier in 2d arrays and sk learn only
# takes 2d arrays
X = np.expand_dims(X,axis=1)
Y = np.expand_dims(Y,axis=1)
model_X = np.expand_dims(model_X,axis=1)

#%% compute the linear regression solution using the closed form solution

# compute
X_b = np.ones((X.shape[0],2))
X_b[:,1] = X.T # add x0 = 1 to each instance

theta_closed_form = np.linalg.inv(X_b.T@X_b)@X_b.T@Y

model_y_closed_form = theta_closed_form[0] + theta_closed_form[1]*3000
model_Y_closed_form = theta_closed_form[0] + theta_closed_form[1]*model_X

plt.figure()
plt.plot(X,Y,'o',markersize=3,label='data points')
plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('price (USD)')
plt.plot(3000,model_y_closed_form,'dr',markersize=10,zorder=10,
         label='inferred data point')
plt.plot(model_X,model_Y_closed_form,'-',label='normal equation')
plt.grid(True)
plt.legend()
plt.tight_layout()

#%% compute the linear regression solution using gradient descent

eta = 0.00000001 # learning rate
n_iterations = 100
m = X.shape[0]
theta_gradient_descent = np.random.randn(2,1) # random initialization
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta_gradient_descent) - Y)
    theta_gradient_descent = theta_gradient_descent - eta * gradients

print(theta_gradient_descent)

model_Y_gradient_descent =  theta_gradient_descent[0] \
    + theta_gradient_descent[1]*model_X

plt.figure()
plt.plot(X,Y,'o',markersize=3,label='data points')
plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('price (USD)')
plt.plot(model_X,model_Y_closed_form,'-',label='normal equation')
plt.plot(model_X,model_Y_gradient_descent,':',label='gradient descent')
plt.grid(True)
plt.legend()
plt.tight_layout()

#%% compute the linear regression solution using sk-learn

# build and train a closed from linear regression model in sk-learn
model_LR = sk.linear_model.LinearRegression()
model_LR.fit(X,Y[:,0])
model_Y_sk_LR = model_LR.predict(model_X)

# build and train a Stochastic Gradient Descent linear regression model in sk-learn.
# Note that in running the model, the best way to do this would be to use a pipeline
# =with feature scaling. However, here we just set 'eta0' to a low value, this
# is done only for educational # purposes and is not the ideal methodology in
# terms of system robustness.
model_SGD = sk.linear_model.SGDRegressor(learning_rate='constant',eta0=0.00000001)
model_SGD.fit(X,Y[:,0])
```

```
model_Y_sk_SGD = model_SGD.predict(model_X)

# plot the modeled results
plt.figure()
plt.plot(X,Y,'o',markersize=2,label='data')
plt.plot(model_X,model_Y_closed_form,'-',label='normal equation')
plt.plot(model_X,model_Y_gradient_descent,'--',label='gradient descent')
plt.plot(model_X,model_Y_sk_LR,':',label='sklearn normal equation')
plt.plot(model_X,model_Y_sk_SGD,'-.',label='sklearn stochastic gradient descent')

plt.xlabel('Above grade (ground) living area square feet')
plt.ylabel('price (USD)')
plt.grid(True)
plt.legend(framealpha=1)
plt.tight_layout()
```