```
import numpy as np

def ridge_regression(X, y, lam):
    n, d = X.shape

    # Add bias term by appending a column of ones
    X_tilde = np.hstack([X, np.ones((n, 1))])

    # Construct regularization matrix (do not regularize bias term)
    reg = 2 * lam * np.eye(d + 1)
    reg[-1, -1] = 0

    # Closed-form solution: (X^T X + reg)^(-1) X^T y
    A = X_tilde.T @ X_tilde + reg
    b_vec = X_tilde.T @ y
    w_tilde = np.linalg.solve(A, b_vec)

    w = w_tilde[:-1]
    b = w_tilde[-1]

    return w, b

import numpy as np

def ridge_regression_gd(X, y, lam=1.0, eta=0.01, max_pass=1000,
tol=1e-5):
    n, d = X.shape
    w = np.zeros(d)
    b = 0.0
    losses = []

    for t in range(max_pass):
        # Compute predictions
        y_pred = X @ w + b

        # Compute gradients
        grad_w = (1/n) * X.T @ (y_pred - y) + 2 * lam * w
        grad_b = (1/n) * np.sum(y_pred - y)

        # Update parameters
        w_new = w - eta * grad_w
        b_new = b - eta * grad_b

        # Compute loss (consistent with compute_loss function)
        mse = (1/n) * np.sum((y_pred - y)**2)  # Standard MSE without
the 1/2 factor
        regularization = lam * np.sum(w**2)
        loss = mse + regularization
        losses.append(loss)
```

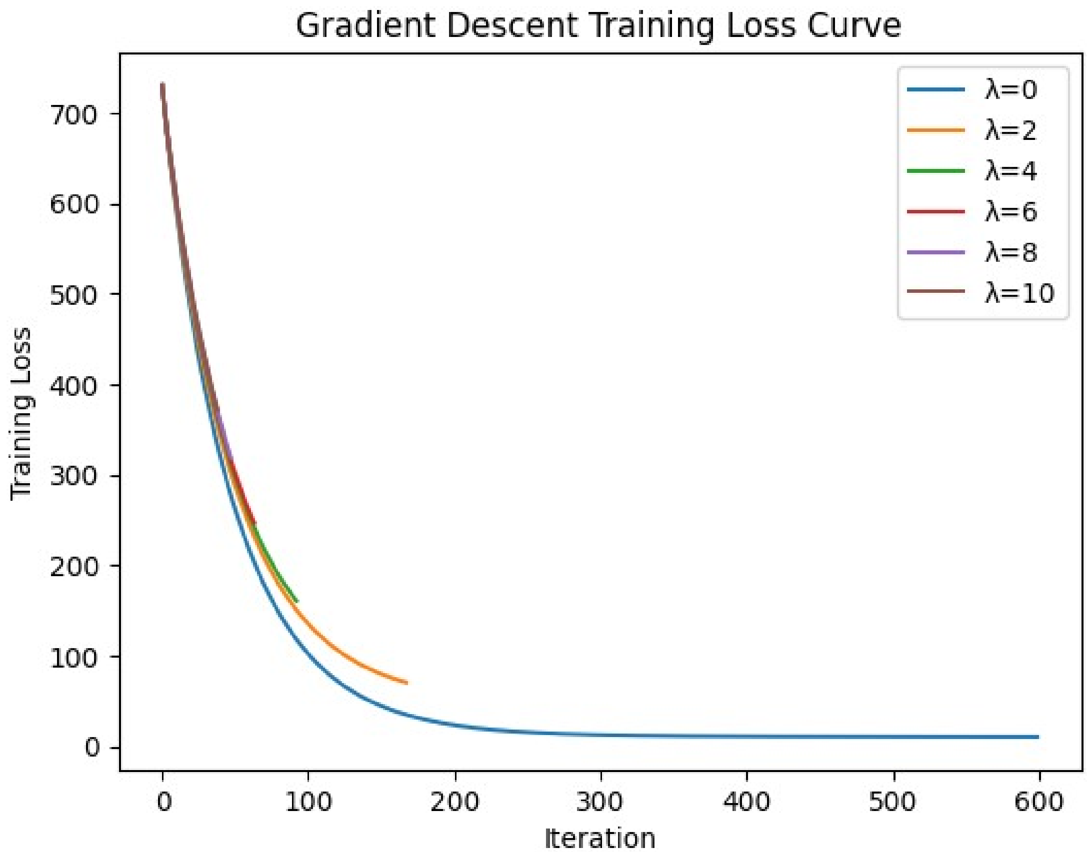
```python
        # Check convergence
        if np.linalg.norm(w_new - w) <= tol:
            w, b = w_new, b_new
            break

        w, b = w_new, b_new


    return w, b, losses

import numpy as np
import time
import matplotlib.pyplot as plt
import pandas as pd

## Standardization function
def standardize(input):
    return (input - np.mean(input, axis=0)) / np.std(input, axis=0)

## Compute error and loss functions
def compute_error(X, y, w, b):
    y_pred = X @ w + b
    return np.mean((y_pred - y)**2)

def compute_loss(X, y, w, b, lam):
    n = X.shape[0]
    y_pred = X @ w + b
    mse = (1/n) * np.sum((y_pred - y)**2)  # Standard MSE without the 
1/2 factor
    regularization = lam * np.sum(w**2)
    return mse + regularization

# Load housing dataset from CSV files
X_train_df = pd.read_csv("./a1-files/housing_X_train.csv", 
header=None)
X_test_df = pd.read_csv("./a1-files/housing_X_test.csv", header=None)
y_train_df = pd.read_csv("./a1-files/housing_y_train.csv", 
header=None)
y_test_df = pd.read_csv("./a1-files/housing_y_test.csv", header=None)

# Convert to numpy and transpose
X_train = X_train_df.values.T
X_test = X_test_df.values.T
y_train = y_train_df.values.ravel()
y_test = y_test_df.values.ravel()

# Fix sample count mismatch
min_train_samples = min(X_train.shape[0], y_train.shape[0])
min_test_samples = min(X_test.shape[0], y_test.shape[0])
```

```python
X_train = X_train[:min_train_samples, :]
y_train = y_train[:min_train_samples]
X_test = X_test[:min_test_samples, :]
y_test = y_test[:min_test_samples]

# Standardize
X_train_std = standardize(X_train)
X_test_std = standardize(X_test)
Y_test_std = (X_test - np.mean(X_train, axis=0)) / np.std(X_train,
axis=0)
Y_train_std = (X_train - np.mean(X_train, axis=0)) / np.std(X_train,
axis=0)

for lam in range(0,11,2):  # Include λ=0 to see unregularized baseline
    print(f"\n--- λ = {lam} ---")

    # Closed form (using function from Cell 1)
    t0 = time.time()
    w_cf, b_cf = ridge_regression(X_train_std, y_train, lam)
    t_cf = time.time() - t0

    train_error_cf = compute_error(X_train_std, y_train, w_cf, b_cf)
    train_loss_cf = compute_loss(X_train_std, y_train, w_cf, b_cf,
lam)
    test_error_cf = compute_error(X_test_std, y_test, w_cf, b_cf)

    print("Closed-form:")
    print(" Training Error:", train_error_cf)
    print(" Training Loss :", train_loss_cf)
    print(" Test Error    :", test_error_cf)
    print(" Time          :", t_cf, "s")

    # Gradient descent (using function from Cell 2)
    t0 = time.time()
    w_gd, b_gd, losses = ridge_regression_gd(X_train_std, y_train,
lam=lam, eta=0.01, max_pass=600)
    t_gd = time.time() - t0

    train_error_gd = compute_error(X_train_std, y_train, w_gd, b_gd)
    train_loss_gd = compute_loss(X_train_std, y_train, w_gd, b_gd,
lam)
    test_error_gd = compute_error(X_test_std, y_test, w_gd, b_gd)

    print("Gradient descent:")
    print(" Training Error:", train_error_gd)
    print(" Training Loss :", train_loss_gd)
    print(" Test Error    :", test_error_gd)
    print(" Time          :", t_gd, "s")
```

```python
    # Plot training loss curve
    plt.plot(losses, label=f"λ={lam}")
    plt.xlabel("Iteration")
    plt.ylabel("Training Loss")
    plt.title("Gradient Descent Training Loss Curve")
    plt.legend()


plt.show()
```

```
--- λ = 0 ---
Closed-form:
 Training Error: 9.69429863890932
 Training Loss : 9.69429863890932
 Test Error    : 128.40265959086983
 Time          : 0.00016283988952636 72 s
Gradient descent:
 Training Error: 10.02168615688538
 Training Loss : 10.02168615688538
 Test Error    : 119.75644528434111
 Time          : 0.009572982788085938 s

--- λ = 2 ---
Closed-form:
 Training Error: 9.711966530146903
 Training Loss : 109.0461782997462
 Test Error    : 126.38550053839941
 Time          : 0.0008127689361572266 s
Gradient descent:
 Training Error: 62.96144309495101
 Training Loss : 69.91283555325397
 Test Error    : 51.141476264087814
 Time          : 0.0023310184478759766 s

--- λ = 4 ---
Closed-form:
 Training Error: 9.757922397503231
 Training Loss : 198.9703631105295
 Test Error    : 124.62522607724712
 Time          : 3.695487976074219e-05 s
Gradient descent:
 Training Error: 153.1073927233596
 Training Loss : 158.4488557942555
 Test Error    : 52.25736974198957
 Time          : 0.0013113021850585938 s

--- λ = 6 ---
Closed-form:
 Training Error: 9.824337191338161
```

```
 Training Loss : 281.3925062287277
 Test Error    : 123.0718508308986
 Time          : 2.5987625122070312e-05 s
Gradient descent:
 Training Error: 238.76609446912357
 Training Loss : 243.0357241827009
 Test Error    : 82.70542588391771
 Time          : 0.0009562969207763672 s

--- λ = 8 ---
Closed-form:
 Training Error: 9.905917248316513
 Training Loss : 357.68904036803826
 Test Error    : 121.68795213345949
 Time          : 3.910064697265625e-05 s
Gradient descent:
 Training Error: 310.83499329234
 Training Loss : 314.38008409623745
 Test Error    : 117.14517259815631
 Time          : 0.0006990432739257812 s

--- λ = 10 ---
Closed-form:
 Training Error: 9.998990986418427
 Training Loss : 428.87164411031347
 Test Error    : 120.44490487858737
 Time          : 2.9087066650390625e-05 s
Gradient descent:
 Training Error: 362.6358955396609
 Training Loss : 365.66359439397917
 Test Error    : 144.9650827586216
 Time          : 0.0005607604980468750 s
```


Gradient Descent Training Loss Curve
λ=0
λ=2
λ=4
λ=6
λ=8
λ=10
Training Loss
0
100
200
300
400
500
600
700
Iteration
0
100
200
300
400
500
600