In [143…
```
'''
Patrick Ballou
ID: 801130521
ECGR 4105
Homework 1
Problem 3
'''
```

Out[143]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4105\nHomework 1\nProblem 3\n'
```

In [144…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
SS = StandardScaler()
MM = MinMaxScaler()
```

In [145…
```
#create pandas dataframe and print first 5 rows
df = pd.read_csv("Housing.csv")
df_copy = df.copy()
df.head()
```

Out[145]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheatin
0	13300000	7420	4	2	3	yes	no	no	n
1	12250000	8960	4	4	4	yes	no	no	n
2	12250000	9960	3	2	2	yes	no	yes	n
3	12215000	7500	4	2	2	yes	no	yes	n
4	11410000	7420	4	1	2	yes	yes	yes	n

In [146…
```
#categorical inputs that need to be mapped to numbers
non_num_varlist = ["mainroad", "guestroom", "basement", "hotwaterheating", "airconditi

#mapping function
def to_num(x):
    return x.map({"yes": 1, "no": 0})
```

In [147…
```
#map inputs and output new dataframe
df[non_num_varlist] = df_copy[non_num_varlist].apply(to_num) #copy df is to avoid prob
df.head()
```

Out[147]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheatin
0	13300000	7420	4	2	3	1	0	0	
1	12250000	8960	4	4	4	1	0	0	
2	12250000	9960	3	2	2	1	0	1	
3	12215000	7500	4	2	2	1	0	1	
4	11410000	7420	4	1	2	1	1	1	

In [148…

```
#train/test split, random_state functions as seed
df_train, df_test = train_test_split(df, train_size=.8, test_size=.2, random_state=7)
```

In [149…

```
#create arrays of relevent inputs for this problem

#part a
vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
train_set = df_train[vars]
test_set = df_test[vars]



#part b
vars_b = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'baseme
train_set_b = df_train[vars_b]
test_set_b = df_test[vars_b]
```

In [150…

```
#scale two different ways

#part a
ss_train_set = SS.fit_transform(train_set)
ss_test_set = SS.fit_transform(test_set)
mm_train_set = MM.fit_transform(train_set)
mm_test_set = MM.fit_transform(test_set)


#part b
ss_train_set_b = SS.fit_transform(train_set_b)
ss_test_set_b = SS.fit_transform(test_set_b)
mm_train_set_b = MM.fit_transform(train_set_b)
mm_test_set_b = MM.fit_transform(test_set_b)
```

In [151…

```
#create variables for test and train set since they are different sizes

m_train = len(train_set)
m_test = len(test_set)

#standard scaler train
ss_Y_train = ss_train_set[:,-1]
ss_x_train = ss_train_set[:,0:6]
ss_x_0_train = np.ones((m_train,1))
ss_X_train = np.hstack((ss_x_0_train, ss_x_train))

#standard scaler test
ss_Y_test = ss_test_set[:,-1]
```

```
ss_x_test = ss_test_set[:, 0:6]
ss_x_0_test = np.ones((m_test,1))
ss_X_test = np.hstack((ss_x_0_test, ss_x_test))

#min max train
mm_Y_train = mm_train_set[:,-1]
mm_x_train = mm_train_set[:,0:6]
mm_x_0_train = np.ones((m_train,1))
mm_X_train = np.hstack((mm_x_0_train, mm_x_train))

#Min max test
mm_Y_test = mm_test_set[:,-1]
mm_x_test = mm_test_set[:, 0:6]
mm_x_0_test = np.ones((m_test,1))
mm_X_test = np.hstack((mm_x_0_test, mm_x_test))
```

In [152…

```
#same thing but for part b

m_train_b = len(train_set_b)
m_test_b = len(test_set_b)

#standard scaler train
ss_Y_train_b = ss_train_set_b[:,-1]
ss_x_train_b = ss_train_set_b[:,0:12]
ss_x_0_train_b = np.ones((m_train_b,1))
ss_X_train_b = np.hstack((ss_x_0_train_b, ss_x_train_b))

#standard scaler test
ss_Y_test_b = ss_test_set_b[:,-1]
ss_x_test_b = ss_test_set_b[:, 0:12]
ss_x_0_test_b = np.ones((m_test_b,1))
ss_X_test_b = np.hstack((ss_x_0_test_b, ss_x_test_b))

#min max train
mm_Y_train_b = mm_train_set_b[:,-1]
mm_x_train_b = mm_train_set_b[:,0:12]
mm_x_0_train_b = np.ones((m_train_b,1))
mm_X_train_b = np.hstack((mm_x_0_train_b, mm_x_train_b))

#Min max test
mm_Y_test_b = mm_test_set_b[:,-1]
mm_x_test_b = mm_test_set_b[:, 0:12]
mm_x_0_test_b = np.ones((m_test_b,1))
mm_X_test_b = np.hstack((mm_x_0_test_b, mm_x_test_b))
```

In [153…

```
#initialize theta, # of iterations, lambda, and learning rate

#part a
iterations = 5000
ss_theta = np.zeros(7)
ss_alpha = .007
ss_lambda = .01

mm_theta = np.zeros(7)
mm_alpha = .02
mm_lambda = .01
```

```
#part b
iterations_b = 5000
ss_theta_b = np.zeros(13)
ss_alpha_b = .008
ss_lambda_b = .01

mm_theta_b = np.zeros(13)
mm_alpha_b = .01
mm_lambda_b = .01
```

In [154…

```
#initialize cost history arrays
ss_train_cost_history = np.zeros(iterations)
ss_test_cost_history = np.zeros(iterations)
mm_train_cost_history = np.zeros(iterations)
mm_test_cost_history = np.zeros(iterations)


ss_train_cost_history_b = np.zeros(iterations_b)
ss_test_cost_history_b = np.zeros(iterations_b)
mm_train_cost_history_b = np.zeros(iterations_b)
mm_test_cost_history_b = np.zeros(iterations_b)
```

In [155…

```
#loss function
def compute_cost(X, y, theta, m):
    predictions = X.dot(theta)
    errors = np.subtract(predictions, y)
    sqrErrors = np.square(errors)
    J = (1/(2*m))*np.sum(sqrErrors)

    return J
```

In [156…

```
#gradient descent function with parameter penalties
def gradient_descent(x_train, y_train, x_test, y_test, theta, alpha, iterations, m_tra

    for i in range(iterations):
        predictions = x_train.dot(theta)
        errors = np.subtract(predictions, y_train)
        sum_delta = (alpha/m_train) * (x_train.transpose().dot(errors) + lambda_val*th
        theta -= sum_delta
        train_cost_history[i] = compute_cost(x_train, y_train, theta, m_train)
        test_cost_history[i] = compute_cost(x_test, y_test, theta, m_test)

    return theta, train_cost_history, test_cost_history
```

In [157…

```
#part a
#standard scaler
ss_theta, ss_train_cost_history, ss_test_cost_history = gradient_descent(ss_X_train, s
print("Final theta values for part a with standard scaler:", ss_theta)

#min max
mm_theta, mm_train_cost_history, mm_test_cost_history = gradient_descent(mm_X_train, m
print("Final theta values for part b with minmax:", mm_theta)


#part b
#standard scaler
ss_theta_b, ss_train_cost_history_b, ss_test_cost_history_b = gradient_descent(ss_X_tr
print("Final theta values for part b with standard scaler:", ss_theta_b)
```

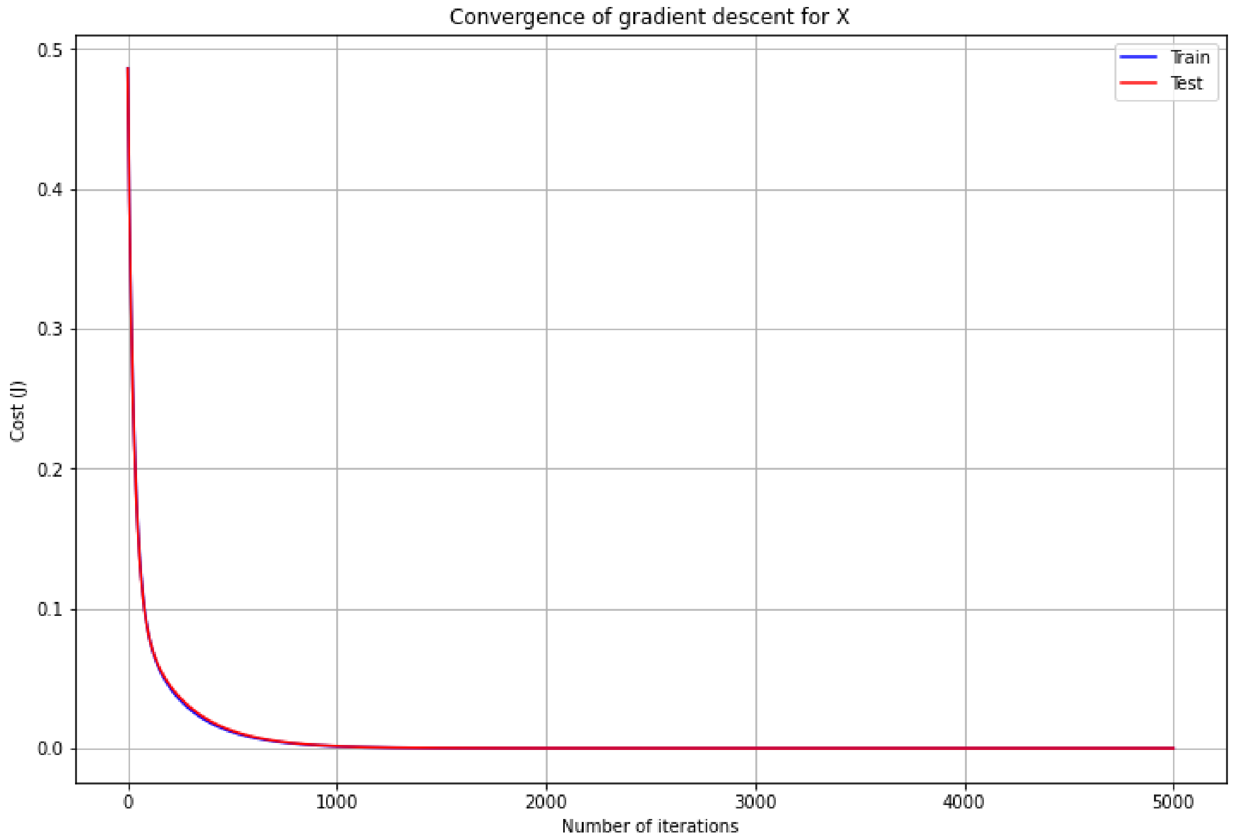
```
#min max
mm_theta_b, mm_train_cost_history_b, mm_test_cost_history_b = gradient_descent(mm_X_tr
print("Final theta values for part b with min max:", mm_theta_b)
```

```
Final theta values for part a with standard scaler: [1.09630087e-19 3.34579794e-05 3.
13108949e-06 2.30000087e-05
 1.90017083e-05 1.04089160e-05 9.99928187e-01]
Final theta values for part b with minmax: [0.00846087 0.16315827 0.0167544  0.102277
21 0.0378647  0.02180989
 0.70661407]
Final theta values for part b with standard scaler: [8.31713901e-20 5.02435226e-05 2.
42229549e-06 3.74117279e-05
 3.12363430e-05 1.03020582e-05 4.63544621e-06 2.00094575e-05
 1.41884234e-05 2.99677873e-05 1.49921293e-05 1.69864908e-05
 9.99869549e-01]
Final theta values for part b with min max: [0.00116796 0.15461798 0.04162156 0.13521
804 0.06933473 0.02519822
 0.01685638 0.02061556 0.04478528 0.04438587 0.05547643 0.0333198
 0.43023494]
```

In [158…

```
#plot standard scaler train and test loss vs iterations for part a
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(range(1, iterations + 1), ss_train_cost_history, color='blue', label='Train')
plt.plot(range(1, iterations + 1), ss_test_cost_history, color='red', label='Test')
plt.grid()
plt.legend()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent for X')
print("Standard scaler training cost:", ss_train_cost_history[-1])
print("Standard scaler testing cost:", ss_test_cost_history[-1])
```

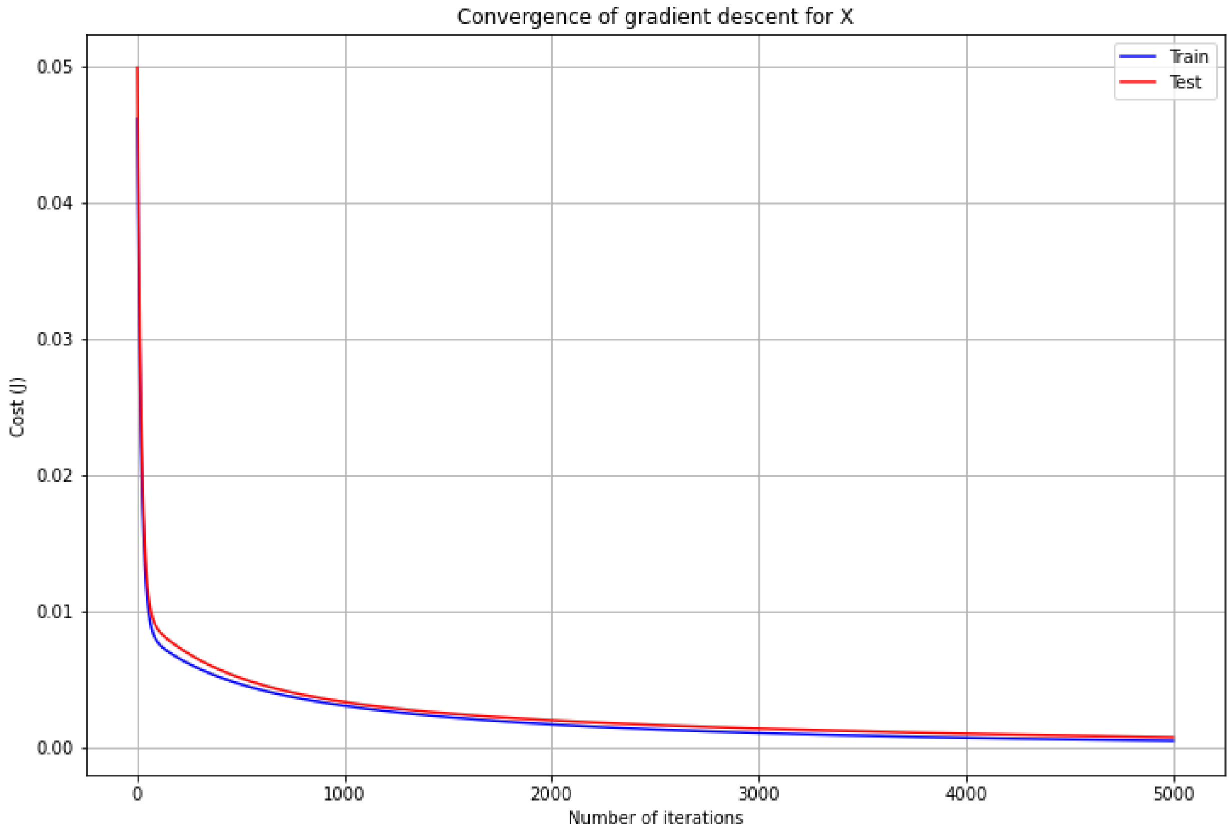
```
Standard scaler training cost: 1.1257937437554153e-09
Standard scaler testing cost: 1.2092123592210432e-09
```



In [159...

```
#plot Min max train and test loss vs iterations for part a
plt.plot(range(1, iterations + 1), mm_train_cost_history, color='blue', label='Train')
plt.plot(range(1, iterations + 1), mm_test_cost_history, color='red', label='Test')
plt.grid()
plt.legend()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent for X')
print("Min max training cost:", mm_train_cost_history[-1])
print("Min max testing cost:", mm_test_cost_history[-1])
```

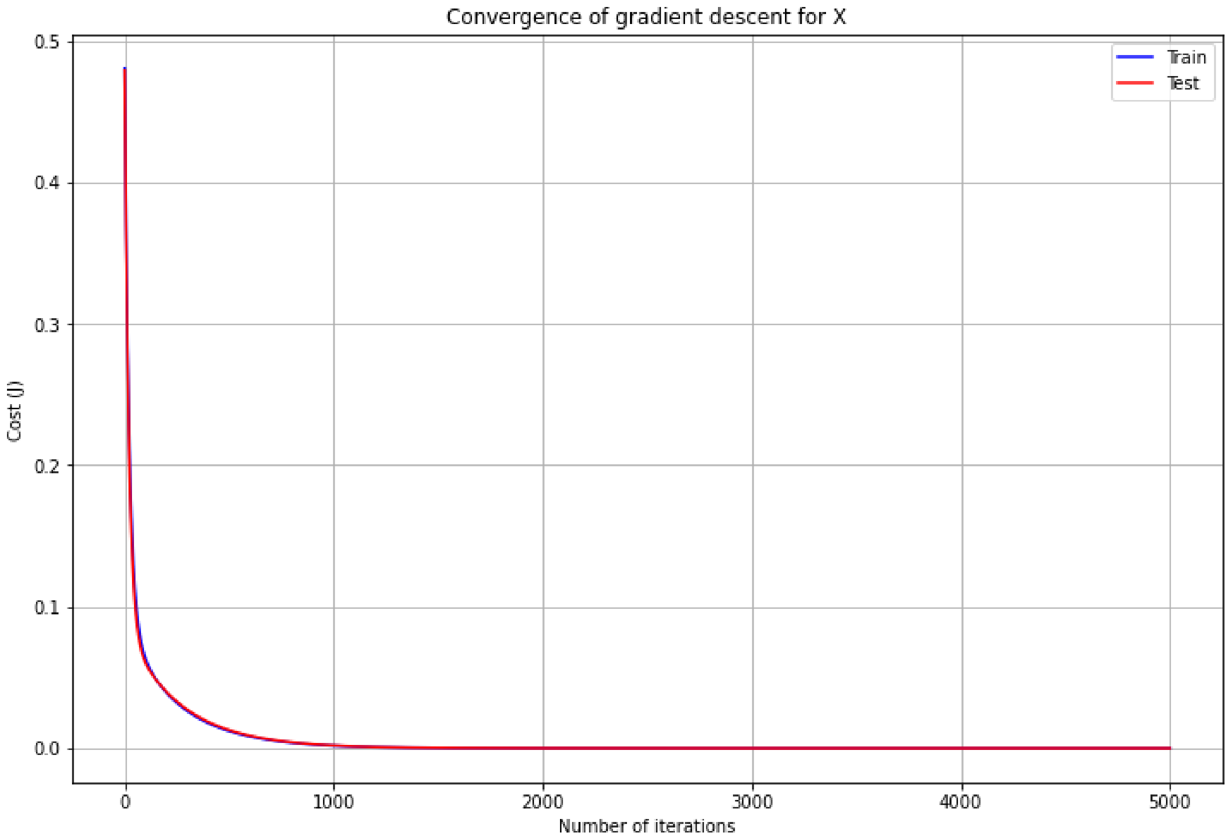
```
Min max training cost: 0.0005096466463897371
Min max testing cost: 0.0007646685550530708
```



In [160…

```
#plot standard scaler train and test loss vs iterations for part b
plt.plot(range(1, iterations_b + 1), ss_train_cost_history_b, color='blue', label='Tra
plt.plot(range(1, iterations_b + 1), ss_test_cost_history_b, color='red', label='Test'
plt.grid()
plt.legend()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent for X')
print("Standard scaler training cost part b:", ss_train_cost_history_b[-1])
print("Standard scaler testing cost part b:", ss_test_cost_history_b[-1])
```

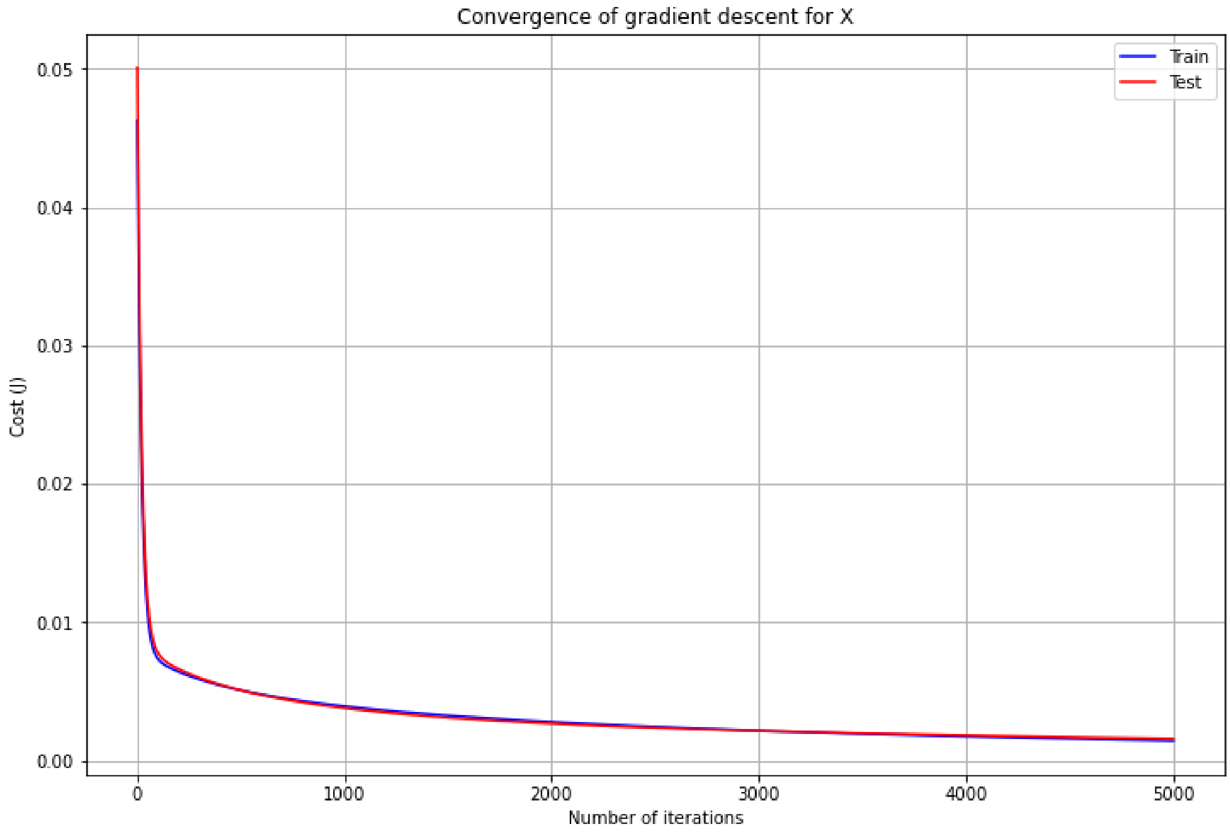
```
Standard scaler training cost part b: 2.8337931624083455e-09
Standard scaler testing cost part b: 2.9944181125407805e-09
```



In [161…

```
#plot Min max train and test loss vs iterations for part b
plt.plot(range(1, iterations_b + 1), mm_train_cost_history_b, color='blue', label='Tra
plt.plot(range(1, iterations_b + 1), mm_test_cost_history_b, color='red', label='Test'
plt.grid()
plt.legend()
plt.xlabel('Number of iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of gradient descent for X')
print("Min max training cost part b:", mm_train_cost_history_b[-1])
print("Min max testing cost part b:", mm_test_cost_history_b[-1])
```

```
Min max training cost part b: 0.001435776583380898
Min max testing cost part b: 0.001589437180705151
```


Convergence of gradient descent for X
Train
Test
Cost (J)
0.05
0.04
0.03
0.02
0.01
0.00
0
1000
2000
3000
4000
5000
Number of iterations