In [80]:
```
'''
Patrick Ballou
ID: 801130521
ECGR 4105
Homework 1
Problem 1
'''
```

Out[80]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4105\nHomework 1\nProblem 1\n'
```

In [81]:
```
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
```

In [82]:
```
#create pandas dataframe and print first 5 rows
df = pd.read_csv("Housing.csv")
df_copy = df.copy() #copy will help later on
df.head()
```

Out[82]:

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

In [83]:
```
#categorical inputs that need to be mapped to numbers
non_num_varlist = ["mainroad", "guestroom", "basement", "hotwaterheating", "airconditi
```

In [84]:
```
#mapping function
def to_num(x):
    return x.map({"yes": 1, "no": 0})
```

In [85]:
```
#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[85]:

	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 [86]:
```
#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 [87]:
```
#don't need all variables for problem 1
vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
train_set = df_train[vars]
test_set = df_test[vars]
```

In [88]:
```
#create variables for test and train set since they are different sizes
m_train = len(train_set)
x_train = train_set[['area', 'bedrooms', 'bathrooms', 'stories', 'parking']]
x_0_train = np.ones((m_train,1))
X_train = np.hstack((x_0_train, x_train))
Y_train = train_set['price']

m_test = len(test_set)
x_test = test_set[['area', 'bedrooms', 'bathrooms', 'stories', 'parking']]
x_0_test = np.ones((m_test,1))
X_test = np.hstack((x_0_test, x_test))
Y_test = test_set['price']
```

In [89]:
```
#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 [90]:
```
#gradient descent function
def gradient_descent(X, y, theta, alpha, iterations):
    train_cost_history = np.zeros(iterations)
    test_cost_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha/m_train) * X.transpose().dot(errors)
        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
```

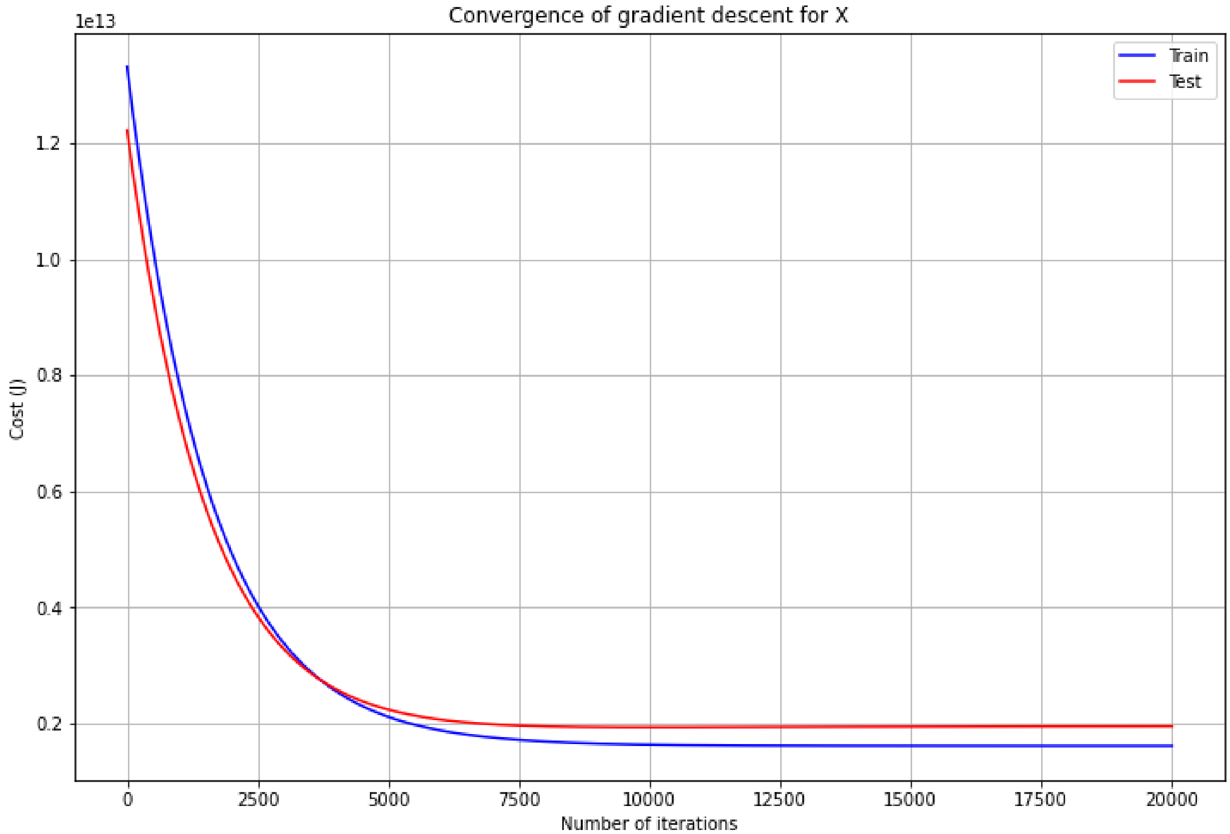
In [91]:
```
#initialize theta, # of iterations, and learning rate
theta = np.zeros(6)
iterations = 20000
#very small theta because input 'area' is messing up the model since it is not scaled
alpha = .00000000001
```

In [92]:
```
#calculate test and train costs, and theta
theta, train_cost_history, test_cost_history = gradient_descent(X_train, Y_train, thet
print("Final theta values for a:", theta)
```

```
Final theta values for a: [2.11132525e-01 8.61083612e+02 6.94719483e-01 3.44382851e-0
1
 4.97239587e-01 1.72230062e-01]
```

In [93]:
```
#plot loss vs iterations
plt.rcParams["figure.figsize"] = (12,8)
plt.plot(range(1, iterations + 1), train_cost_history, color='blue', label='Train')
plt.plot(range(1, iterations + 1), 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("Training cost:", train_cost_history[-1])
print("Testing cost:", test_cost_history[-1])
```

```
Training cost: 1606259143218.0923
Testing cost: 1944119106864.634
```



In [94]:
```
#Problem 1, part b
#same process as part a except more input variables

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


#train and test set variables
m_train_b = len(train_set_b)
x_train_b = train_set_b[['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'gues
x_0_train_b = np.ones((m_train_b,1))
X_train_b = np.hstack((x_0_train_b, x_train_b))
Y_train_b = train_set_b['price']
```

```
m_test_b = len(test_set_b)
x_test_b = test_set_b[['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestr
x_0_test_b = np.ones((m_test_b,1))
X_test_b = np.hstack((x_0_test_b, x_test_b))
Y_test_b = test_set_b['price']


#small theta is needed for same reason as part a
theta_b = np.zeros(12)
iterations_b = 1600
alpha_b = .0000000001
```

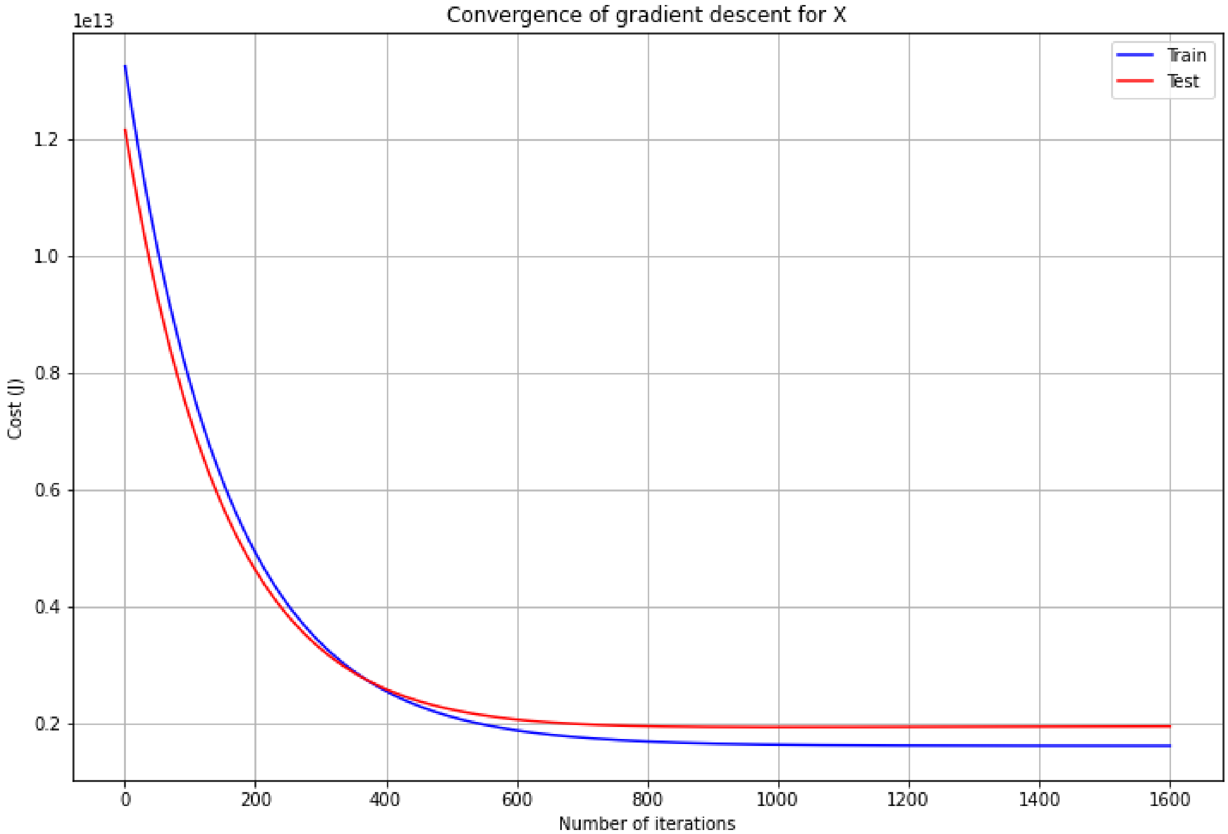
In [95]:
```
#new gradient descent for part b to avoid problems, same structure as part a
def gradient_descent_b(X, y, theta, alpha, iterations):
    train_cost_history_b = np.zeros(iterations)
    test_cost_history_b = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = np.subtract(predictions, y)
        sum_delta = (alpha/m_train_b) * X.transpose().dot(errors)
        theta -= sum_delta
        train_cost_history_b[i] = compute_cost(X_train_b, Y_train_b, theta, m_train_b)
        test_cost_history_b[i] = compute_cost(X_test_b, Y_test_b, theta, m_test_b)

    return theta, train_cost_history_b, test_cost_history_b
```

In [96]:
```
theta_b, train_cost_history_b, test_cost_history_b = gradient_descent_b(X_train_b, Y_t
print("Final theta values for b:", theta_b)
```

```
Final theta values for b: [1.96526674e-01 8.57123320e+02 6.39843637e-01 3.12821014e-0
1
 4.49239267e-01 1.74607418e-01 5.08318396e-02 9.38560300e-02
 1.43031168e-02 1.02501195e-01 1.61114151e-01 6.11611234e-02]
```

In [97]:
```
plt.plot(range(1, iterations_b + 1), train_cost_history_b, color='blue', label='Train'
plt.plot(range(1, iterations_b + 1), 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("Training cost:", train_cost_history_b[-1])
print("Testing cost:", test_cost_history_b[-1])
```

```
Training cost: 1606703724154.2678
Testing cost: 1940242093479.1543
```


Convergence of gradient descent for X
1e13
Train
Test
1.2
1.0
0.8
0.6
0.4
0.2
Cost (J)
0
200
400
600
800
1000
1200
1400
1600
Number of iterations