In [20]:
```
# Ryan Picariello - 800856548 - Homework 1 Part 1a
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [21]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [22]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[22]:

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

In [23]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[23]:

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

In [24]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [25]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Newtrain.head()
```

Out[25]:

	area	bedrooms	bathrooms	stories	parking	price
454	4500	3	1	2	0	3143000
392	3990	3	1	2	0	3500000
231	4320	3	1	1	0	4690000
271	1905	5	1	2	0	4340000
250	3510	3	1	3	0	4515000

In [26]:
```
XTrain = df_Newtrain.values[:,[0,1,2,3,4]]
YTrain = df_Newtrain.values[:,5]

XTest = df_Newtest.values[:,[0,1,2,3,4]]
YTest = df_Newtest.values[:,5]
```

In [27]:
```
mean = np.ones(XTrain.shape[1])
std = np.ones(XTrain.shape[1])
for i in range(0, XTrain.shape[1]):
    mean[i] = np.mean(XTrain.transpose()[i])
    std[i] = np.std(XTrain.transpose()[i])
    for j in range(0, XTrain.shape[0]):
        XTrain[j][i] = (XTrain[j][i] - mean[i])/std[i]
```

In [28]:
```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [29]:
```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [30]:
```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    for i in range(0,iterations):
```

```
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [31]:

```
def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [32]:

```
iterations = 500;
alpha = 0.1;
alpha2 = 0.01
```

In [33]:

```
ThetaTraining, CostTraining = linear_regression(XTrain, YTrain, alpha, iterations)
print('Final value of theta with an alpha of 0.1 =', ThetaTraining)
CostTraining = list(CostTraining)
nIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of 0.1 = [[4112038.79202804  792419.7178822   507988.
14580124 1057659.53538904
   891202.57476334  441457.24168317]]
```

In [34]:

```
ThetaTraining2, CostTraining2 = linear_regression(XTrain, YTrain, alpha2, iterations)
print('Final value of theta with an alpha of 0.01 =', ThetaTraining2)
CostTraining2 = list(CostTraining2)
nIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of 0.01 = [[3911369.42084099  684721.7983618   36402
6.32729177 1215935.510898
   993151.45290111  772794.71890888]]
```

In [35]:

```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta with an alpha of 0.1 =', theta_Test)
cost_Test = list(cost_Test)
nIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of 0.1 = [[4009323.46427773  844638.61768703  225437.
77741561  911745.77297157
   885446.81234427  751101.29064712]]
```

In [36]:

```
theta_Test2, cost_Test2 = linear_regression(XTest, YTest, alpha2, iterations)
print('Final value of theta with an alpha of 0.01 =', theta_Test2)
cost_Test2 = list(cost_Test2)
nIterations_Test2 = [x for x in range(1,(iterations + 1))]
```

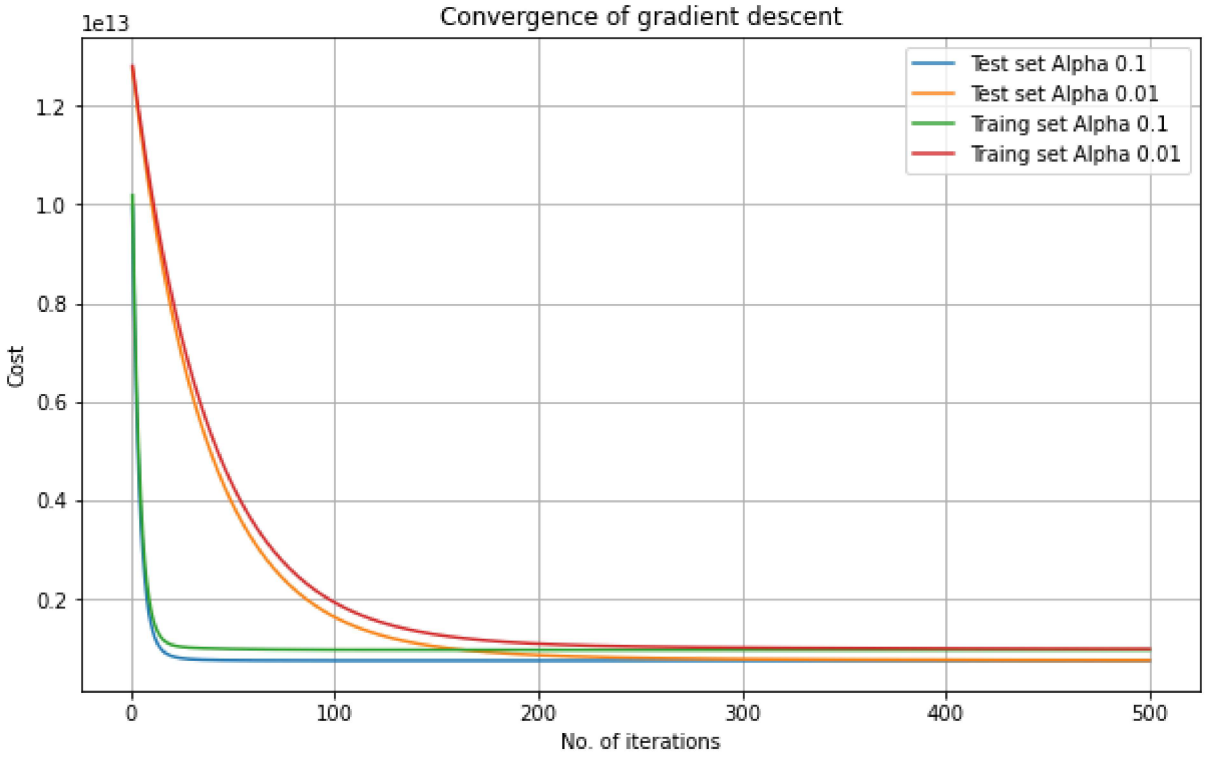
```
Final value of theta with an alpha of 0.01 = [[3896885.81334708  798864.59108174  15151
0.77459081 1093108.39710527
```

```
870883.11233557  848681.31817011]]
```

In [38]:

```
plt.plot(nIterations_Test, cost_Test, label='Test set Alpha 0.1')
plt.plot(nIterations_Test2, cost_Test2, label='Test set Alpha 0.01')
plt.plot(nIterations_Training, CostTraining, label='Traing set Alpha 0.1')
plt.plot(nIterations_Training2, CostTraining2, label='Traing set Alpha 0.01')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[38]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]:

In [21]:
```
# Ryan Picariello - 822856548 - Intro to ML Homework 1 part 1b
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [22]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [23]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[23]:

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

In [24]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
 return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[24]:

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

In [25]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [26]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'mainroad', 'guestroom', 'basement', 'hotw
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Newtrain.head()
```

Out[26]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning
454	4500	3	1	1	0	0	0	1
392	3990	3	1	1	0	0	0	0
231	4320	3	1	1	0	0	0	0
271	1905	5	1	0	0	1	0	0
250	3510	3	1	1	0	0	0	0

In [27]:

```
XTrain = df_Newtrain.values[:,0:10]
YTrain = df_Newtrain.values[:,10]

XTest = df_Newtest.values[:,0:10]
YTest = df_Newtest.values[:,10]
```

In [28]:

```
mean = np.ones(XTrain.shape[1])
std = np.ones(XTrain.shape[1])
for i in range(0, XTrain.shape[1]):
    mean[i] = np.mean(XTrain.transpose()[i])
    std[i] = np.std(XTrain.transpose()[i])
    for j in range(0, XTrain.shape[0]):
        XTrain[j][i] = (XTrain[j][i] - mean[i])/std[i]
```

In [29]:

```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [30]:

```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [31]:

```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
```

```
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [32]:

```
def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [33]:

```
iterations = 500;
alpha = 0.1;
alpha2 = 0.01
```

In [34]:

```
ThetaTraining, CostTraining = linear_regression(XTrain, YTrain, alpha, iterations)
print('Final value of theta with an alpha of ', alpha,' =', ThetaTraining)
CostTraining = list(CostTraining)
nIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of  0.1  = [[3670731.06244055  551896.57649369  38579
5.47482823  992491.61099674
   406487.69378465  296569.19055127  109169.67597818  340894.17909666
  1248570.78075159  254848.24455347  899005.83201632]]
```

In [35]:

```
ThetaTraining2, CostTraining2 = linear_regression(XTrain, YTrain, alpha2, iterations)
print('Final value of theta with an alpha of ', alpha2,' =', ThetaTraining2)
CostTraining2 = list(CostTraining2)
nIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of  0.01  = [[3237989.41217222  547127.46438364  2470
32.84559761 1022333.00298265
   153039.07459016  364712.32134447  470973.32733094  381655.9909572
  1307691.88614664  604186.7049498   911391.41717596]]
```

In [36]:

```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta with an alpha of ', alpha,' =', theta_Test)
cost_Test = list(cost_Test)
nIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of  0.1  = [[3665283.80998753  741402.85068115  26465
7.01674763  992948.00356364
   305680.32344454  126509.594599    265573.12956302   96887.61766261
  1318395.63714537  574221.3020385   498783.75949985]]
```

In [37]:

```
theta_Test2, cost_Test2 = linear_regression(XTest, YTest, alpha2, iterations)
print('Final value of theta with an alpha of ', alpha2,' =', theta_Test2)
```

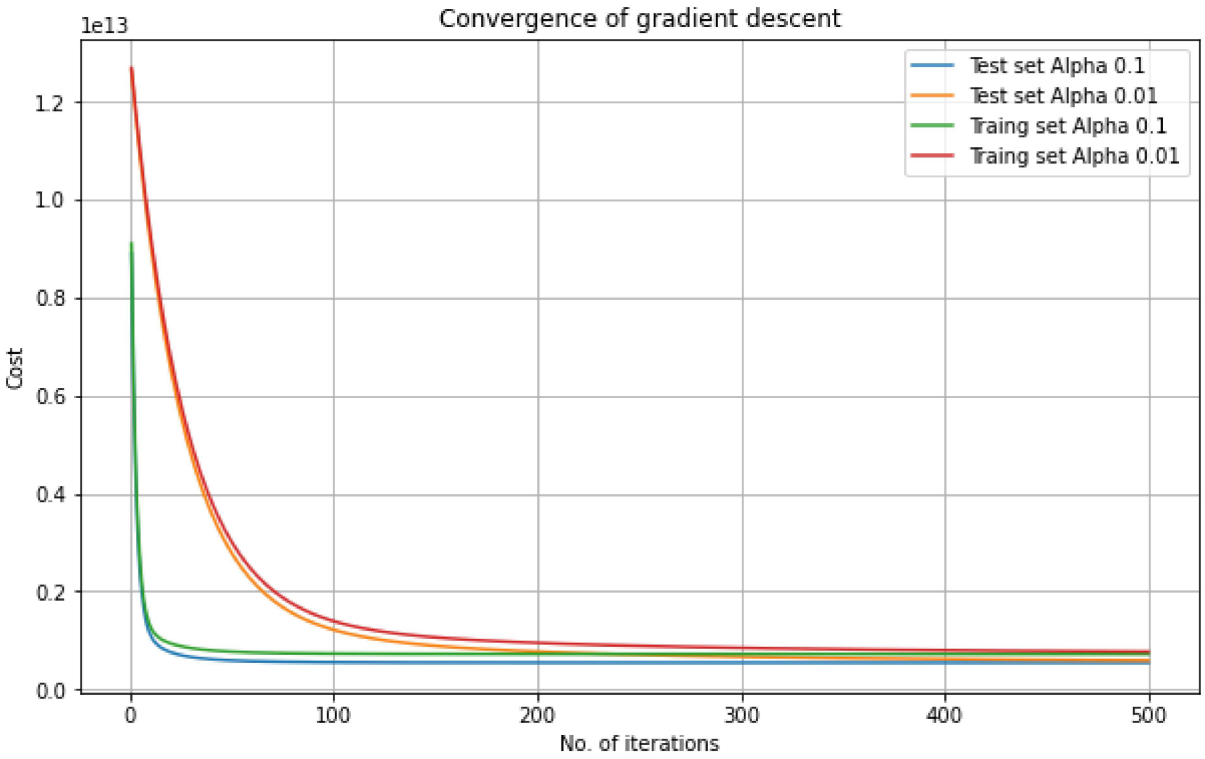
```
cost_Test2 = list(cost_Test2)
nIterations_Test2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with an alpha of  0.01  = [[3211733.75281949  791345.02851107  1628
38.95231366 1164613.49088194
    51779.18076014  239993.96046932  566736.95476624  137642.82043416
  1204854.90843431  783960.095531    689075.93949973]]
```

In [39]:

```
plt.plot(nIterations_Test, cost_Test, label='Test set Alpha 0.1')
plt.plot(nIterations_Test2, cost_Test2, label='Test set Alpha 0.01')
plt.plot(nIterations_Training, CostTraining, label='Traing set Alpha 0.1')
plt.plot(nIterations_Training2, CostTraining2, label='Traing set Alpha 0.01')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[39]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]:

In [1]:
```
# Ryan Picariello - 800856548 - Intro to ML Homework 1 Part 2a
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [3]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[3]:

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

In [4]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[4]:

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

In [5]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [6]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Normalization = df_Newtrain
df_Standardization  = df_Newtrain
df_Newtrain.head()
```

Out[6]:

	area	bedrooms	bathrooms	stories	parking	price
454	4500	3	1	2	0	3143000
392	3990	3	1	2	0	3500000
231	4320	3	1	1	0	4690000
271	1905	5	1	2	0	4340000
250	3510	3	1	3	0	4515000

In [7]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_Normalization[num_vars] = scaler.fit_transform(df_Normalization[num_vars])
df_Normalization.head(20)
```

Out[7]:

	area	bedrooms	bathrooms	stories	parking	price
454	0.193548	0.50	0.0	0.333333	0.000000	0.120606
392	0.156495	0.50	0.0	0.333333	0.000000	0.151515
231	0.180471	0.50	0.0	0.000000	0.000000	0.254545
271	0.005013	1.00	0.0	0.333333	0.000000	0.224242
250	0.121622	0.50	0.0	0.666667	0.000000	0.239394
541	0.040976	0.50	0.0	0.000000	0.000000	0.001485
461	0.226969	0.25	0.0	0.000000	0.000000	0.115152
124	0.340671	0.50	0.5	1.000000	0.333333	0.363636
154	0.131793	0.50	0.5	0.333333	0.666667	0.327273
451	0.357018	0.25	0.0	0.000000	0.000000	0.121212
59	0.302528	0.50	0.5	1.000000	0.333333	0.472727
493	0.154316	0.50	0.0	0.000000	0.000000	0.090909
465	0.142691	0.25	0.0	0.000000	0.000000	0.112121
490	0.182650	0.50	0.0	0.333333	0.333333	0.093939

	area	bedrooms	bathrooms	stories	parking	price
540	0.084568	0.25	0.0	0.000000	0.666667	0.006061
406	0.253124	0.25	0.0	0.000000	0.333333	0.148485
289	0.291630	0.25	0.0	0.000000	0.666667	0.212121
190	0.418774	0.75	0.0	0.333333	0.666667	0.284848
55	0.302528	0.50	0.0	0.333333	0.333333	0.484848
171	0.612685	0.50	0.0	0.000000	0.333333	0.303030

In [8]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = StandardScaler()
df_Standardization[num_vars] = scaler.fit_transform(df_Standardization[num_vars])
df_Standardization.head(20)
```

Out[8]:

	area	bedrooms	bathrooms	stories	parking	price
454	-0.286366	0.073764	-0.581230	0.207401	-0.822960	-0.868394
392	-0.544762	0.073764	-0.581230	0.207401	-0.822960	-0.677628
231	-0.377564	0.073764	-0.581230	-0.937813	-0.822960	-0.041744
271	-1.601145	2.884176	-0.581230	0.207401	-0.822960	-0.228768
250	-0.787958	0.073764	-0.581230	1.352614	-0.822960	-0.135256
541	-1.350349	0.073764	-0.581230	-0.937813	-0.822960	-1.603589
461	-0.053303	-1.331442	-0.581230	-0.937813	-0.822960	-0.902058
124	0.739618	0.073764	1.488383	2.497828	0.321375	0.631546
154	-0.717026	0.073764	1.488383	0.207401	1.465710	0.407116
451	0.853616	-1.331442	-0.581230	-0.937813	-0.822960	-0.864653
59	0.473622	0.073764	1.488383	2.497828	0.321375	1.304836
493	-0.559962	0.073764	-0.581230	-0.937813	-0.822960	-1.051678
465	-0.641027	-1.331442	-0.581230	-0.937813	-0.822960	-0.920761
490	-0.362365	0.073764	-0.581230	0.207401	0.321375	-1.032976
540	-1.046354	-1.331442	-0.581230	-0.937813	1.465710	-1.575348
406	0.129094	-1.331442	-0.581230	-0.937813	0.321375	-0.696331
289	0.397623	-1.331442	-0.581230	-0.937813	1.465710	-0.303578
190	1.284276	1.478970	-0.581230	0.207401	1.465710	0.145281
55	0.473622	0.073764	-0.581230	0.207401	0.321375	1.379646
171	2.636548	0.073764	-0.581230	-0.937813	0.321375	0.257496

In [9]:
```
XTrain_N = df_Normalization.values[:,[0,1,2,3,4]]
YTrain_N = df_Normalization.values[:,5]

XTest = df_Newtest.values[:,[0,1,2,3,4]]
YTest = df_Newtest.values[:,5]

XTrain_S = df_Standardization.values[:,[0,1,2,3,4]]
YTrain_S = df_Standardization.values[:,5]
```

In [10]:
```
mean = np.ones(XTrain_N.shape[1])
std = np.ones(XTrain_N.shape[1])
for i in range(0, XTrain_N.shape[1]):
    mean[i] = np.mean(XTrain_N.transpose()[i])
    std[i] = np.std(XTrain_N.transpose()[i])
    for j in range(0, XTrain_N.shape[0]):
        XTrain_N[j][i] = (XTrain_N[j][i] - mean[i])/std[i]
```

In [11]:
```
mean = np.ones(XTrain_S.shape[1])
std = np.ones(XTrain_S.shape[1])
for i in range(0, XTrain_S.shape[1]):
    mean[i] = np.mean(XTrain_S.transpose()[i])
    std[i] = np.std(XTrain_S.transpose()[i])
    for j in range(0, XTrain_S.shape[0]):
        XTrain_S[j][i] = (XTrain_S[j][i] - mean[i])/std[i]
```

In [12]:
```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [13]:
```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [14]:
```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [15]:
```
def linear_regression(X, y, alpha, iterations):
```

```
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [16]:
```
iterations = 500;
alpha = 0.01;
```

In [17]:
```
ThetaTrain, CostTrain = linear_regression(XTrain_N, YTrain_N, alpha, iterations)
print('Final value of theta with normalization =', ThetaTrain)
CostTrain = list(CostTrain)
nIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.29872054e-16 3.79061776e-01 1.10230128e-01
2.94608603e-01
  2.32422016e-01 1.53858866e-01]]
```

In [18]:
```
ThetaTrain2, CostTrain2 = linear_regression(XTrain_S, YTrain_S, alpha, iterations)
print('Final value of theta with standardization =', ThetaTrain2)
CostTrain2 = list(CostTrain2)
nIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

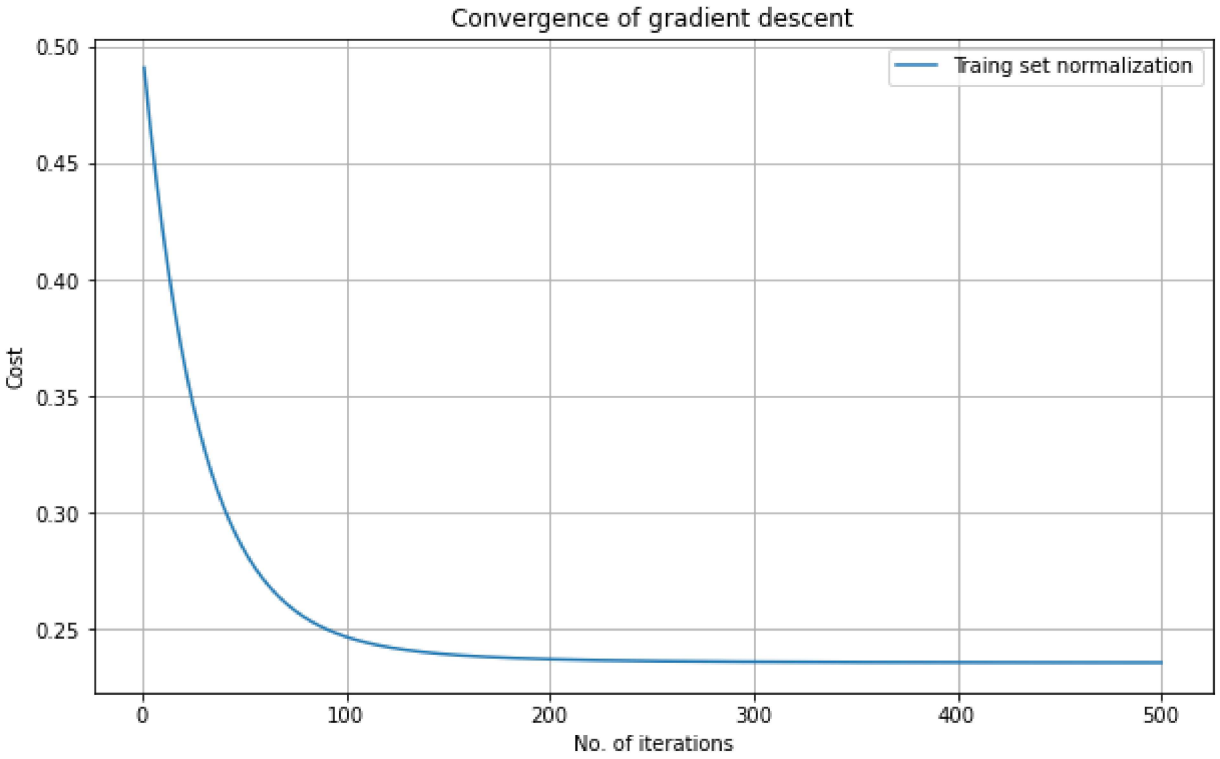
```
Final value of theta with standardization = [[1.29872054e-16 3.79061776e-01 1.10230128e-
01 2.94608603e-01
  2.32422016e-01 1.53858866e-01]]
```

In [19]:
```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta of the test set =', theta_Test)
cost_Test = list(cost_Test)
nIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta of the test set = [[3896885.81334708  798864.59108174  151510.77459
081 1093108.39710527
   870883.11233557  848681.31817011]]
```

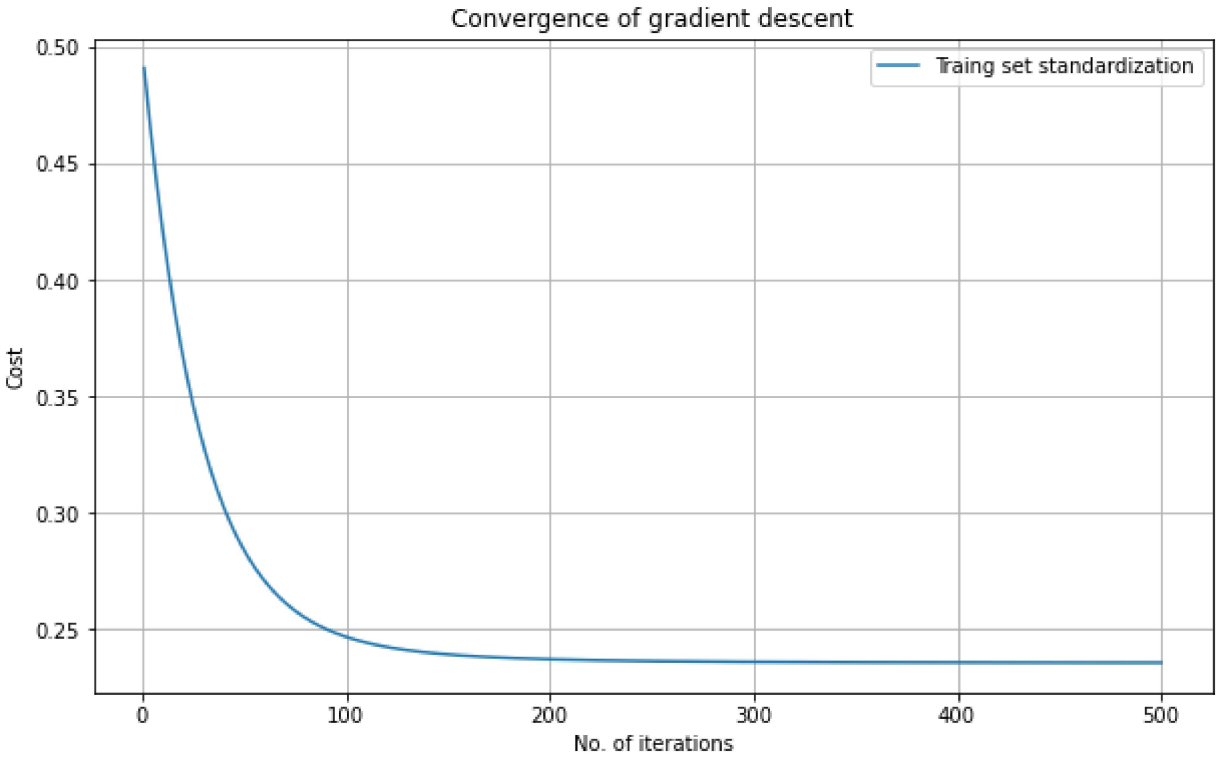
In [23]:
```
plt.plot(nIterations_Training, CostTrain, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[23]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



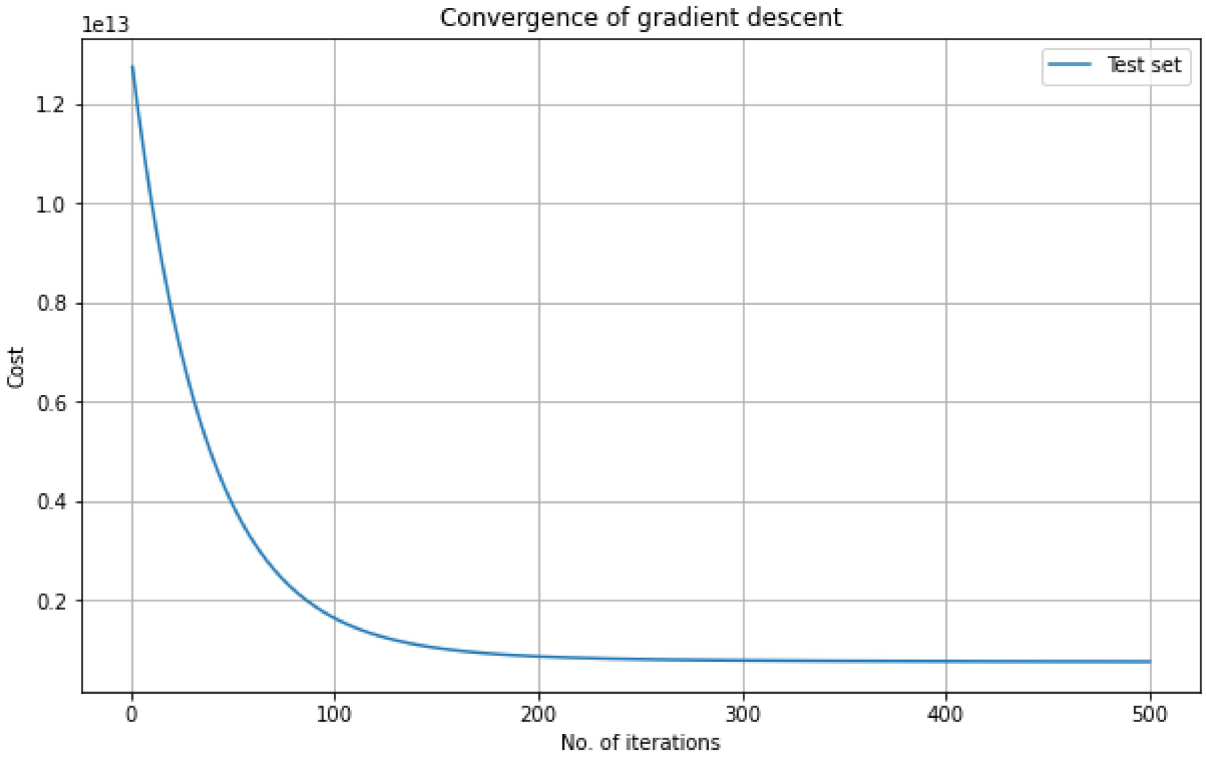
In [24]:
```
plt.plot(nIterations_Training2, CostTrain2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[24]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [25]:
```
plt.plot(nIterations_Test, cost_Test, label='Test set')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[25]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]:

In [1]:
```
#Ryan Picariello - 822856548 - Intro to ML Homework 1 part 2b
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [3]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[3]:

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

In [4]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[4]:

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

In [5]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [6]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'mainroad', 'guestroom', 'basement', 'hotw
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Normalization = df_Newtrain
df_Standardization = df_Newtrain
df_Newtrain.head()
```

Out[6]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning
454	4500	3	1	1	0	0	0	1
392	3990	3	1	1	0	0	0	0
231	4320	3	1	1	0	0	0	0
271	1905	5	1	0	0	1	0	0
250	3510	3	1	1	0	0	0	0

In [7]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_Normalization[num_vars] = scaler.fit_transform(df_Normalization[num_vars])
df_Normalization.head(20)
```

Out[7]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditionir
454	0.193548	0.50	0.0	1.0	0.0	0.0	0.0	1
392	0.156495	0.50	0.0	1.0	0.0	0.0	0.0	0
231	0.180471	0.50	0.0	1.0	0.0	0.0	0.0	0
271	0.005013	1.00	0.0	0.0	0.0	1.0	0.0	0
250	0.121622	0.50	0.0	1.0	0.0	0.0	0.0	0
541	0.040976	0.50	0.0	0.0	0.0	0.0	0.0	0
461	0.226969	0.25	0.0	1.0	0.0	1.0	0.0	1
124	0.340671	0.50	0.5	1.0	0.0	0.0	0.0	0
154	0.131793	0.50	0.5	1.0	0.0	0.0	0.0	0
451	0.357018	0.25	0.0	1.0	0.0	0.0	0.0	0
59	0.302528	0.50	0.5	1.0	1.0	0.0	0.0	1
493	0.154316	0.50	0.0	1.0	0.0	0.0	0.0	0
465	0.142691	0.25	0.0	1.0	0.0	0.0	0.0	0
490	0.182650	0.50	0.0	0.0	0.0	0.0	1.0	0

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditionir
540	0.084568	0.25	0.0	1.0	0.0	1.0	0.0	0
406	0.253124	0.25	0.0	1.0	0.0	0.0	0.0	0
289	0.291630	0.25	0.0	1.0	1.0	1.0	0.0	0
190	0.418774	0.75	0.0	1.0	0.0	0.0	0.0	1
55	0.302528	0.50	0.0	1.0	0.0	0.0	0.0	1
171	0.612685	0.50	0.0	1.0	0.0	0.0	0.0	0

In [8]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
scaler = StandardScaler()
#scaler = MinMaxScaler()
df_Standardization[num_vars] = scaler.fit_transform(df_Standardization[num_vars])
df_Standardization.head(20)
```

Out[8]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioni
454	-0.286366	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226
392	-0.544762	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
231	-0.377564	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
271	-1.601145	2.884176	-0.581230	-2.543735	-0.457738	1.405903	-0.216109	-0.7029
250	-0.787958	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
541	-1.350349	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	-0.216109	-0.7029
461	-0.053303	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	1.4226
124	0.739618	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
154	-0.717026	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
451	0.853616	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
59	0.473622	0.073764	1.488383	0.393123	2.184657	-0.711287	-0.216109	1.4226
493	-0.559962	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
465	-0.641027	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
490	-0.362365	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	4.627285	-0.7029
540	-1.046354	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	-0.7029
406	0.129094	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
289	0.397623	-1.331442	-0.581230	0.393123	2.184657	1.405903	-0.216109	-0.7029
190	1.284276	1.478970	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226
55	0.473622	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioni
171	2.636548	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029

In [9]:
```
XTrain_N = df_Normalization.values[:,0:10]
YTrain_N = df_Normalization.values[:,10]

XTest = df_Newtest.values[:,0:10]
YTest = df_Newtest.values[:,10]

XTrain_S = df_Standardization.values[:,0:10]
YTrain_S = df_Standardization.values[:,10]
```

In [10]:
```
mean = np.ones(XTrain_N.shape[1])
std = np.ones(XTrain_N.shape[1])
for i in range(0, XTrain_N.shape[1]):
    mean[i] = np.mean(XTrain_N.transpose()[i])
    std[i] = np.std(XTrain_N.transpose()[i])
    for j in range(0, XTrain_N.shape[0]):
        XTrain_N[j][i] = (XTrain_N[j][i] - mean[i])/std[i]
```

In [11]:
```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [12]:
```
mean = np.ones(XTrain_S.shape[1])
std = np.ones(XTrain_S.shape[1])
for i in range(0, XTrain_S.shape[1]):
    mean[i] = np.mean(XTrain_S.transpose()[i])
    std[i] = np.std(XTrain_S.transpose()[i])
    for j in range(0, XTrain_S.shape[0]):
        XTrain_S[j][i] = (XTrain_S[j][i] - mean[i])/std[i]
```

In [13]:
```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [14]:
```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
```

```
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [15]:
```
def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [23]:
```
iterations = 500;
alpha = 0.01;
```

In [24]:
```
ThetaTrain, CostTrain = linear_regression(XTrain_N, YTrain_N, alpha, iterations)
print('Final value of theta with normalization =', ThetaTrain)
CostTrain = list(CostTrain)
NIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.25821634e-16 2.61568196e-01 1.31838088e-01
2.84928147e-01
  1.20892351e-01 1.00586412e-01 3.92265249e-02 1.40343579e-01
  2.67270954e-01 9.57629183e-02 1.68134175e-01]]
```

In [25]:
```
ThetaTrain2, CostTrain2 = linear_regression(XTrain_S, YTrain_S, alpha, iterations)
print('Final value of theta with standardization  =', ThetaTrain2)
CostTrain2 = list(CostTrain2)
NIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

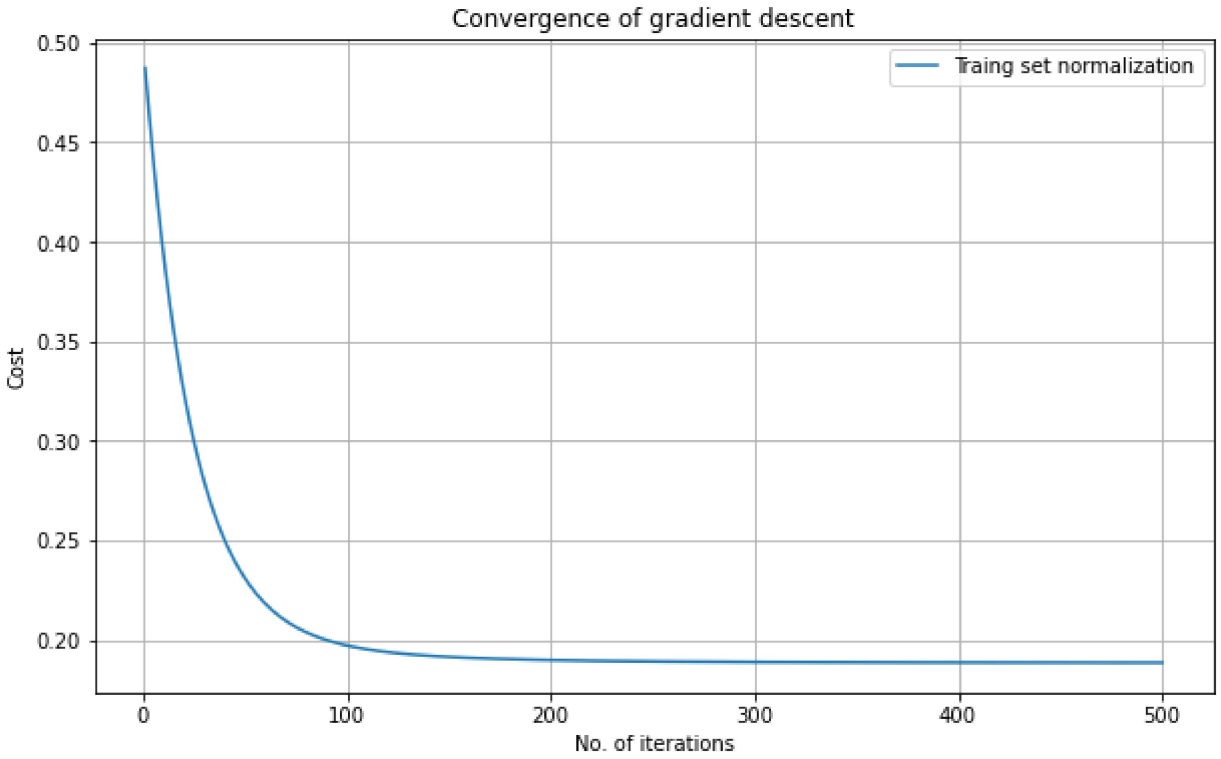
```
Final value of theta with standardization  = [[1.25821634e-16 2.61568196e-01 1.31838088e
-01 2.84928147e-01
  1.20892351e-01 1.00586412e-01 3.92265249e-02 1.40343579e-01
  2.67270954e-01 9.57629183e-02 1.68134175e-01]]
```

In [26]:
```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta =', theta_Test)
cost_Test = list(cost_Test)
NIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta = [[3211733.75281949  791345.02851107  162838.95231366 1164613.4908
8194
    51779.18076014  239993.96046932  566736.95476624  137642.82043416
  1204854.90843431  783960.095531    689075.93949973]]
```

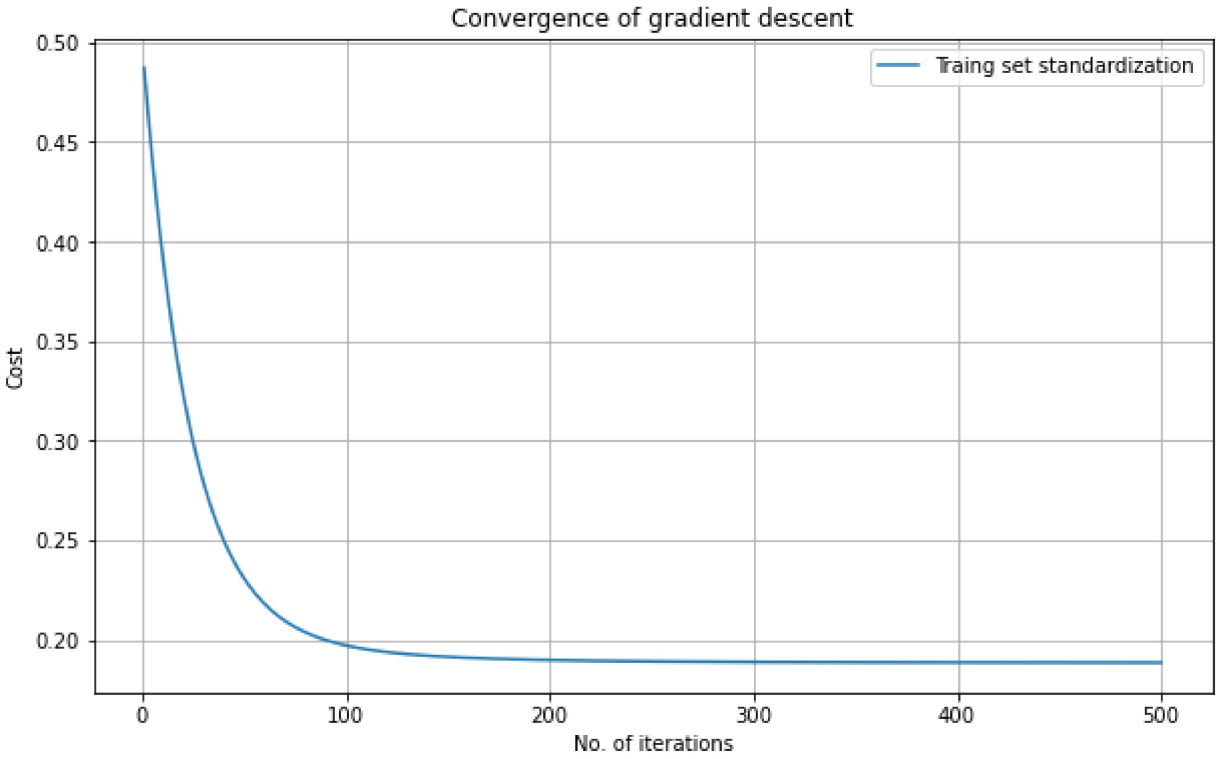
In [27]:
```
plt.plot(NIterations_Training, CostTrain, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[27]: Text(0.5, 1.0, 'Convergence of gradient descent')



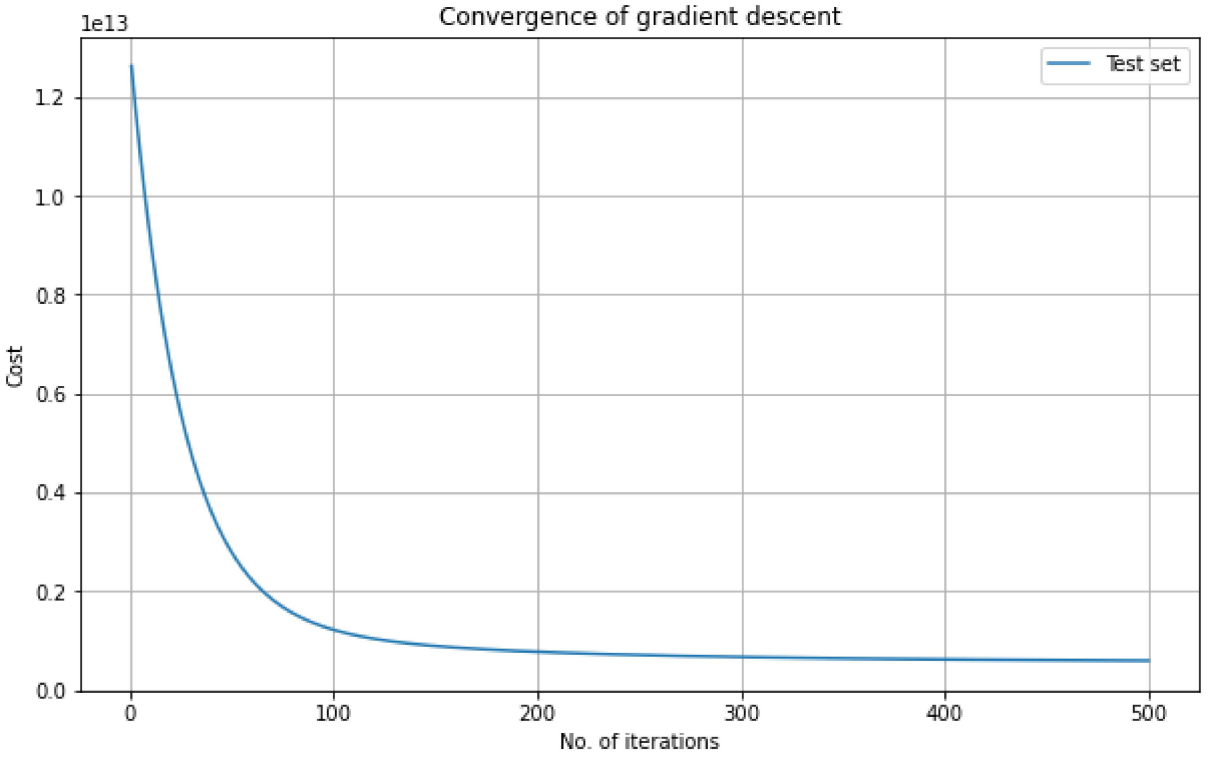
In [21]:
```
plt.plot(NIterations_Training2, CostTrain2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[21]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [22]:
```
plt.plot(NIterations_Test, cost_Test, label='Test set')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[22]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



In [1]:
```
# Ryan Picariello - 800856548 - Intro to ML Homework 1 Part 3a
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [3]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[3]:

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

In [4]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[4]:

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

In [5]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [6]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking','price']
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Normalization = df_Newtrain
df_Standardization  = df_Newtrain
df_Newtrain.head()
```

Out[6]:

	area	bedrooms	bathrooms	stories	parking	price
454	4500	3	1	2	0	3143000
392	3990	3	1	2	0	3500000
231	4320	3	1	1	0	4690000
271	1905	5	1	2	0	4340000
250	3510	3	1	3	0	4515000

In [7]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_Normalization[num_vars] = scaler.fit_transform(df_Normalization[num_vars])
df_Normalization.head(20)
```

Out[7]:

	area	bedrooms	bathrooms	stories	parking	price
454	0.193548	0.50	0.0	0.333333	0.000000	0.120606
392	0.156495	0.50	0.0	0.333333	0.000000	0.151515
231	0.180471	0.50	0.0	0.000000	0.000000	0.254545
271	0.005013	1.00	0.0	0.333333	0.000000	0.224242
250	0.121622	0.50	0.0	0.666667	0.000000	0.239394
541	0.040976	0.50	0.0	0.000000	0.000000	0.001485
461	0.226969	0.25	0.0	0.000000	0.000000	0.115152
124	0.340671	0.50	0.5	1.000000	0.333333	0.363636
154	0.131793	0.50	0.5	0.333333	0.666667	0.327273
451	0.357018	0.25	0.0	0.000000	0.000000	0.121212
59	0.302528	0.50	0.5	1.000000	0.333333	0.472727
493	0.154316	0.50	0.0	0.000000	0.000000	0.090909
465	0.142691	0.25	0.0	0.000000	0.000000	0.112121
490	0.182650	0.50	0.0	0.333333	0.333333	0.093939

	area	bedrooms	bathrooms	stories	parking	price
540	0.084568	0.25	0.0	0.000000	0.666667	0.006061
406	0.253124	0.25	0.0	0.000000	0.333333	0.148485
289	0.291630	0.25	0.0	0.000000	0.666667	0.212121
190	0.418774	0.75	0.0	0.333333	0.666667	0.284848
55	0.302528	0.50	0.0	0.333333	0.333333	0.484848
171	0.612685	0.50	0.0	0.000000	0.333333	0.303030

In [8]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = StandardScaler()
df_Standardization[num_vars] = scaler.fit_transform(df_Standardization[num_vars])
df_Standardization.head(20)
```

Out[8]:

	area	bedrooms	bathrooms	stories	parking	price
454	-0.286366	0.073764	-0.581230	0.207401	-0.822960	-0.868394
392	-0.544762	0.073764	-0.581230	0.207401	-0.822960	-0.677628
231	-0.377564	0.073764	-0.581230	-0.937813	-0.822960	-0.041744
271	-1.601145	2.884176	-0.581230	0.207401	-0.822960	-0.228768
250	-0.787958	0.073764	-0.581230	1.352614	-0.822960	-0.135256
541	-1.350349	0.073764	-0.581230	-0.937813	-0.822960	-1.603589
461	-0.053303	-1.331442	-0.581230	-0.937813	-0.822960	-0.902058
124	0.739618	0.073764	1.488383	2.497828	0.321375	0.631546
154	-0.717026	0.073764	1.488383	0.207401	1.465710	0.407116
451	0.853616	-1.331442	-0.581230	-0.937813	-0.822960	-0.864653
59	0.473622	0.073764	1.488383	2.497828	0.321375	1.304836
493	-0.559962	0.073764	-0.581230	-0.937813	-0.822960	-1.051678
465	-0.641027	-1.331442	-0.581230	-0.937813	-0.822960	-0.920761
490	-0.362365	0.073764	-0.581230	0.207401	0.321375	-1.032976
540	-1.046354	-1.331442	-0.581230	-0.937813	1.465710	-1.575348
406	0.129094	-1.331442	-0.581230	-0.937813	0.321375	-0.696331
289	0.397623	-1.331442	-0.581230	-0.937813	1.465710	-0.303578
190	1.284276	1.478970	-0.581230	0.207401	1.465710	0.145281
55	0.473622	0.073764	-0.581230	0.207401	0.321375	1.379646
171	2.636548	0.073764	-0.581230	-0.937813	0.321375	0.257496

In [9]:
```
XTrain_N = df_Normalization.values[:,[0,1,2,3,4]]
YTrain_N = df_Normalization.values[:,5]

XTest = df_Newtest.values[:,[0,1,2,3,4]]
YTest = df_Newtest.values[:,5]

XTrain_S = df_Standardization.values[:,[0,1,2,3,4]]
YTrain_S = df_Standardization.values[:,5]
```

In [10]:
```
mean = np.ones(XTrain_N.shape[1])
std = np.ones(XTrain_N.shape[1])
for i in range(0, XTrain_N.shape[1]):
    mean[i] = np.mean(XTrain_N.transpose()[i])
    std[i] = np.std(XTrain_N.transpose()[i])
    for j in range(0, XTrain_N.shape[0]):
        XTrain_N[j][i] = (XTrain_N[j][i] - mean[i])/std[i]
```

In [11]:
```
mean = np.ones(XTrain_S.shape[1])
std = np.ones(XTrain_S.shape[1])
for i in range(0, XTrain_S.shape[1]):
    mean[i] = np.mean(XTrain_S.transpose()[i])
    std[i] = np.std(XTrain_S.transpose()[i])
    for j in range(0, XTrain_S.shape[0]):
        XTrain_S[j][i] = (XTrain_S[j][i] - mean[i])/std[i]
```

In [12]:
```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [13]:
```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [20]:
```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    lam= 5000
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = (theta[j]*(1-(alpha*(lam/X.shape[0])))) - (alpha/X.shape[0]) * s
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [21]:
```
def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [22]:
```
iterations = 500;
alpha = 0.01;
```

In [23]:
```
ThetaTrain, CostTrain = linear_regression(XTrain_N, YTrain_N, alpha, iterations)
print('Final value of theta with normalization =', ThetaTrain)
CostTrain = list(CostTrain)
nIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.45777237e-16 3.48009818e-02 2.44727614e-02
3.27690196e-02
  2.68501832e-02 2.36723160e-02]]
```

In [24]:
```
ThetaTrain2, CostTrain2 = linear_regression(XTrain_S, YTrain_S, alpha, iterations)
print('Final value of theta with standardization =', ThetaTrain2)
CostTrain2 = list(CostTrain2)
nIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

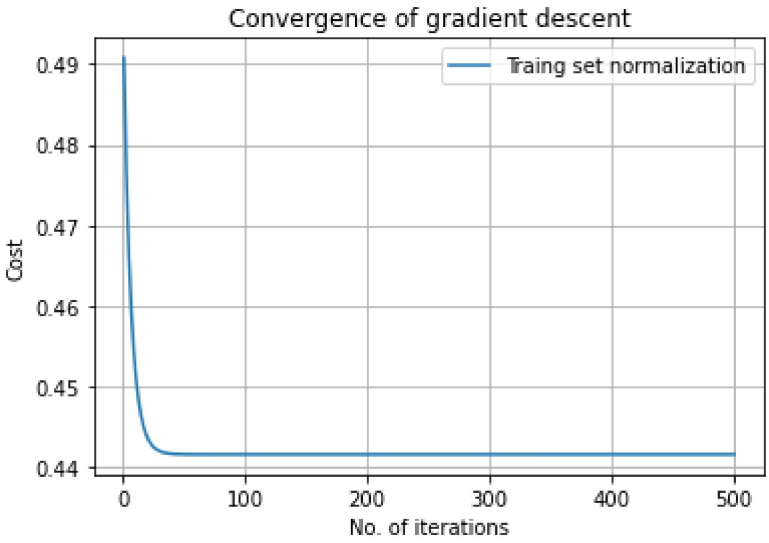
```
Final value of theta with standardization = [[1.45777237e-16 3.48009818e-02 2.44727614e-
02 3.27690196e-02
  2.68501832e-02 2.36723160e-02]]
```

In [25]:
```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta of the test set =', theta_Test)
cost_Test = list(cost_Test)
nIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta of the test set = [[4715355.79822782   20504.02734063   16283.46187
037   23092.80303923
    15597.1540135    11461.89955231]]
```

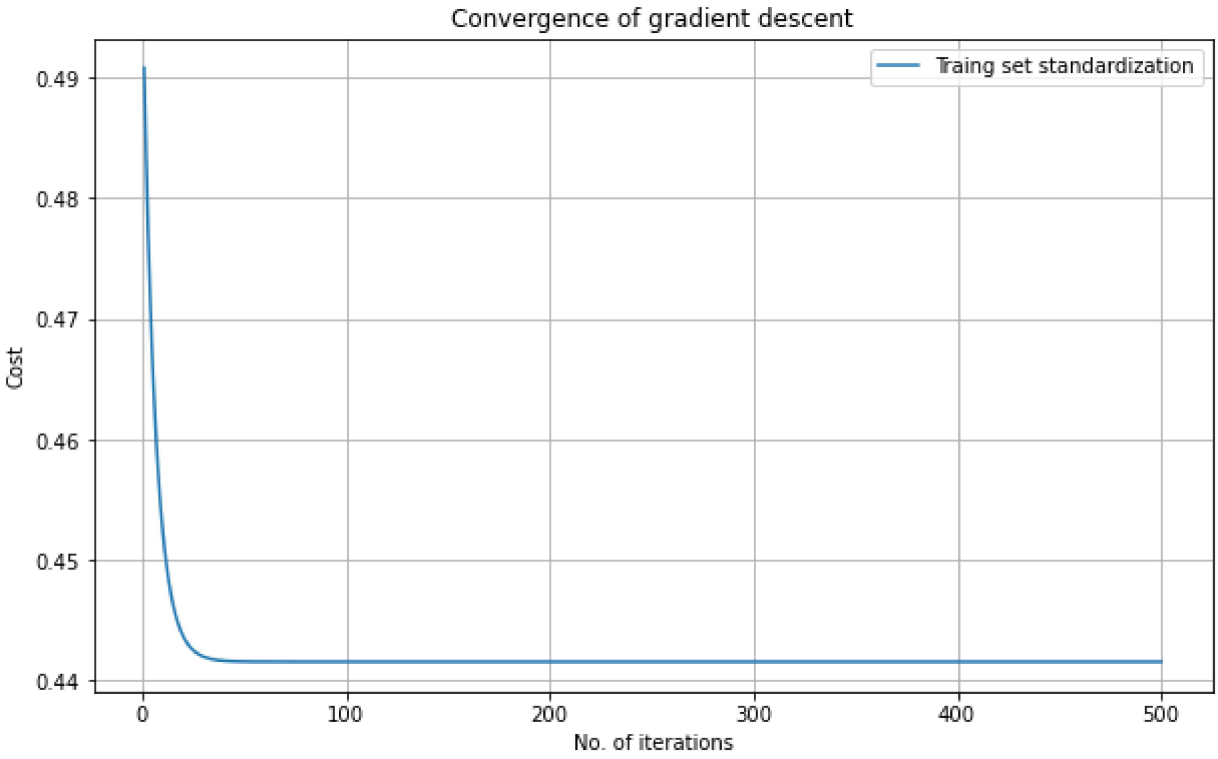
In [26]:
```
plt.plot(nIterations_Training, CostTrain, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[26]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



In [27]:
```
plt.plot(nIterations_Training2, CostTrain2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

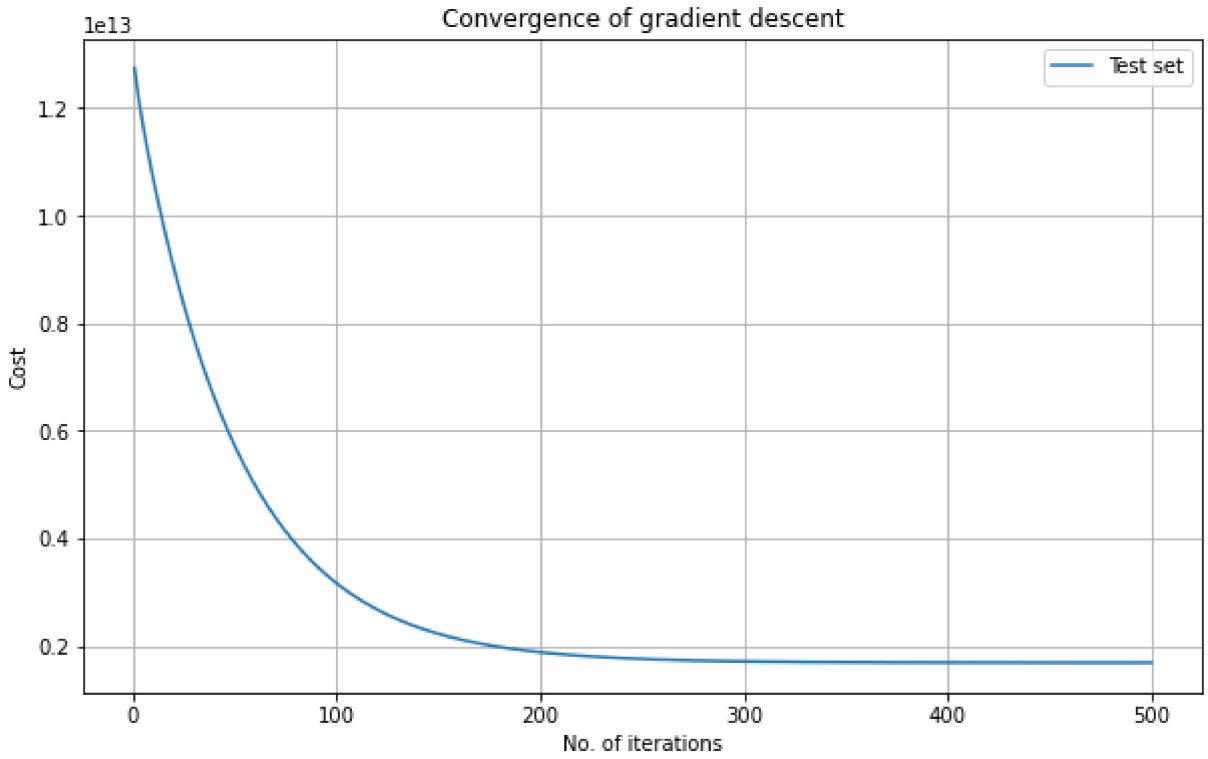
Out[27]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [28]:
```
plt.plot(nIterations_Test, cost_Test, label='Test set')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
```

```
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[28]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]:

In [ ]:

In [1]:
```
#Ryan Picariello - 822856548 - Intro to ML Homework 1 part 3b
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [3]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()
```

Out[3]:

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

In [4]:
```
# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Out[4]:

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

In [5]:
```
#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
```

```
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random
```

In [6]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'mainroad', 'guestroom', 'basement', 'hotw
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Normalization = df_Newtrain
df_Standardization = df_Newtrain
df_Newtrain.head()
```

Out[6]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning
454	4500	3	1	1	0	0	0	1
392	3990	3	1	1	0	0	0	0
231	4320	3	1	1	0	0	0	0
271	1905	5	1	0	0	1	0	0
250	3510	3	1	1	0	0	0	0

In [7]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_Normalization[num_vars] = scaler.fit_transform(df_Normalization[num_vars])
df_Normalization.head(20)
```

Out[7]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditionir
454	0.193548	0.50	0.0	1.0	0.0	0.0	0.0	1
392	0.156495	0.50	0.0	1.0	0.0	0.0	0.0	0
231	0.180471	0.50	0.0	1.0	0.0	0.0	0.0	0
271	0.005013	1.00	0.0	0.0	0.0	1.0	0.0	0
250	0.121622	0.50	0.0	1.0	0.0	0.0	0.0	0
541	0.040976	0.50	0.0	0.0	0.0	0.0	0.0	0
461	0.226969	0.25	0.0	1.0	0.0	1.0	0.0	1
124	0.340671	0.50	0.5	1.0	0.0	0.0	0.0	0
154	0.131793	0.50	0.5	1.0	0.0	0.0	0.0	0
451	0.357018	0.25	0.0	1.0	0.0	0.0	0.0	0
59	0.302528	0.50	0.5	1.0	1.0	0.0	0.0	1
493	0.154316	0.50	0.0	1.0	0.0	0.0	0.0	0
465	0.142691	0.25	0.0	1.0	0.0	0.0	0.0	0
490	0.182650	0.50	0.0	0.0	0.0	0.0	1.0	0

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditionir
540	0.084568	0.25	0.0	1.0	0.0	1.0	0.0	0
406	0.253124	0.25	0.0	1.0	0.0	0.0	0.0	0
289	0.291630	0.25	0.0	1.0	1.0	1.0	0.0	0
190	0.418774	0.75	0.0	1.0	0.0	0.0	0.0	1
55	0.302528	0.50	0.0	1.0	0.0	0.0	0.0	1
171	0.612685	0.50	0.0	1.0	0.0	0.0	0.0	0

In [8]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
scaler = StandardScaler()
#scaler = MinMaxScaler()
df_Standardization[num_vars] = scaler.fit_transform(df_Standardization[num_vars])
df_Standardization.head(20)
```

Out[8]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioni
454	-0.286366	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226
392	-0.544762	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
231	-0.377564	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
271	-1.601145	2.884176	-0.581230	-2.543735	-0.457738	1.405903	-0.216109	-0.7029
250	-0.787958	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
541	-1.350349	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	-0.216109	-0.7029
461	-0.053303	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	1.4226
124	0.739618	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
154	-0.717026	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
451	0.853616	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
59	0.473622	0.073764	1.488383	0.393123	2.184657	-0.711287	-0.216109	1.4226
493	-0.559962	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
465	-0.641027	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
490	-0.362365	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	4.627285	-0.7029
540	-1.046354	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	-0.7029
406	0.129094	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029
289	0.397623	-1.331442	-0.581230	0.393123	2.184657	1.405903	-0.216109	-0.7029
190	1.284276	1.478970	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226
55	0.473622	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.4226

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioni
171	2.636548	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.7029

In [9]:
```
XTrain_N = df_Normalization.values[:,0:10]
YTrain_N = df_Normalization.values[:,10]

XTest = df_Newtest.values[:,0:10]
YTest = df_Newtest.values[:,10]

XTrain_S = df_Standardization.values[:,0:10]
YTrain_S = df_Standardization.values[:,10]
```

In [10]:
```
mean = np.ones(XTrain_N.shape[1])
std = np.ones(XTrain_N.shape[1])
for i in range(0, XTrain_N.shape[1]):
    mean[i] = np.mean(XTrain_N.transpose()[i])
    std[i] = np.std(XTrain_N.transpose()[i])
    for j in range(0, XTrain_N.shape[0]):
        XTrain_N[j][i] = (XTrain_N[j][i] - mean[i])/std[i]
```

In [11]:
```
mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]
```

In [12]:
```
mean = np.ones(XTrain_S.shape[1])
std = np.ones(XTrain_S.shape[1])
for i in range(0, XTrain_S.shape[1]):
    mean[i] = np.mean(XTrain_S.transpose()[i])
    std[i] = np.std(XTrain_S.transpose()[i])
    for j in range(0, XTrain_S.shape[0]):
        XTrain_S[j][i] = (XTrain_S[j][i] - mean[i])/std[i]
```

In [13]:
```
def compute_cost(X, n, theta):
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h
```

In [14]:
```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    lam= 5000
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
```

```
            theta[j] = (theta[j]*(1-(alpha*(lam/X.shape[0])))) - (alpha/X.shape[0]) * s
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost
```

In [15]:
```
def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost
```

In [16]:
```
iterations = 500;
alpha = 0.01;
```

In [17]:
```
ThetaTrain, CostTrain = linear_regression(XTrain_N, YTrain_N, alpha, iterations)
print('Final value of theta with normalization =', ThetaTrain)
CostTrain = list(CostTrain)
NIterations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.42204344e-16 3.34497938e-02 2.44254794e-02
3.24447841e-02
  1.76107462e-02 1.76762851e-02 1.33464743e-02 9.45805497e-03
  2.87532615e-02 2.28368271e-02 2.40009491e-02]]
```

In [18]:
```
ThetaTrain2, CostTrain2 = linear_regression(XTrain_S, YTrain_S, alpha, iterations)
print('Final value of theta with standardization  =', ThetaTrain2)
CostTrain2 = list(CostTrain2)
NIterations_Training2 = [x for x in range(1,(iterations + 1))]
```

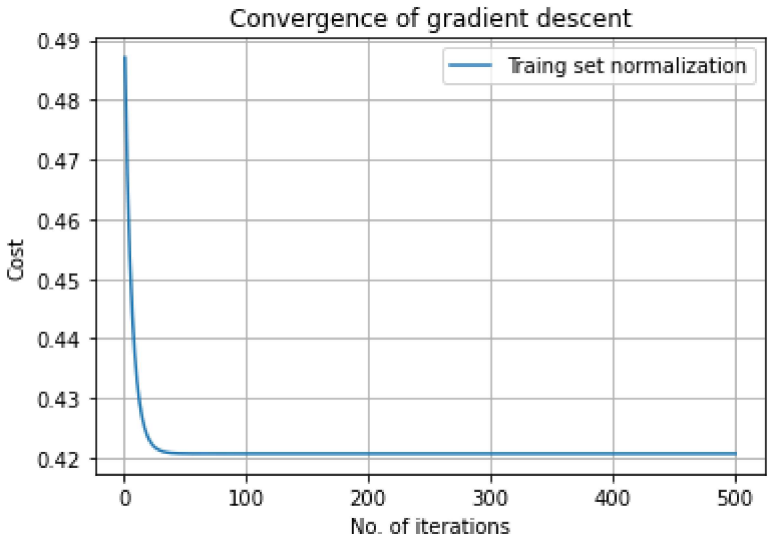
```
Final value of theta with standardization  = [[1.42204344e-16 3.34497938e-02 2.44254794e
-02 3.24447841e-02
  1.76107462e-02 1.76762851e-02 1.33464743e-02 9.45805497e-03
  2.87532615e-02 2.28368271e-02 2.40009491e-02]]
```

In [19]:
```
theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta =', theta_Test)
cost_Test = list(cost_Test)
NIterations_Test = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta = [[4.71044818e+06 2.04554726e+04 1.62895832e+04 2.31186933e+04
  1.49544590e+04 1.00880850e+04 4.08059345e+03 2.80712118e+02
  1.39671985e+04 1.14500716e+04 6.10239368e+03]]
```

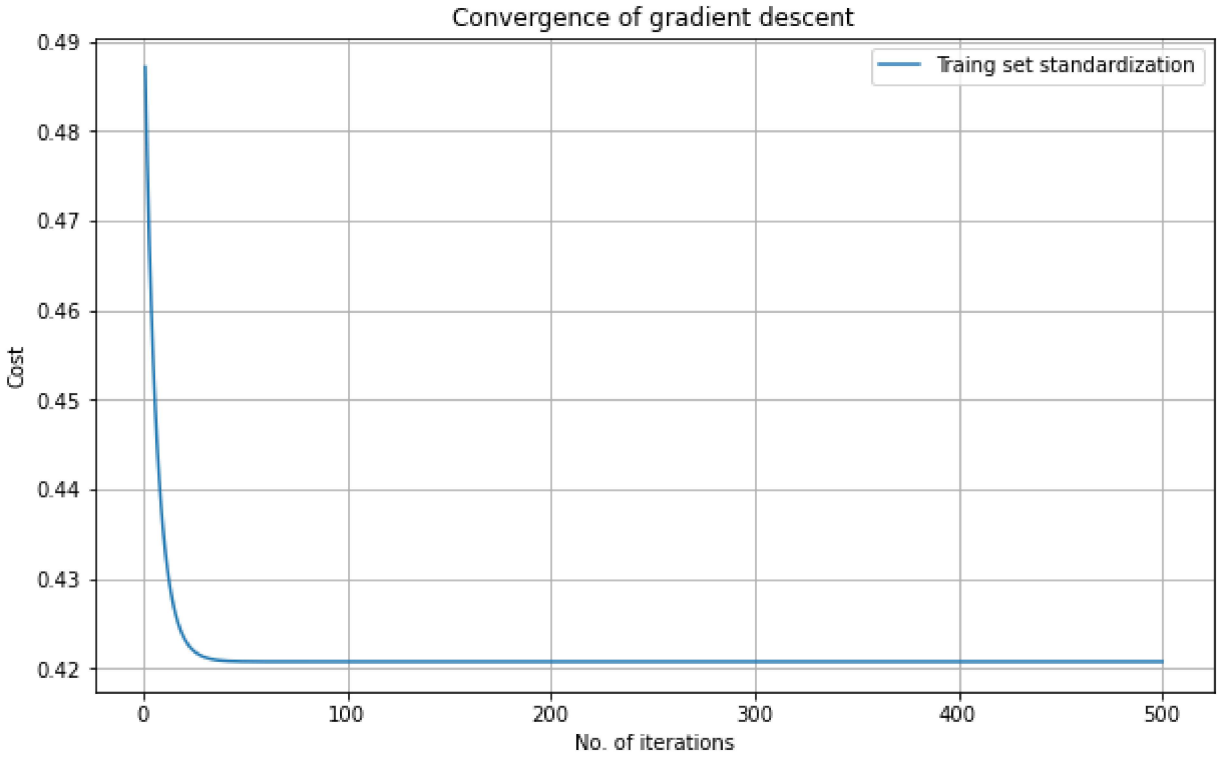
In [20]:
```
plt.plot(NIterations_Training, CostTrain, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[20]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [21]:
```
plt.plot(NIterations_Training2, CostTrain2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

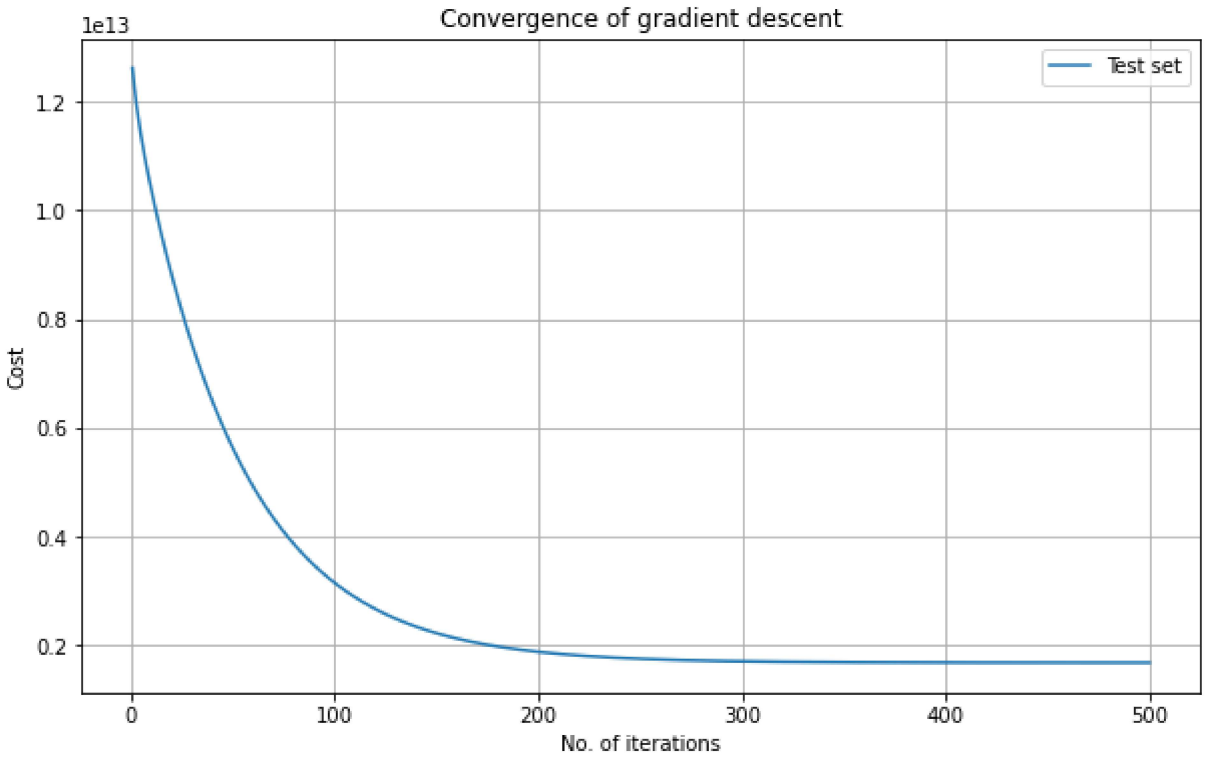
Out[21]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [22]:
```
plt.plot(NIterations_Test, cost_Test, label='Test set')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
```

```
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[22]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [23]:

```
#Ryan Picariello - 822856548 - Intro to ML Homework 1 part 2b
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)

housing = pd.DataFrame(pd.read_csv('C:/Users/Ryanj/Downloads/Housing.csv'))
housing.head()

# You can see that your dataset has many columns with values as 'Yes' or 'No'.
# But in order to fit a regression line, we would need numerical values and not string.
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', '
# Defining the map function
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()

#Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the same rows, respec
np.random.seed(0)
```

```
df_train, df_test = train_test_split(housing, train_size = 0.7, test_size = 0.3, random

num_vars = ['area', 'bedrooms', 'bathrooms', 'mainroad', 'guestroom', 'basement', 'hotw
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Normalization = df_Newtrain
df_Standardization = df_Newtrain
df_Newtrain.head()

import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
#scaler = StandardScaler()
scaler = MinMaxScaler()
df_Normalization[num_vars] = scaler.fit_transform(df_Normalization[num_vars])
df_Normalization.head(20)

import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# define standard scaler
scaler = StandardScaler()
#scaler = MinMaxScaler()
df_Standardization[num_vars] = scaler.fit_transform(df_Standardization[num_vars])
df_Standardization.head(20)

XTrain_N = df_Normalization.values[:,0:10]
YTrain_N = df_Normalization.values[:,10]

XTest = df_Newtest.values[:,0:10]
YTest = df_Newtest.values[:,10]

XTrain_S = df_Standardization.values[:,0:10]
YTrain_S = df_Standardization.values[:,10]

mean = np.ones(XTrain_N.shape[1])
std = np.ones(XTrain_N.shape[1])
for i in range(0, XTrain_N.shape[1]):
    mean[i] = np.mean(XTrain_N.transpose()[i])
    std[i] = np.std(XTrain_N.transpose()[i])
    for j in range(0, XTrain_N.shape[0]):
        XTrain_N[j][i] = (XTrain_N[j][i] - mean[i])/std[i]

mean = np.ones(XTest.shape[1])
std = np.ones(XTest.shape[1])
for i in range(0, XTest.shape[1]):
    mean[i] = np.mean(XTest.transpose()[i])
    std[i] = np.std(XTest.transpose()[i])
    for j in range(0, XTest.shape[0]):
        XTest[j][i] = (XTest[j][i] - mean[i])/std[i]

mean = np.ones(XTrain_S.shape[1])
std = np.ones(XTrain_S.shape[1])
for i in range(0, XTrain_S.shape[1]):
    mean[i] = np.mean(XTrain_S.transpose()[i])
    std[i] = np.std(XTrain_S.transpose()[i])
    for j in range(0, XTrain_S.shape[0]):
        XTrain_S[j][i] = (XTrain_S[j][i] - mean[i])/std[i]

def compute_cost(X, n, theta):
```

```
    h = np.ones((X.shape[0],1))
    theta = theta.reshape(1,n+1)
    for i in range(0,X.shape[0]):
        h[i] = float(np.matmul(theta, X[i]))
    h = h.reshape(X.shape[0])
    return h

def gradient_descent(X, y, theta, alpha, iterations, n, h):
    cost = np.ones(iterations)
    for i in range(0,iterations):
        theta[0] = theta[0] - (alpha/X.shape[0]) * sum(h - y)
        for j in range(1,n+1):
            theta[j] = theta[j] - (alpha/X.shape[0]) * sum((h-y) * X.transpose()[j])
        h = compute_cost(X, n, theta)
        cost[i] = (1/X.shape[0]) * 0.5 * sum(np.square(h - y))
    theta = theta.reshape(1,n+1)
    return theta, cost

def linear_regression(X, y, alpha, iterations):
    n = X.shape[1]
    one_column = np.ones((X.shape[0],1))
    X = np.concatenate((one_column, X), axis = 1)
    theta = np.zeros(n+1)
    h = compute_cost(X, n, theta)
    theta, cost = gradient_descent(X, y, theta, alpha, iterations, n, h)
    return theta, cost

iterations = 500;
alpha = 0.01;

ThetaTrain, CostTrain = linear_regression(XTrain_N, YTrain_N, alpha, iterations)
print('Final value of theta with normalization =', ThetaTrain)
CostTrain = list(CostTrain)
NIterations_Training = [x for x in range(1,(iterations + 1))]

ThetaTrain2, CostTrain2 = linear_regression(XTrain_S, YTrain_S, alpha, iterations)
print('Final value of theta with standardization  =', ThetaTrain2)
CostTrain2 = list(CostTrain2)
NIterations_Training2 = [x for x in range(1,(iterations + 1))]

theta_Test, cost_Test = linear_regression(XTest, YTest, alpha, iterations)
print('Final value of theta =', theta_Test)
cost_Test = list(cost_Test)
NIterations_Test = [x for x in range(1,(iterations + 1))]

plt.plot(NIterations_Training, CostTrain, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')

plt.plot(NIterations_Training2, CostTrain2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

```
plt.plot(NIterations_Test, cost_Test, label='Test set')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

```
Final value of theta with normalization = [[1.25821634e-16 2.61568196e-01 1.31838088e-01
2.84928147e-01
  1.20892351e-01 1.00586412e-01 3.92265249e-02 1.40343579e-01
  2.67270954e-01 9.57629183e-02 1.68134175e-01]]
Final value of theta with standardization  = [[1.25821634e-16 2.61568196e-01 1.31838088e
-01 2.84928147e-01
  1.20892351e-01 1.00586412e-01 3.92265249e-02 1.40343579e-01
  2.67270954e-01 9.57629183e-02 1.68134175e-01]]
Final value of theta = [[3211733.75281949  791345.02851107  162838.95231366 1164613.4908
8194
    51779.18076014  239993.96046932  566736.95476624  137642.82043416
  1204854.90843431  783960.095531    689075.93949973]]
```

Out[23]: Text(0.5, 1.0, 'Convergence of gradient descent')