In [23]:
```
# Kemp Carswell 801017179
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [24]:
```
df = pd.read_csv('C:/Users/kemp/Downloads/Housing.csv')
df.head() # To get first n rows from the dataset default value of n is 5
M=len(df)
```

In [25]:
```
housing = pd.DataFrame(pd.read_csv('C:/Users/kemp/Downloads/Housing.csv'))
housing.head()
```

Out[25]:

	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 [26]:
```
# 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[26]:

	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 [27]:
```
#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 [28]:

```
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[28]:

	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 [29]:

```
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[29]:

	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 [30]:

```
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[30]:

	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 [31]:

```
X_Training_N = df_Normalization.values[:,0:10]
y_Training_N = df_Normalization.values[:,10]

X_Test = df_Newtest.values[:,0:10]
y_Test = df_Newtest.values[:,10]

X_Training_S = df_Standardization.values[:,0:10]
y_Training_S = df_Standardization.values[:,10]
```

In [32]:

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

In [33]:

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

In [34]:

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

In [35]:

```
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 [36]:

```
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 [37]:

```
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 [45]:

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

In [46]:

```
theta_Training, cost_Training = linear_regression(X_Training_N, y_Training_N, alpha, it
print('Final value of theta with normalization =', theta_Training)
cost_Training = list(cost_Training)
n_ierations_Training = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.24798830e-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 [47]:

```
theta_Training2, cost_Training2 = linear_regression(X_Training_S, y_Training_S, alpha,
print('Final value of theta with standardization  =', theta_Training2)
cost_Training2 = list(cost_Training2)
n_ierations_Training2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with standardization  = [[1.24798830e-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 [48]:

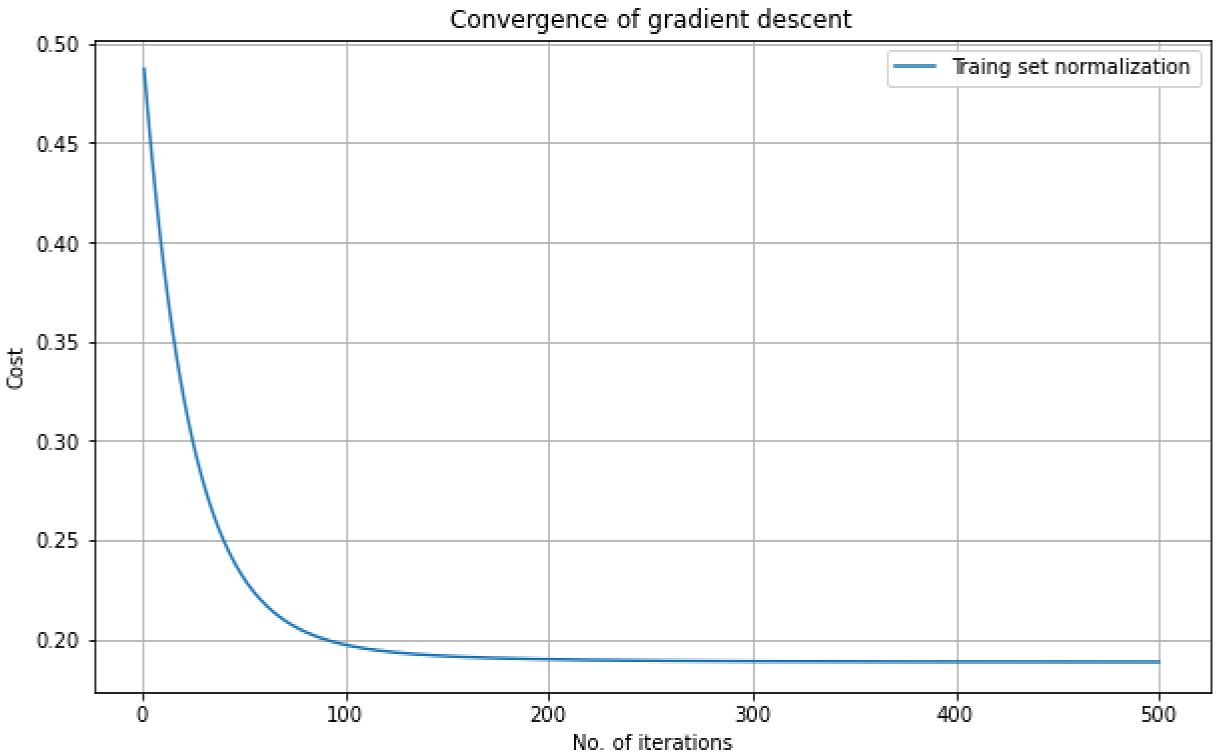
```
theta_Test, cost_Test = linear_regression(X_Test, y_Test, alpha, iterations)
print('Final value of theta =', theta_Test)
cost_Test = list(cost_Test)
n_ierations_Test = [x for x in range(1,(iterations + 1))]
```

```
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 [49]:

```
plt.plot(n_ierations_Training, cost_Training, 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')
```

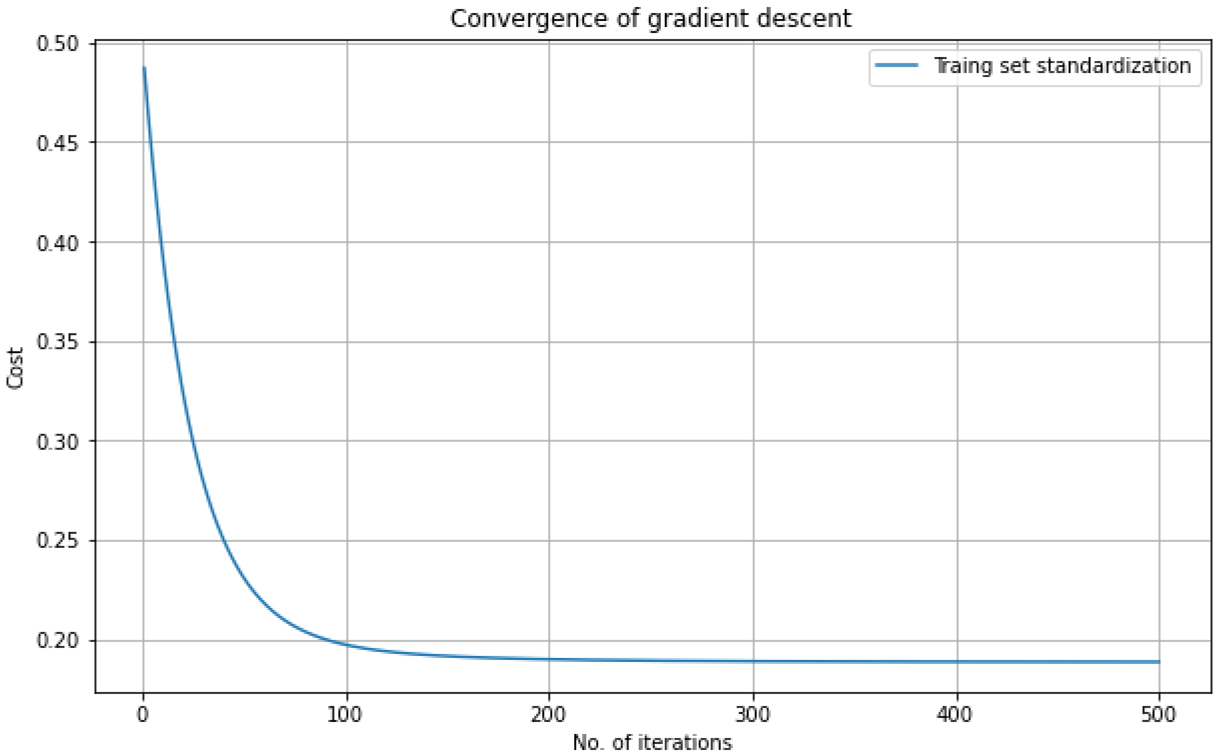
Out[49]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [50]:

```
plt.plot(n_ierations_Training2, cost_Training2, 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')
```

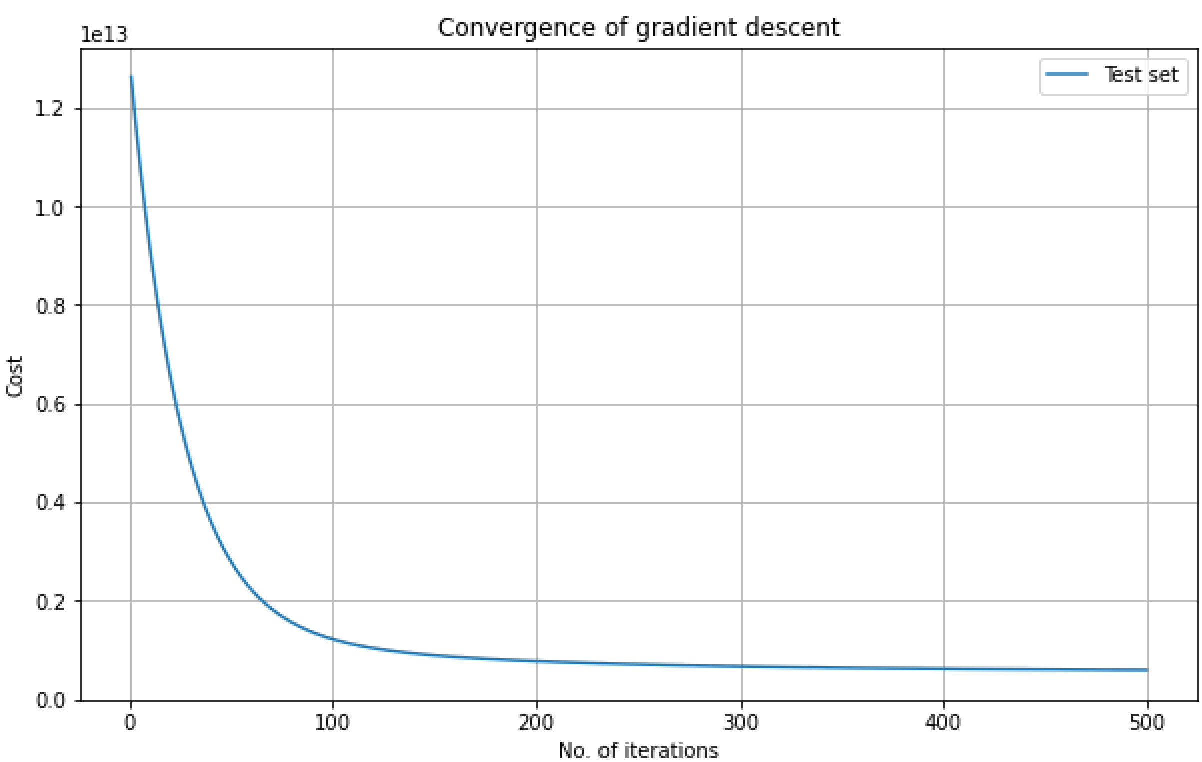
Out[50]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [51]:

```
plt.plot(n_ierations_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[51]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]: