In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
housing = pd.DataFrame(pd.read_csv("Housing.csv"))
housing.head()
```

Out[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

In [3]:

```
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes' : 1, "no" : 0})

housing[varlist] = housing[varlist].apply(binary_map)
housing.head()
```

Out[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

In [4]:

```
from sklearn.model_selection import train_test_split

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

Out[4]: (381, 13)

In [5]:

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

Out[5]:

	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 [6]:

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

Out[6]:

	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
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 [7]:

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

Out[7]:

	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 [8]:

```
X_n = df_Norm.values[:,[0,1,2,3,4]]
y_n = df_Norm.values[:,5]
X_t = df_Newtest.values[:,[0,1,2,3,4]]
y_t = df_Newtest.values[:,5]
X_s = df_Standard.values[:,[0,1,2,3,4]]
y_s = df_Standard.values[:,5]
```

In [9]:

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

In [10]:

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

In [11]:

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

In [12]:

```
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 [13]:

```
def gradient_descent(X, y, theta, alpha, iterations, n, h):
    lam = 1000
    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]*(1-(alpha*(lam/X.shape[0])))) - (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 [14]:

```
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 [15]:

```
iterations = 500;
theta, cost = linear_regression(X_n, y_n, 0.1, iterations)
print('Final value of theta with normalization =', theta)
cost = list(cost)
n_iterations = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.42056096e-16 1.24270894e-01 7.53179957e-02 1.10985844e-01
  9.01203112e-02 7.75127501e-02]]
```

In [16]:

```
theta2, cost2 = linear_regression(X_s, y_s, 0.1, iterations)
print('Final value of theta with standardization =', theta2)
cost = list(cost2)
n_iterations2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with standardization = [[1.42056096e-16 1.24270894e-01 7.53179957e-02 1.10985844e-01
  9.01203112e-02 7.75127501e-02]]
```

In [17]:

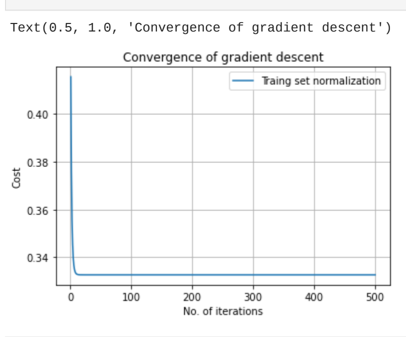
```
theta_t, cost_t = linear_regression(X_t, y_t, 0.1, iterations)
print('Final value of theta of the test set =', theta)
cost_t = list(cost_t)
n_iterations_t = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta of the test set = [[1.42056096e-16 1.24270894e-01 7.53179957e-02 1.10985844e-01
  9.01203112e-02 7.75127501e-02]]
```

In [18]:

```
plt.plot(n_iterations, cost, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

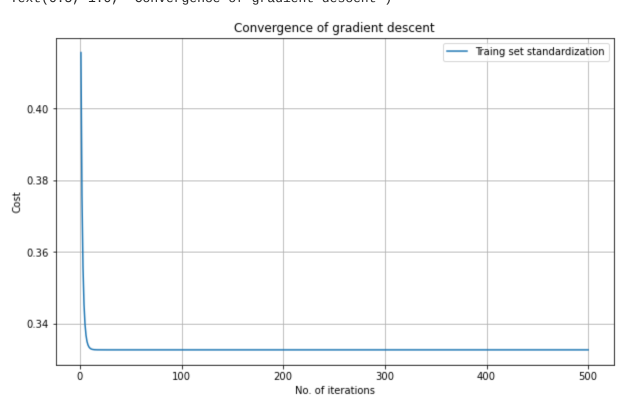
Out[18]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [19]:

```
plt.plot(n_iterations2, cost2, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

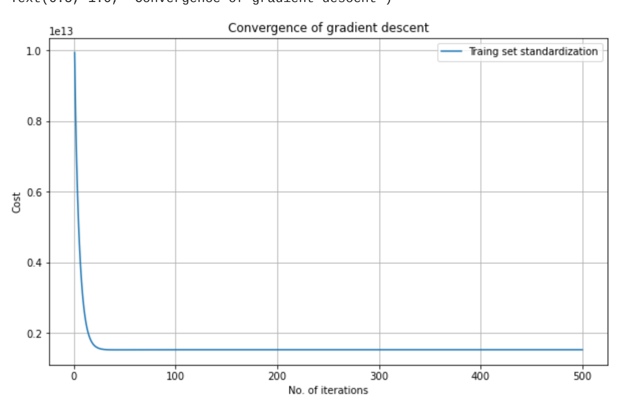
Out[19]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [20]:

```
plt.plot(n_iterations_t, cost_t, label='Traing set standardization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

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



In [21]:

```
num_vars = ['area', 'bedrooms', 'bathrooms', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea', 'price']
df_Newtrain = df_train[num_vars]
df_Newtest = df_test[num_vars]
df_Norm = df_Newtrain
df_Standard = df_Newtrain
df_Newtrain.head()
```

Out[21]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
454	4500	3	1	1	0	0	0	1	0	0	3143000
392	3990	3	1	1	0	0	0	0	0	0	3500000
231	4320	3	1	1	0	0	0	0	0	1	4690000
271	1905	5	1	0	0	1	0	0	0	0	4340000
250	3510	3	1	1	0	0	0	0	0	0	4515000

In [22]:

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

Out[22]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
454	0.193548	0.50	0.0	1.0	0.0	0.0	0.0	1.0	0.000000	0.0	0.120606
392	0.156495	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.151515
231	0.180471	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	1.0	0.254545
271	0.005013	1.00	0.0	0.0	0.0	1.0	0.0	0.0	0.000000	0.0	0.224242
250	0.121622	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.239394
541	0.040976	0.50	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.001485
461	0.226969	0.25	0.0	1.0	0.0	1.0	0.0	1.0	0.000000	0.0	0.115152
124	0.340671	0.50	0.5	1.0	0.0	0.0	0.0	0.0	0.333333	0.0	0.363636
154	0.131793	0.50	0.5	1.0	0.0	0.0	0.0	0.0	0.666667	0.0	0.327273
451	0.357018	0.25	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.121212
59	0.302528	0.50	0.5	1.0	1.0	0.0	0.0	1.0	0.333333	0.0	0.472727
493	0.154316	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.090909
465	0.142691	0.25	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.112121
490	0.182650	0.50	0.0	0.0	0.0	0.0	1.0	0.0	0.333333	0.0	0.093939
540	0.084568	0.25	0.0	1.0	0.0	1.0	0.0	0.0	0.666667	0.0	0.006061
406	0.253124	0.25	0.0	1.0	0.0	0.0	0.0	0.0	0.333333	1.0	0.148485
289	0.291630	0.25	0.0	1.0	1.0	1.0	0.0	0.0	0.666667	0.0	0.212121
190	0.418774	0.75	0.0	1.0	0.0	0.0	0.0	1.0	0.666667	0.0	0.284848
55	0.302528	0.50	0.0	1.0	0.0	0.0	0.0	1.0	0.333333	0.0	0.484848
171	0.612685	0.50	0.0	1.0	0.0	0.0	0.0	0.0	0.333333	1.0	0.303030

In [23]:

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

Out[23]:

	area	bedrooms	bathrooms	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
454	-0.286366	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.422607	-0.822960	-0.564215	-0.868394
392	-0.544762	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-0.677628
231	-0.377564	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	1.772373	-0.041744
271	-1.601145	2.884176	-0.581230	-2.543735	-0.457738	1.405903	-0.216109	-0.702935	-0.822960	-0.564215	-0.228768
250	-0.787958	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-0.135256
541	-1.350349	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-1.603589
461	-0.053303	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	1.422607	-0.822960	-0.564215	-0.902058
124	0.739618	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	0.321375	-0.564215	0.631546
154	-0.717026	0.073764	1.488383	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	1.465710	-0.564215	0.407116
451	0.853616	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-0.864653
59	0.473622	0.073764	1.488383	0.393123	2.184657	-0.711287	-0.216109	1.422607	0.321375	-0.564215	1.304836
493	-0.559962	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-1.051678
465	-0.641027	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	-0.822960	-0.564215	-0.920761
490	-0.362365	0.073764	-0.581230	-2.543735	-0.457738	-0.711287	4.627285	-0.702935	0.321375	-0.564215	-1.032976
540	-1.046354	-1.331442	-0.581230	0.393123	-0.457738	1.405903	-0.216109	-0.702935	1.465710	-0.564215	-1.575348
406	0.129094	-1.331442	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	0.321375	1.772373	-0.696331
289	0.397623	-1.331442	-0.581230	0.393123	2.184657	1.405903	-0.216109	-0.702935	1.465710	-0.564215	-0.303578
190	1.284276	1.478970	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.422607	1.465710	-0.564215	0.145281
55	0.473622	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	1.422607	0.321375	-0.564215	1.379646
171	2.636548	0.073764	-0.581230	0.393123	-0.457738	-0.711287	-0.216109	-0.702935	0.321375	1.772373	0.257496

In [24]:

```
X_n = df_Norm.values[:,0:10]
y_n = df_Norm.values[:,10]
X_t = df_Newtest.values[:,0:10]
y_t = df_Newtest.values[:,10]
X_s = df_Standard.values[:,0:10]
y_s = df_Standard.values[:,10]
```

In [25]:

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

In [26]:

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

In [27]:

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

In [28]:

```
theta, cost = linear_regression(X_n, y_n, 0.1, iterations)
print('Final value of theta with normalization =', theta)
cost = list(cost)
n_iterations = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with normalization = [[1.38369922e-16 1.08337017e-01 7.58521283e-02 1.08013107e-01
  5.54076634e-02 5.32500668e-02 3.66508358e-02 3.65154719e-02
  9.68953820e-02 6.81505402e-02 7.57869446e-02]]
```

In [29]:

```
theta2, cost2 = linear_regression(X_s, y_s, 0.1, iterations)
print('Final value of theta with standardization =', theta2)
cost2 = list(cost2)
n_iterations2 = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta with standardization = [[1.38369922e-16 1.08337017e-01 7.58521283e-02 1.08013107e-01
  5.54076634e-02 5.32500668e-02 3.66508358e-02 3.65154719e-02
  9.68953820e-02 6.81505402e-02 7.57869446e-02]]
```

In [30]:

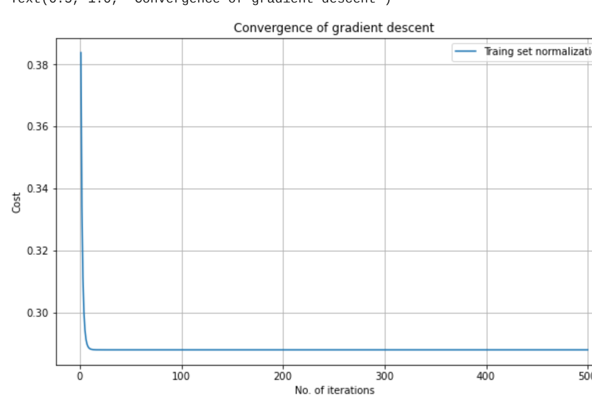
```
theta_t, cost_t = linear_regression(X_Test, y_Test, 0.1, iterations)
print('Final value of theta of the test set =', theta_t)
cost_t = list(cost_t)
n_iterations_t = [x for x in range(1,(iterations + 1))]
```

```
Final value of theta of the test set = [[4.68135283e+06 9.14174376e+04 7.10541172e+04 1.03091554e+05
  6.81362933e+04 4.24713613e+04 1.63927974e+04 8.78780977e+02
  6.38058529e+04 5.14196389e+04 2.74614616e+04]]
```

In [31]:

```
plt.plot(n_iterations, cost, label='Traing set normalization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

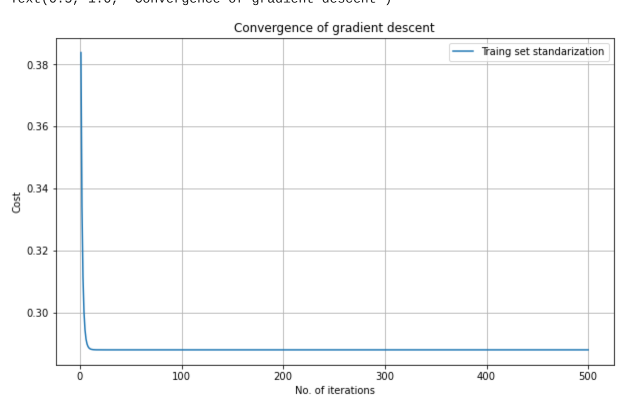
Out[31]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [32]:

```
plt.plot(n_iterations2, cost2, label='Traing set standarization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

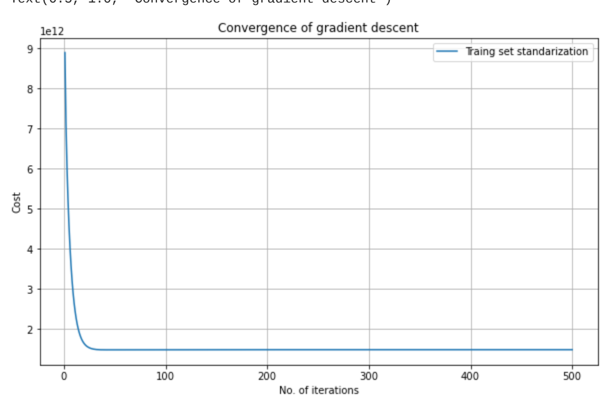
Out[32]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [33]:

```
plt.plot(n_iterations_t, cost_t, label='Traing set standarization')
plt.legend()
plt.rcParams["figure.figsize"]=(10,6)
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Convergence of gradient descent')
```

Out[33]: Text(0.5, 1.0, 'Convergence of gradient descent')



In [ ]: