In [158…

```
# Kemp Carswell 801017179
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [159…

```
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 [160…

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

Out[160…

	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 [161…

```
# 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[161…

	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 [162…

```
#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 [163…

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

Out[163…

	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 [164…

```
X_Training = df_Newtrain.values[:,[0,1,2,3,4]]
y_Training = df_Newtrain.values[:,5]

X_Test = df_Newtest.values[:,[0,1,2,3,4]]
y_Test = df_Newtest.values[:,5]
```

In [165…

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

In [166…

```
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 [167…

```
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 [168…

```
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 [169…

```
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 [170…

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

In [171…

```
theta_Training, cost_Training = linear_regression(X_Training, y_Training, alpha, iterat
print('Final value of theta with an alpha of 0.1 =', theta_Training)
cost_Training = list(cost_Training)
n_ierations_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 [172…

```
theta_Training2, cost_Training2 = linear_regression(X_Training, y_Training, alpha2, ite
print('Final value of theta with an alpha of 0.01 =', theta_Training2)
cost_Training2 = list(cost_Training2)
n_ierations_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 [173…

```
theta_Test, cost_Test = linear_regression(X_Test, y_Test, alpha, iterations)
print('Final value of theta with an alpha of 0.1 =', theta_Test)
cost_Test = list(cost_Test)
n_ierations_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 [174…

```
theta_Test2, cost_Test2 = linear_regression(X_Test, y_Test, alpha2, iterations)
print('Final value of theta with an alpha of 0.01 =', theta_Test2)
cost_Test2 = list(cost_Test2)
n_ierations_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
```

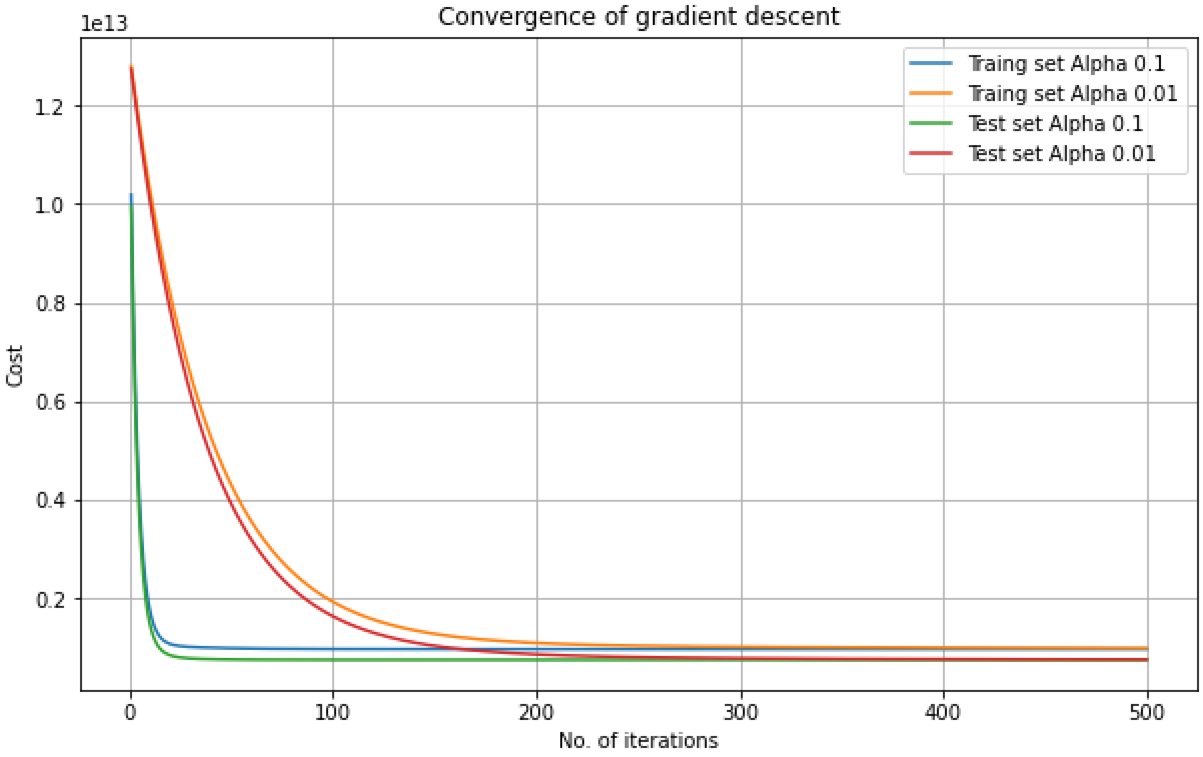
```
   870883.11233557  848681.31817011]]
```

In [175…

```
plt.plot(n_ierations_Training, cost_Training, label='Traing set Alpha 0.1')
plt.plot(n_ierations_Training2, cost_Training2, label='Traing set Alpha 0.01')
plt.plot(n_ierations_Test, cost_Test, label='Test set Alpha 0.1')
plt.plot(n_ierations_Test2, cost_Test2, label='Test 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[175…

```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



In [ ]: