In [1]:

```
# Importing Libraries
import numpy as np
import pandas as pd
```

In [2]:

```
#reading .csv file and storing it in "dataset" dataframe
dataset=pd.read_csv("/home/ankit/Desktop/ex1data2.csv")
```

In [3]:

```
print(dataset)
```

```
       A  B       C
0   2104  3  399900
1   1600  3  329900
2   2400  3  369000
3   1416  2  232000
4   3000  4  539900
5   1985  4  299900
6   1534  3  314900
7   1427  3  198999
8   1380  3  212000
9   1494  3  242500
10  1940  4  239999
11  2000  3  347000
12  1890  3  329999
13  4478  5  699900
14  1268  3  259900
15  2300  4  449900
16  1320  2  299900
17  1236  3  199900
18  2609  4  499998
19  3031  4  599000
20  1767  3  252900
21  1888  2  255000
22  1604  3  242900
23  1962  4  259900
24  3890  3  573900
25  1100  3  249900
26  1458  3  464500
27  2526  3  469000
28  2200  3  475000
29  2637  3  299900
30  1839  2  349900
31  1000  1  169900
32  2040  4  314900
33  3137  3  579900
34  1811  4  285900
35  1437  3  249900
36  1239  3  229900
37  2132  4  345000
38  4215  4  549000
39  2162  4  287000
40  1664  2  368500
41  2238  3  329900
42  2567  4  314000
43  1200  3  299000
44   852  2  179900
45  1852  4  299900
46  1203  3  239500
```

In [4]:

```
print(dataset.shape) # size of the dataset
```

```
(47, 3)
```

In [5]:

```
data = dataset.to_numpy() # storing dataset as numpy array
```

In [6]:

```
#To take separate values of both features in lists x and y
```

```
x1=[]
y1=[]
z1=[]
for i in range(len(data)):
    x1.append(data[i][0])
for i in range(len(data)):
    y1.append(data[i][1])
for i in range(len(data)):
    z1.append(data[i][2])

# To normalize the data to put it on same scale
norm = np.linalg.norm(x1)
x=x1/norm
norm1 = np.linalg.norm(y1)
y=y1/norm1
norm2 = np.linalg.norm(z1)
z=z1/norm2



print(x)
print(y)
print(z) #actual value of z in Z=ax+by + c
```

```
[0.14276982 0.10857021 0.16285531 0.09608463 0.20356914 0.13469491
 0.10409169 0.09683105 0.0936418  0.10137743 0.13164137 0.13571276
 0.12824856 0.30386086 0.08604189 0.15606967 0.08957042 0.08387048
 0.17703729 0.20567268 0.11990222 0.12811284 0.10884163 0.13313422
 0.26396131 0.07464202 0.0989346  0.17140521 0.14928403 0.17893727
 0.12478788 0.06785638 0.13842701 0.21286546 0.1228879  0.09750962
 0.08407405 0.1446698  0.28601464 0.14670549 0.11291301 0.15186258
 0.17418732 0.08142765 0.05781363 0.12567001 0.08163122]
[0.13429844 0.13429844 0.13429844 0.0895323  0.17906459 0.17906459
 0.13429844 0.13429844 0.13429844 0.13429844 0.17906459 0.13429844
 0.13429844 0.22383074 0.13429844 0.17906459 0.0895323  0.13429844
 0.17906459 0.17906459 0.13429844 0.0895323  0.13429844 0.17906459
 0.13429844 0.13429844 0.13429844 0.13429844 0.13429844 0.13429844
 0.0895323  0.04476615 0.17906459 0.13429844 0.17906459 0.13429844
 0.13429844 0.17906459 0.17906459 0.17906459 0.0895323  0.13429844
 0.17906459 0.13429844 0.0895323  0.17906459 0.13429844]
[0.16105104 0.13286006 0.14860674 0.09343296 0.217433   0.12077821
 0.12681914 0.08014253 0.0853784  0.09766161 0.09665439 0.13974671
 0.13289993 0.28186953 0.10466908 0.18118746 0.12077821 0.08050538
 0.20136334 0.24123424 0.10184998 0.10269571 0.0978227  0.10466908
 0.23112576 0.1006418  0.18706729 0.18887957 0.19129594 0.12077821
 0.14091463 0.06842354 0.12681914 0.23354213 0.11514002 0.1006418
 0.09258723 0.13894126 0.22109783 0.11558302 0.14840537 0.13286006
 0.12645668 0.12041576 0.07245082 0.12077821 0.09645342]
```

In [7]:

```
# Applying 10-fold cross-validation

#1st shuffle the data
import random
random.shuffle(x)
random.shuffle(y)
random.shuffle(z)
#splitting the shuffled data of 1st feature into 10 groups



fold1_x=[]
fold2_x=[]
fold3_x=[]
fold4_x=[]
fold5_x=[]
fold6_x=[]
fold7_x=[]
fold8_x=[]
fold9_x=[]
```

```
fold10_x=[]
for i in range(0,4):
    fold1_x.append(x[i])
for i in range(4,8):
    fold2_x.append(x[i])
for i in range(8,12):
    fold3_x.append(x[i])
for i in range(12,17):
    fold4_x.append(x[i])
for i in range(17,22):
    fold5_x.append(x[i])
for i in range(22,27):
    fold6_x.append(x[i])
for i in range(27,32):
    fold7_x.append(x[i])
for i in range(32,37):
    fold8_x.append(x[i])
for i in range(37,42):
    fold9_x.append(x[i])
for i in range(42,47):
    fold10_x.append(x[i])


fold1_y=[]
fold2_y=[]
fold3_y=[]
fold4_y=[]
fold5_y=[]
fold6_y=[]
fold7_y=[]
fold8_y=[]
fold9_y=[]
fold10_y=[]
for i in range(0,4):
    fold1_y.append(y[i])
for i in range(4,8):
    fold2_y.append(y[i])
for i in range(8,12):
    fold3_y.append(y[i])
for i in range(12,17):
    fold4_y.append(y[i])
for i in range(17,22):
    fold5_y.append(y[i])
for i in range(22,27):
    fold6_y.append(y[i])
for i in range(27,32):
    fold7_y.append(y[i])
for i in range(32,37):
    fold8_y.append(y[i])
for i in range(37,42):
    fold9_y.append(y[i])
for i in range(42,47):
    fold10_y.append(y[i])


fold1_z=[]
fold2_z=[]
fold3_z=[]
fold4_z=[]
fold5_z=[]
fold6_z=[]
fold7_z=[]
fold8_z=[]
fold9_z=[]
fold10_z=[]
for i in range(0,4):
    fold1_z.append(z[i])
for i in range(4,8):
    fold2_z.append(z[i])
for i in range(8,12):
    fold3_z.append(z[i])
for i in range(12,17):
    fold4_z.append(z[i])
for i in range(17,22):
    fold5_z.append(z[i])
for i in range(22,27):
    fold6_z.append(z[i])
for i in range(27,32):
    fold7_z.append(z[i])
for i in range(32,37):
    fold8_z.append(z[i])
for i in range(37,42):
    fold9_z.append(z[i])
for i in range(42,47):
    fold10_z.append(z[i])

for i in range(47):
    print(fold1_x,fold2_x,fold3_x,fold4_x,fold5_x,fold6_x,fold7_x,fold8_x,
for i in range(47):
    print(fold1_y,fold2_y,fold3_y,fold4_y,fold5_y,fold6_y,fold7_y,fold8_y,
for i in range(47):
```

```
[0.10884163159437348, 0.09608463238007035, 0.057813634737160974, 0.14928403335
886636] [0.20567268414123815, 0.2639613135299955, 0.15186257575324677, 0.12811
284317342714] [0.13842701275094882, 0.28601463663982807, 0.1446697996005014,
0.09750961633485952] [0.11291301432234256, 0.10857020607917553, 0.134694911916
97713, 0.17418732437827725, 0.12288790200586681] [0.08407405333256156, 0.20356
913639845411, 0.1282485559310261, 0.21286546029398354, 0.07464201667943318]
[0.10137742992643016, 0.0989346002896487, 0.1770372922878556, 0.12478788061225
238, 0.0816312236957801] [0.13164137487100033, 0.14276982099411584, 0.06785637
879948471, 0.08604188831774662, 0.0936418027432889] [0.14670549096448593, 0.09
683105254686468, 0.08142765455938165, 0.1628553091187633, 0.15606967123881482]
[0.17140521284749838, 0.1789372708942412, 0.10409168507840955, 0.1256700135366
4567, 0.0838704841961631] [0.08957042001531981, 0.13571275759896942, 0.3038608
6426409253, 0.133134215204589, 0.11990222133868948]
[0.10884163159437348, 0.09608463238007035, 0.057813634737160974, 0.14928403335
886636] [0.20567268414123815, 0.2639613135299955, 0.15186257575324677, 0.12811
284317342714] [0.13842701275094882, 0.28601463663982807, 0.1446697996005014,
0.09750961633485952] [0.11291301432234256, 0.10857020607917553, 0.134694911916
97713, 0.17418732437827725, 0.12288790200586681] [0.08407405333256156, 0.20356
913639845411, 0.1282485559310261, 0.21286546029398354, 0.07464201667943318]
[0.10137742992643016, 0.0989346002896487, 0.1770372922878556, 0.12478788061225
```

In [8]:

```
len(fold10_x)
len(fold10_y)
len(fold10_z)
```

Out[8]: 5

In [9]:

```
#Now we define the gradient descent method to build the model

def gradient_descent(x,y):

    a=0.45
    b=0.35
    c=0.33 # initializing unknown parameters which we have to find with son

    l = 0.001 # assigning learning rate as 0.0001 for good accuracy

    iterations = 1000 # initializaing number of iterations

    n= 47 # total number of datapoints

# Here, error function is sum of squared error function which is E = 1/2 su
# where z_pred is ax_i + by_i + c



```

```

    for p in range(iterations):
        sum1=0
        D_a=0
        D_b=0
        D_c=0
        for i in range(10):
             z_pred = a*x[i]+b*y[i]+c
             #print(z_pred)
             sum1 += (z[i]-z_pred)**2 # Calculating total sum of squared er
             #print(sum1)
             sse = sum1/2
             #print(sse)
             if sse < 0.02: # considering 0.02 is close to zero
                    break
        for i in range(10):
            z_pred = a*x[i]+b*y[i]+c
            #print(z_pred)
            D_a += (-1)*x[i]*(z[i] - z_pred)    #partial derivative of erro
            #print(D_a)
            D_b += (-1)*y[i]*(z[i] - z_pred)    #partial derivative of erro
            #print(D_b)
            D_c += (-1)*(z[i] - z_pred)         #partial derivative of erro
            #print(D_c)

            # Adjust the weights with the gradients to reach the optimal va

        a = a - l*D_a
        b = b - l*D_b
        c = c - l*D_c
        #print(a,b,c)
        # we will use these updated value for next iteration and we do this
        #print("At iteration %d, The value of sse is: %2.5f  " %(p,sse))
        #print("final sum of squared error using gradient descent : ", sse)
        #print("Finalvalues of a,b,c are: ",a,b,c)
        return a,b,c
```

In [10]:

```
# Now, we take 9 folds data for training and one fold data for testing
d1=fold1_x + fold2_x + fold3_x + fold4_x + fold5_x + fold6_x + fold7_x + fo
d2=fold1_y + fold2_y + fold3_y + fold4_y + fold5_y + fold6_y + fold7_y + fo
```

In [11]:

```
(a,b,c)=gradient_descent(d1,d2) # to get the hyperplane parameters for d1 a

print(a,b,c)
```

```
0.4495065432163269 0.3495738296045063 0.326943286546175
```

In [12]:

```
# So, equation of hyperplane is z=0.4495745592685494*x + 0.3494710627195641
# Now, we test the model on 10th fold data and calculate the mean squared e
sum=0
for i in range(5):
    z_predict= 0.4495745592685494*fold10_x[i] + 0.34947106271956413*fold10_
    z_actual=fold10_z[i]
    sum +=(z_actual-z_predict)**2
error=sum/5
```

In [13]:

```
print(error) # it is mean-squared error when we take 1st 9 folds data as tr

```

```
0.0966384762110927
```

In [14]:

```
#simialrly by taking 9 combinations of folding data for training and remair
#and calculate the mean squared error
```