In [1]:
```
# For this task, again consider, dependent variable is "Price of the flat (
# i.e "Size of the flat in square ft(x)" & "number of bedrooms in the flat(
# let z = ax+by+c is a hyperplane which is nearest to the points given in t
# Here, I will use gradient descent method
# I will consider the sum of squared error function as loss function to cal
```

In [2]:
```
#Importing libraries
import numpy as np
```

In [3]:
```
dataset = [[1600,3,8.2],
           [1260,2,6.6],
           [1800,4,10.3],
           [600,1,1.7],
           [850,2,3.6],
           [920,2,4.4],
           [1090,2,5.4],
           [890,2,4.8],
           [1340,3,10.5],
           [1650,2,7.4]]
# For each sublist of list dataset, 1st entry shows x value, 2nd entry show
```

In [4]:
```
#To take separate values of both features in lists x and y
x1=[]
y1=[]
z1=[]
for i in range(10):
    x1.append(dataset[i][0])
for i in range(10):
    y1.append(dataset[i][1])
for i in range(10):
    z1.append(dataset[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
```

```
[0.40233529 0.31683904 0.4526272  0.15087573 0.21374062 0.23134279
 0.27409092 0.22379901 0.33695581 0.41490827]
[0.39056673 0.26037782 0.52075564 0.13018891 0.26037782 0.26037782
 0.26037782 0.26037782 0.39056673 0.26037782]
[0.37851574 0.30465901 0.4754527  0.07847278 0.16617764 0.20310601
 0.24926647 0.22157019 0.48468479 0.34158738]
```

In [5]:
```
# Applying Gradient Descent algorithm

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

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

iterations = 1000 # initializaing number of iterations

n= 10 # 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 error(s
        #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 error fu
        #print(D_a)
        D_b += (-1)*y[i]*(z[i] - z_pred)    #partial derivative of error fu
        #print(D_b)
        D_c += (-1)*(z[i] - z_pred)         #partial derivative of error fu
        #print(D_c)

    # Adjust the weights with the gradients to reach the optimal values whe

    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 unt
    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)
```

```
At iteration 0, The value of sse is: 0.41189
At iteration 1, The value of sse is: 0.40280
At iteration 2, The value of sse is: 0.39393
At iteration 3, The value of sse is: 0.38526
At iteration 4, The value of sse is: 0.37680
At iteration 5, The value of sse is: 0.36854
At iteration 6, The value of sse is: 0.36047
At iteration 7, The value of sse is: 0.35259
At iteration 8, The value of sse is: 0.34490
At iteration 9, The value of sse is: 0.33739
At iteration 10, The value of sse is: 0.33005
At iteration 11, The value of sse is: 0.32289
At iteration 12, The value of sse is: 0.31590
At iteration 13, The value of sse is: 0.30907
At iteration 14, The value of sse is: 0.30240
At iteration 15, The value of sse is: 0.29588
At iteration 16, The value of sse is: 0.28952
At iteration 17, The value of sse is: 0.28331
At iteration 18, The value of sse is: 0.27725
At iteration 19, The value of sse is: 0.27133
```

In [6]:

```
print("final sum of squared error using gradient descent : ", sse)
print("Finalvalues of a,b,c are: ", a,b,c)
```

```
final sum of squared error using gradient descent :  0.021821776978798908
Finalvalues of a,b,c are:  0.4171638121446569 0.32211296484688473 0.0703519441
3069641
```

As we can see here, after each iteration, value of sse(sum of squared error)

is decreasing and converging to zero.

After 1000 iteration, values of a,b and c are 0.4171638121446569 0.32211296484688473 0.07035194413069641

So, by using Gradient Descent, Best fit hyperplane for given data is z=0.41x + 0.32y + 0.07 for normalized data. If data is not normalized then it would be different because after normalization, data points will be in range 0 to 1.

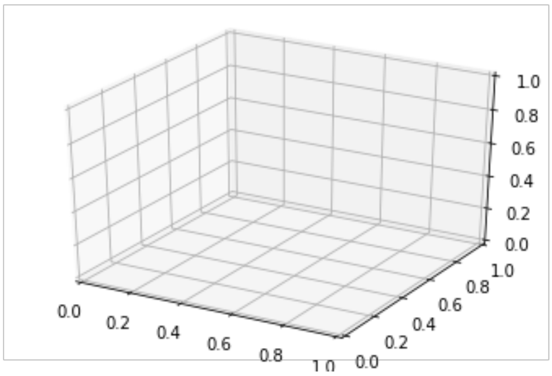
Task 2.2.c

In [7]:

```
#Now we will plot the 3d points for the given normalized data
#For 3D plotting, I didn't know anything, so, I referred  this article http
```

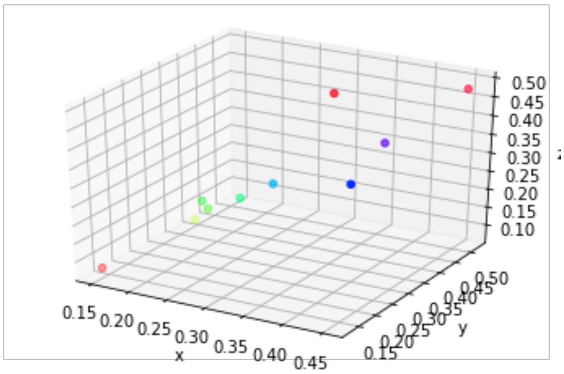
In [8]:

```
#Importing libraries
from mpl_toolkits import mplot3d

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection="3d")

plt.show()
```



In [9]:

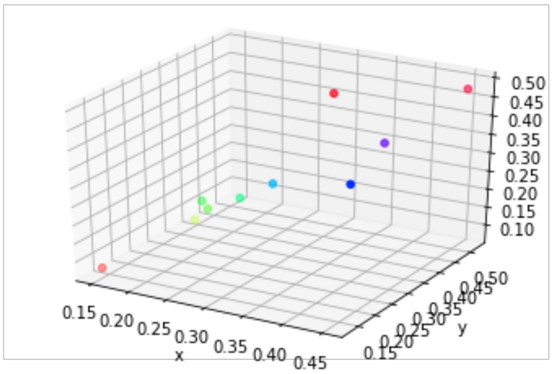
```
#Now that our axes are created we can start plotting in 3D.
fig = plt.figure()
ax = plt.axes(projection="3d")

z_points = z
x_points = x
y_points = y
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
#here, z points are colored
```

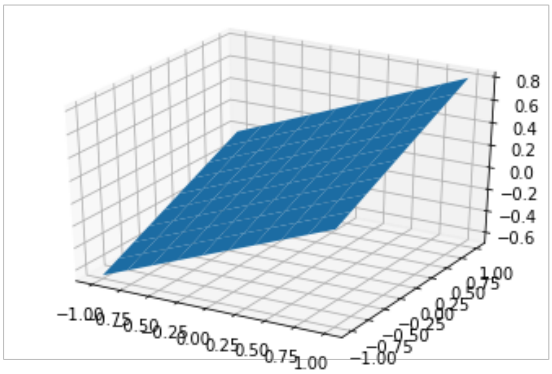


In [10]:

```
# To draw a plane, I have referred this link https://stackoverflow.com/ques
```

In [11]:
```
from mpl_toolkits.mplot3d import Axes3D

x1 = np.linspace(-1,1,10)
y1 = np.linspace(-1,1,10)

fig = plt.figure()
ax = plt.axes(projection="3d")

z_points = z
x_points = x
y_points = y
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')


X,Y = np.meshgrid(x1,y1)
Z= 0.4171638121446569*X + 0.32211296484688473*Y + 0.07035194413069641

fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, Z)
plt.show()
```





In [12]:
```
# plot of hyperplane which I got using least method will be :
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-1,1,10)
y = np.linspace(-1,1,10)

X,Y = np.meshgrid(x,y)
Z=0.00362953*X + 1.67818099*Y-1.9252501

fig = plt.figure()
```

```
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, Z)
```