Importing Required Packages

In [35]:
```
import numpy as np
import pandas as pd
import sklearn.linear_model as lr
```

In [36]:
```
Y=[1,0,1,4,3,2,5,6,9,13,15,16]
Y
```

Out[36]:
```
[1, 0, 1, 4, 3, 2, 5, 6, 9, 13, 15, 16]
```

In [37]:
```
X=[[1,1,1],
[1,2,1],
[1,2,2],
[1,3,2],
[1,5,4],
[1,5,6],
[1,6,5],
[1,7,4],
[1,10,8],
[1,11,7],
[1,11,9],
[1,12,10]]
print(np.array(X).shape)
X
```

```
(12, 3)
```

Out[37]:
```
[[1, 1, 1],
 [1, 2, 1],
 [1, 2, 2],
 [1, 3, 2],
 [1, 5, 4],
 [1, 5, 6],
 [1, 6, 5],
 [1, 7, 4],
 [1, 10, 8],
 [1, 11, 7],
 [1, 11, 9],
 [1, 12, 10]]
```

Modifying the beta co-efficients such that Matrix Multiplication Conditions are met

In [38]:
```
b=[0]
#Inistialized beta coefficients to 0
b=np.transpose(b*len(X[0]))
# Matrix multiplication is possible by transpose
print(b.shape)
b
```

```
(3,)
```

Out[38]:
```
array([0, 0, 0])
```

Inistializing Required values

In [39]:
```
min=float('inf')
a=0.0001
i=0
```

Applying Gradient Descent Algorithm

In [40]:
```
def linear(X,Y,b,a):
    y1=np.matmul(X,b)
    gradient=-2*np.matmul(np.transpose(X),(Y-y1))
    b=b-a*gradient
    loss=sum(np.power(Y-y1,2))
    return(loss,b)
```

In [41]:
```
for iu in range(30000):
    loss,b=linear(X,Y,b,a)
    if min>loss:
        min=loss
        beta=b
    i+=1
    if(i<10 or i> 29990):
        print(f'iteration={i}, Beta Coeffients: {b}, Loss={loss}')
```

```
iteration=1, Beta Coeffients: [0.015  0.1404 0.1084], Loss=823
iteration=2, Beta Coeffients: [0.02657888 0.25185692 0.19406028], Loss=539.0223544
iteration=3, Beta Coeffients: [0.03544733 0.34038133 0.26170369], Loss=360.7318699
583712
iteration=4, Beta Coeffients: [0.04216843 0.41073554 0.31507223], Loss=248.7891117
8512737
iteration=5, Beta Coeffients: [0.04718834 0.46669283 0.35713079], Loss=178.4977897
913792
iteration=6, Beta Coeffients: [0.05086055 0.51124286 0.39022852], Loss=134.3542052
8676462
iteration=7, Beta Coeffients: [0.05346515 0.5467544  0.41622668], Loss=106.6255331
0358618
iteration=8, Beta Coeffients: [0.05522404 0.57510428 0.43660001], Loss=89.20175472
042988
iteration=9, Beta Coeffients: [0.05631306 0.59777955 0.45251696], Loss=78.24715687
403072
iteration=29991, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257622
iteration=29992, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
00883442576226
iteration=29993, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257623
iteration=29994, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257623
iteration=29995, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257624
iteration=29996, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257625
iteration=29997, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257624
iteration=29998, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
00883442576254
iteration=29999, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257623
iteration=30000, Beta Coeffients: [-2.26303789  1.54972927 -0.2385295 ], Loss=34.1
0088344257623
```

Final Beta Values after Gradient Descent

In [42]:
```
beta #Beta Coefficents of Gradient descent.
```

Out[42]:
```
array([-2.26303788,  1.54972927, -0.2385295 ])
```

Beta Values in OLS

In [43]:
```
X1=np.transpose(X)
#Beta Coefficents of Gradient descent.
np.matmul(np.matmul(np.linalg.inv(np.matmul(X1,X)),X1),Y)
#As both Co-efficients are same upto 7 decimals, Hence Verified.
```

Out[43]:
```
array([-2.2630379 ,  1.54972927, -0.2385295 ])
```