Task 1 :Try Linear Regression just using numpy (Without Tensorflow/Pytorch or other torch library). You can optionally use sklearn (if you want)

In [1]:
```
#import numpy library
import numpy as np
```

In [2]:
```
# Input (temp, rainfall, humidity)
inputs = np.array([[73, 67, 43],
                   [91, 88, 64],
                   [87, 134, 58],
                   [102, 43, 37],
                   [69, 96, 70]], dtype='float32')
# Target (apples)
targets = np.array([[56],
                    [81],
                    [119],
                    [22],
                    [103]], dtype='float32')
m = np.shape(targets)
print("Data size is :",m[0])
```

```
Data size is : 5
```

In [3]:
```
#Add bias
bias = np.ones(m[0])
bias.shape = (1,m[0])
new_input = np.concatenate((inputs,bias.T),axis=1)
print(new_input)
```

```
[[ 73.  67.  43.   1.]
 [ 91.  88.  64.   1.]
 [ 87. 134.  58.   1.]
 [102.  43.  37.   1.]
 [ 69.  96.  70.   1.]]
```

In [4]:
```
#Define All Functions
def gradientDescent(x,y,alpha,num_of_epochs,weight):
  for i in range(0,num_of_epochs):
    weight = weight - (alpha/m[0])*np.dot(x.T,(np.dot(x,weight)-y))
  return weight

def predict(input,weight):
  return np.dot(input,weight)

def costfunc(x,targets,weight):
  term = (predict(x,weight)-targets)
  term = np.dot(term.T,term)
  return term/(2*m[0])
```

In [5]:
```
#Initialize weight with 0
weight = np.zeros((new_input.shape[1],1),dtype='float32')
weight.shape = (new_input.shape[1],1)
```

In [6]:
```
#Intial Cost
init_cost = costfunc(new_input,targets,weight)
print("Initial Cost : ",int(init_cost))
```

```
Initial Cost :  3495
```

In [7]:
```
#Initialize alpha,num_of_epochs
alpha = 0.00001
num_of_epochs = 10000
```

In [8]:
```
#find out weight of each feature
final_weight = gradientDescent(new_input,targets,alpha,num_of_epochs,weight)
```

In [9]:
```
print("Final weight:")
print(final_weight)
```

```
Final weight:
[[-4.00196772e-01]
 [ 8.48044773e-01]
 [ 6.87453282e-01]
 [-8.26566154e-05]]
```

In [10]:
```
final_cost = costfunc(new_input,targets,final_weight)
print("Final cost : ",float(final_cost))
```

```
Final cost :  0.49174454181664
```

In [11]:
```
#Predict output
predicted_output = predict(new_input,final_weight)
print("predicted_output:")
print(predicted_output)
```

```
predicted_output:
[[ 57.16504387]
 [ 82.20696112]
 [118.69308805]
 [ 21.08154323]
 [101.92036798]]
```

In [12]:
```
#Actual target
print("Actual Target:")
print(targets)
```

```
Actual Target:
[[ 56.]
 [ 81.]
 [119.]
 [ 22.]
 [103.]]
```