Linear Regression using Gradient Descent

Gradient descent algorithm's main objective is to minimise the cost function. It is one of the best optimisation algorithms to minimise errors (difference of actual value and predicted value).

Simple Linear Regression

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

In [3]:
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
```

In [5]:
```
df=pd.read_csv(r"C:\Users\Hp\OneDrive\Desktop\study.csv")
```

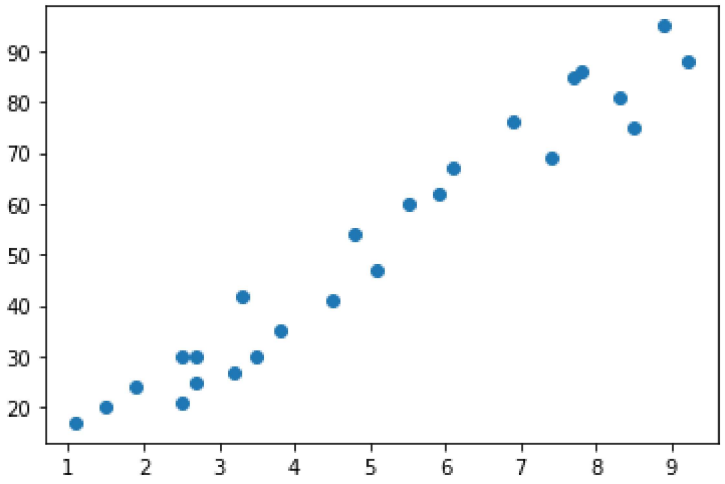
In [6]:
```
df.head()
```

Out[6]:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

In [9]:
```
plt.scatter(df["Hours"],df["Scores"])
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x1b959edae08>
```



In [83]:
```
x=np.array(df["Hours"])
```

In [84]:
```
y=np.array(df["Scores"])
```

Gradient Descent Algorithm

In [371]:
```
def gradient_descent(x,y,w,b,learning_rate):
    dw=0
    db=0
    m=len(x)
    for i in range(m):
        #cost = (1/2*m) * (((w * x) + b) - y)**2
        error=(w*x[i]+b-y[i])
        dw=(2/m)*((error)*x[i])
        db=(2/m)*(error)
        w=w-(learning_rate*dw)
        b=b-(learning_rate*db)
    return(w,b)
```

In [372]:
```
def run(x,y,l,iterations):
    b = 0
    w = 0

    for i in range(iterations):
        w,b= gradient_descent(x,y,w,b,l)
    return [w,b]
```

In [394]:
```
w,b=run(x,y,0.01,2000)
```

In [395]:
```
w
```

Out[395]: 9.888535962544527

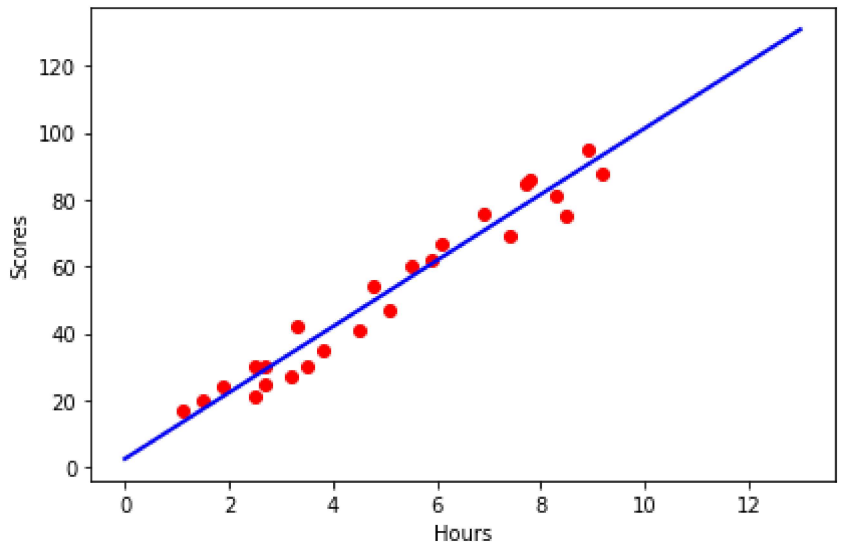
In [396]:
```
b
```

Out[396]: 2.4277094606967387

Plot

In [402]:
```
def drawPlot(x, y, m, b):
    plt.plot(x,y, 'ro')
    plt.plot([0, 13], [0 + b, 13*m + b], color='b', linestyle='-', linewidth=2
)
    plt.xlabel('Hours')
    plt.ylabel('Scores')
    plt.tight_layout()
    plt.show()
```

In [403]:
```
drawPlot(x,y,w,b)
```



In [399]:
```
a=w*x+b
```

In [400]:
```
print("No of Hours = {}".format(9.25))
print("Predicted Score = {}".format(w*9.25+b))
```

```
No of Hours = 9.25
Predicted Score = 93.89666711423361
```

In [401]:
```
from sklearn import metrics
print('Mean Absolute Error:',
      metrics.mean_absolute_error(y, a))
```

```
Mean Absolute Error: 4.94389750055232
```

In [ ]:
```

```