# Experiment 7

**AIM : Linear Regression with Gradient Descent**

***To peform linear regression using gradient descent***

### Description

The idea of linear regression is to find a relationship between our target or dependent variable (y) and a set of explanatory variables ($x_1, x_2 \ldots$). This relatonship can then be used to predict other values.

In our case with one variable, this relationship is a line defined by parameters $\beta$ and the following form: $h_\theta(x) = \beta_0 + \beta_1 x$, where $\beta_0$ is our intercept.

This can be extended to multivariable regression by extending the equation in vector form: $h_\theta(x) = X\beta$

**Cost Function** - (Mean Square Error) measures the average squared difference between an observation's actual and predicted values. The output is a single number representing the cost, or score, associated with our current set of weights. Our goal is to minimize MSE to improve the accuracy of our model.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

**Gradient Descent** - It is a first-order iterative optimization algorithm for finding the local minimum of a function.To minimize Cost Function we use Gradient Descent to calculate the gradient of our cost function. And we simultaneously update $\theta_j$ for all $j$ using the equation below

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Using this equation we will update $\theta$ in order to minimize $J(\theta)$. Until $J(\theta)$ converges to its local minima

**CODE and OUTPUT:**

In [1]:

```
# Linear Regression Using Gradient Descent
# Required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

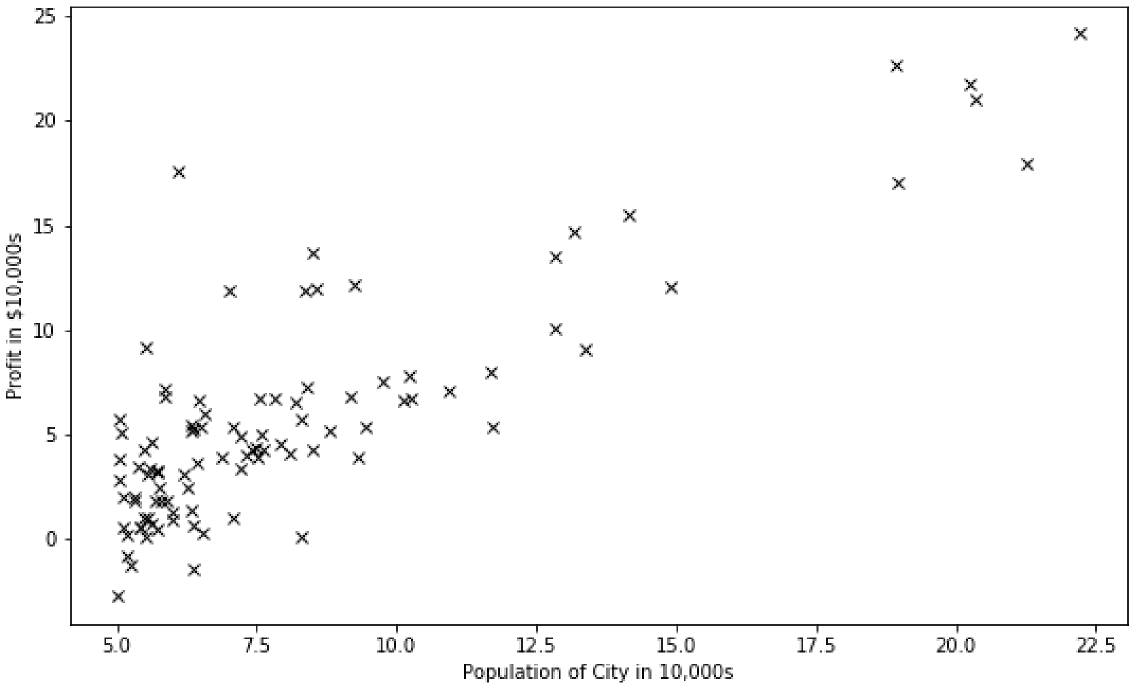
```
# Loading Dataset
df = pd.read_csv('data.txt',names=['x','y'])
X_df = pd.DataFrame(df.x)
y_df = pd.DataFrame(df.y)
df.head()
```

Out[2]:

| | x | y |
|---|---|---|
| **0** | 6.1101 | 17.5920 |
| **1** | 5.5277 | 9.1302 |
| **2** | 8.5186 | 13.6620 |
| **3** | 7.0032 | 11.8540 |
| **4** | 5.8598 | 6.8233 |

In [3]:

```
# Visualing Data
plt.figure(figsize=(10,6))
plt.plot(X_df,y_df,'kx')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.show()
```



In [4]:

```
# converting Pandas DataFrame to Numpy arrays to do calculations
X_df['bias'] = 1
m = df.count()[0]
X = np.array(X_df)
y = np.array(y_df).flatten()
y.shape
```

Out[4]:

```
(97,)
```

In [5]:

```
# Initializing theta and Defining Parameter
theta = np.array([0,0])
alpha = 0.01
iters = 1500
```

In [6]:

```
# Cost Function
def cost_func(X,y,theta):
    m = len(y)
    return np.sum((X.dot(theta) -y)**2)/(2*m)
cost_func(X,y,theta)
```

Out[6]:

```
32.072733877455676
```

In [7]:

```
# Performing Gradient Descent on data
def gradientDescent(X,y,theta,alpha,iters):
    m = len(y)
    cost_hist = [0]*(iters+1)
    for i in range(iters):
        cost_hist[i] = cost_func(X,y,theta)
        h = X.dot(theta)
        loss = h - y
        grad = X.T.dot(loss)/m
        theta = theta - alpha*grad

    cost_hist[iters] = cost_func(X,y,theta)
    return theta,cost_hist
t,c = gradientDescent(X,y,theta,alpha,iters)
t
```

Out[7]:

```
array([ 1.16636235, -3.63029144])
```

In [8]:

```
# Predicting Values using Calculated Theta i.e. t
print(np.array([3.5,1]).dot(t))
print(np.array([7,1]).dot(t))
```

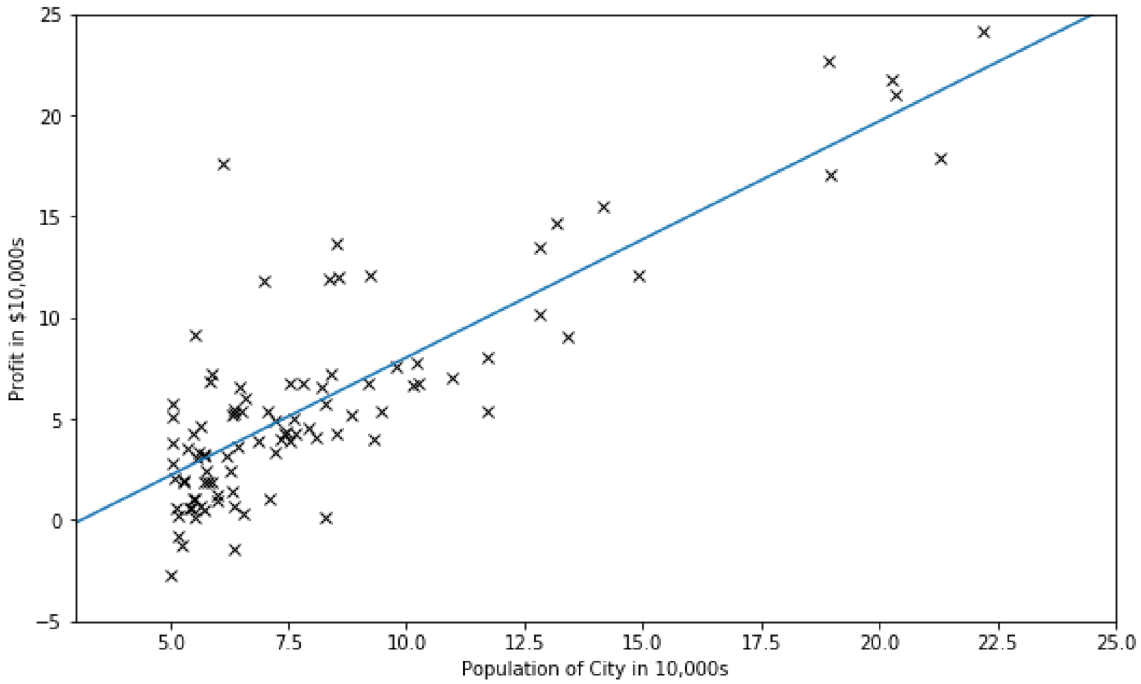
```
0.4519767867701767
4.534245012944714
```

In [9]:

```
# Best Fit Line
X_best_fit = np.linspace(0,25,20)
y_best_fit = [np.array([x,1]).dot(t) for x in X_best_fit]
```

In [10]:

```
# Plotting Best Fit Line
plt.figure(figsize=(10,6))
plt.plot(X_df.x,y_df,'kx')
plt.plot(X_best_fit,y_best_fit)
plt.axis([3,25,-5,25])
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.show()
```




**LEARNING OUTCOMES :**

***In this Experiment, we learned about linear regression using gradient descent and how can we use gradient descent to minimize cost function.***