Gradient Descent

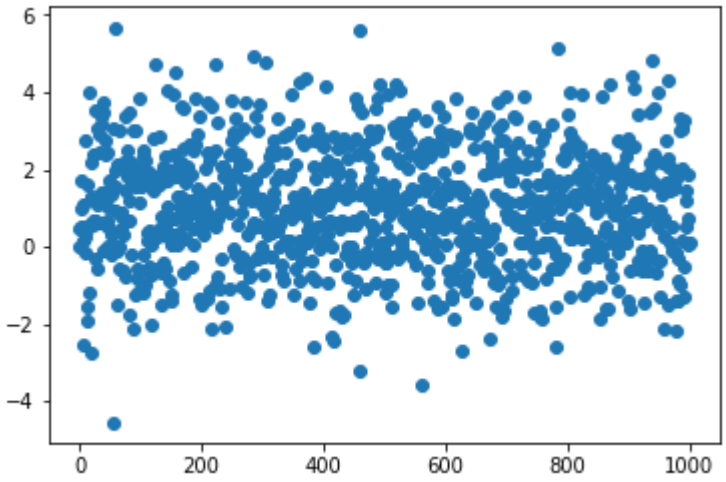
In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
w1 = 0.2
w2 = 0.4
x1 = np.random.normal(1,1.5, 1000)
x2 = np.random.normal(0.5, 2, 1000)
e = np.random.uniform(0.1,1, 1000)
#y_actual = w1*x1 + w2*x2 + e
y_actual = w1*x1 + e
```

In [3]:
```
import matplotlib.pyplot as plt
```

In [4]:
```
plt.scatter(np.linspace(0,1000,1000), x1 )
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x1400c1f1668>
```



In [5]:
```
type(x1)
```

Out[5]:
```
numpy.ndarray
```

In [6]:
```
col1 = x1.reshape(1000,1)
col2 = x2.reshape(1000,1)
```

In [7]:
```
data = np.hstack((col1, col2))
```

In [8]:
```
df = pd.DataFrame({'X1': data[:,0], 'X2': data[:, 1]})
df.head()
```

Out[8]:

	X1	X2
0	0.022737	0.147426
1	0.493748	3.335361
2	0.545081	-0.922906
3	1.740082	3.841892
4	0.990417	-0.756901

- cost = squared.sum(diff(y_actual , y_predicted))
- delta_cost/d_w1 = -2/m *sum((y_actual - y_pred)* x1)
- delta_cost/d_e = -2/m * sum((y_actual - y_pred))

In [9]:
```
m = 0.1
c = 0
n = 1000
L = 0.01  # The learning Rate
ct = []
para = []
for epoch in range(0,1000):
    y_pred = m*x1 + c
    cost = -2/n * (sum(np.power((y_actual - y_pred), 2)))
    D_m = -2/n * (sum(x1*(y_actual - y_pred))) # Derivative wrt m
    D_c = (-2/n) * sum(y_actual - y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
    ct.append(cost)
    para.append((m,c))
```

In [10]:
```
pos = ct.index(min(ct, key = abs))
m = para[pos][0]
c = para[pos][1]
cost = ct[pos]
```

In [11]:
```
m,c, cost
```

Out[11]:
```
(0.19733356529361595, 0.5621527289678955, -0.13025097842687666)
```

In [12]:
```
Y_pred = m*x1 + c

plt.scatter(x1, y_actual)
plt.plot([min(x1), max(x1)], [min(Y_pred), max(Y_pred)], color='red')  # regre
ssion line
plt.show()
```