In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_csv("country_wise_latest.csv")
x= data['Active']
y= data['Deaths']

data
```

Out[2]:

	Country/Region	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	De
0	Afghanistan	36263	1269	25198	9796	106	10	18	
1	Albania	4880	144	2745	1991	117	6	63	
2	Algeria	27973	1163	18837	7973	616	8	749	
3	Andorra	907	52	803	52	10	0	0	
4	Angola	950	41	242	667	18	1	0	
...	...	...	...	...	...	...	...	...	
182	West Bank and Gaza	10621	78	3752	6791	152	2	0	
183	Western Sahara	10	1	8	1	0	0	0	
184	Yemen	1691	483	833	375	10	4	36	
185	Zambia	4552	140	2815	1597	71	1	465	
186	Zimbabwe	2704	36	542	2126	192	2	24	

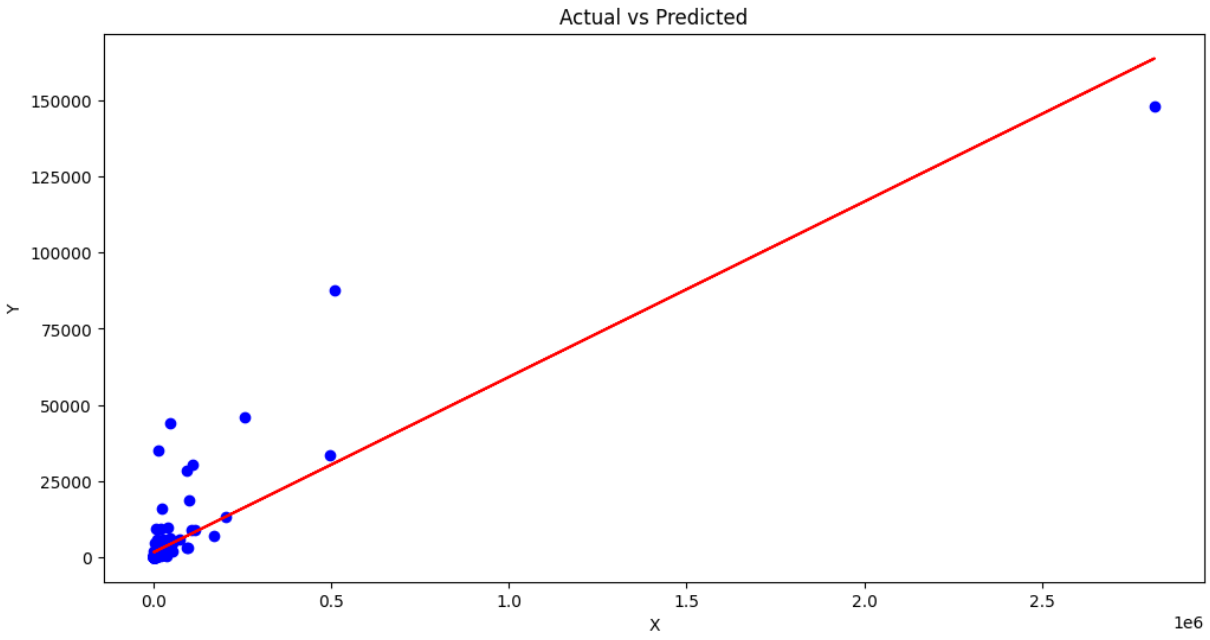
187 rows × 15 columns

In [3]:
```
xmean = np.mean(x)
ymean = np.mean(y)

data['xycov'] = ((x - xmean) * (y- ymean))
data['xvar'] = (x - xmean)**2

beta = data['xycov'].sum() / data['xvar'].sum()
alpha = ymean - (beta * xmean)

print(beta)
print(alpha)
```

```
0.05760832536327237
1538.7241343664855
```

In [4]:
```
ypred = beta*x + alpha
```

In [5]:
```
#plotting
plt.figure(figsize = (12,6))
plt.plot(x,y, 'bo')
plt.plot(x,ypred,'r') #linear regression line
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Actual vs Predicted')

plt.show()
```



In [6]:
```
from numpy import *
```

In [7]:
```
learning_rate = 0.0001
initial_b = 0
initial_m = 0
iterations = 20
```

In [8]:
```
X = data['Active'].to_numpy()
Y = data['Deaths'].to_numpy()
dpoints = np.column_stack((X,Y))
```

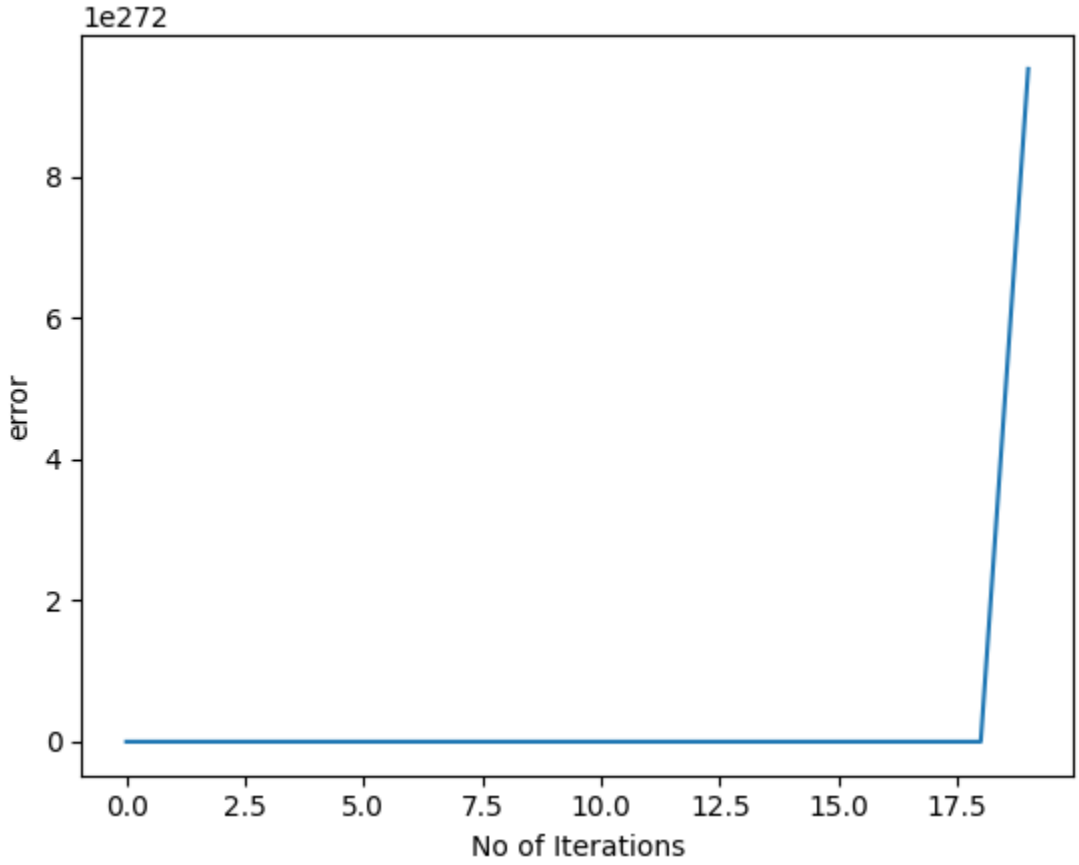
In [9]:
```
def compute_perror (b,m,dpoints):
    total_error = 0
    N = float(len(dpoints))

    for i in range (0, len(dpoints)):
        x = dpoints[i,0]
        y = dpoints[i,1]
        total_error += (y-(m*x + b))**2
```

```
    return total_error / N

def gradient_descent_runner(dpoints, starting_b, starting_m, learning_rate, iterati
    b = starting_b
    m = starting_m
    error_graph = []

    for i in range(iterations):
        error_graph.append(compute_perror(b,m,dpoints))
        b,m = gradient_descent(b,m, array(dpoints),learning_rate)

    return [b,m,error_graph]

def gradient_descent(b_current,m_current, dpoints,learning_rate):
    m_gradient = 0
    b_gradient = 0

    N = float(len(dpoints))

    for i in range(0, len(dpoints)):
        x = dpoints[i,0]
        y = dpoints[i,1]

        m_gradient += -(2/N)  * x * (y-m_current * x + b_current)
        b_gradient += -(2/N)  * (y-(m_current * x + b_current))

        m_updated = m_current - learning_rate * m_gradient
        b_updated = b_current - learning_rate * b_gradient

    return b_updated, m_updated
```

In [10]:
```
b,m,error_graph = gradient_descent_runner(dpoints, initial_b,initial_m,learning_rat

print("Optimized b: ",b)
print("Optimized m: ",m)
print("Minimized Error: ", compute_perror(b,m,dpoints))
```

```
Optimized b:  -9.739743724136337e+131
Optimized m:  -1.3297095290957034e+138
Minimized Error:  8.20779416835156e+286
```

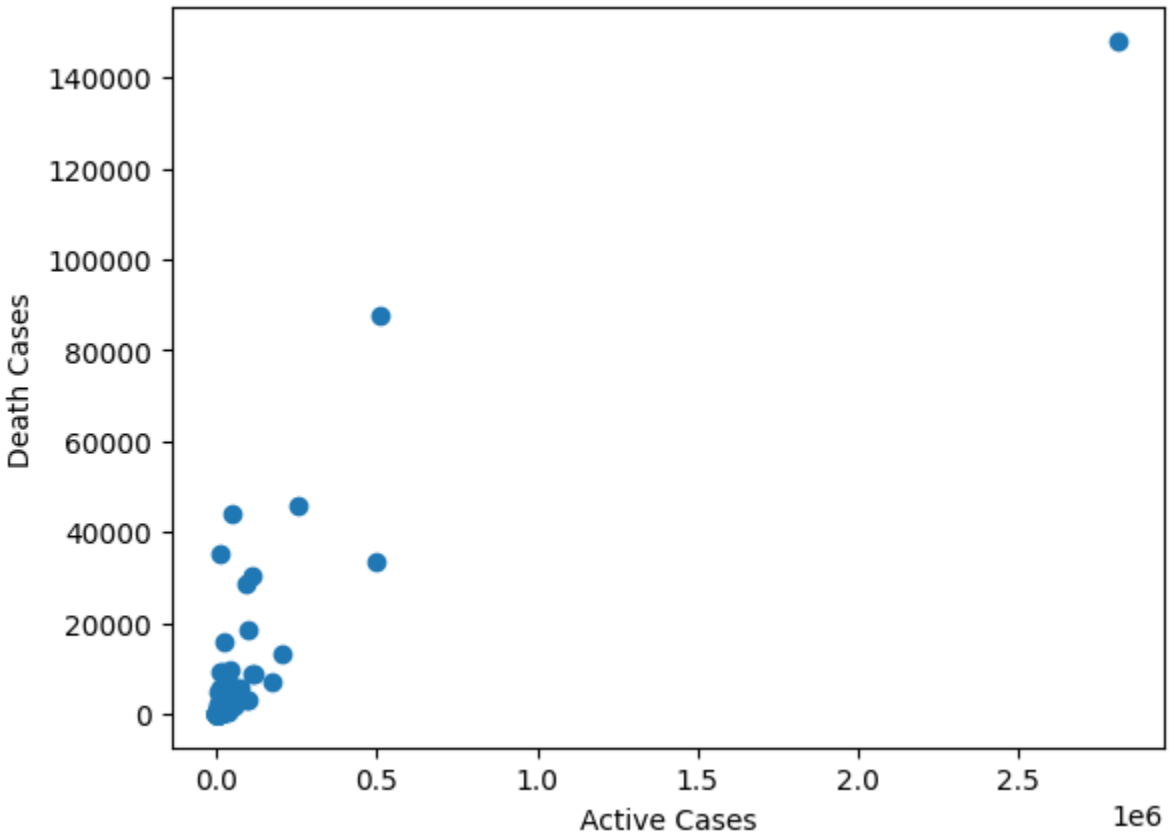
In [11]:
```
plt.plot(error_graph)
plt.xlabel('No of Iterations')
plt.ylabel('error')
plt.show()
```



In [12]:
```
from sklearn.linear_model import LinearRegression
```

In [16]:
```
cols= ['Active']
x = data[cols]
y= data['Deaths']

lm = LinearRegression()
model = lm.fit(x,y)

plt.xlabel('Active Cases')
plt.ylabel('Death Cases')
plt.scatter(x,y)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x19f2b4488d0>
```

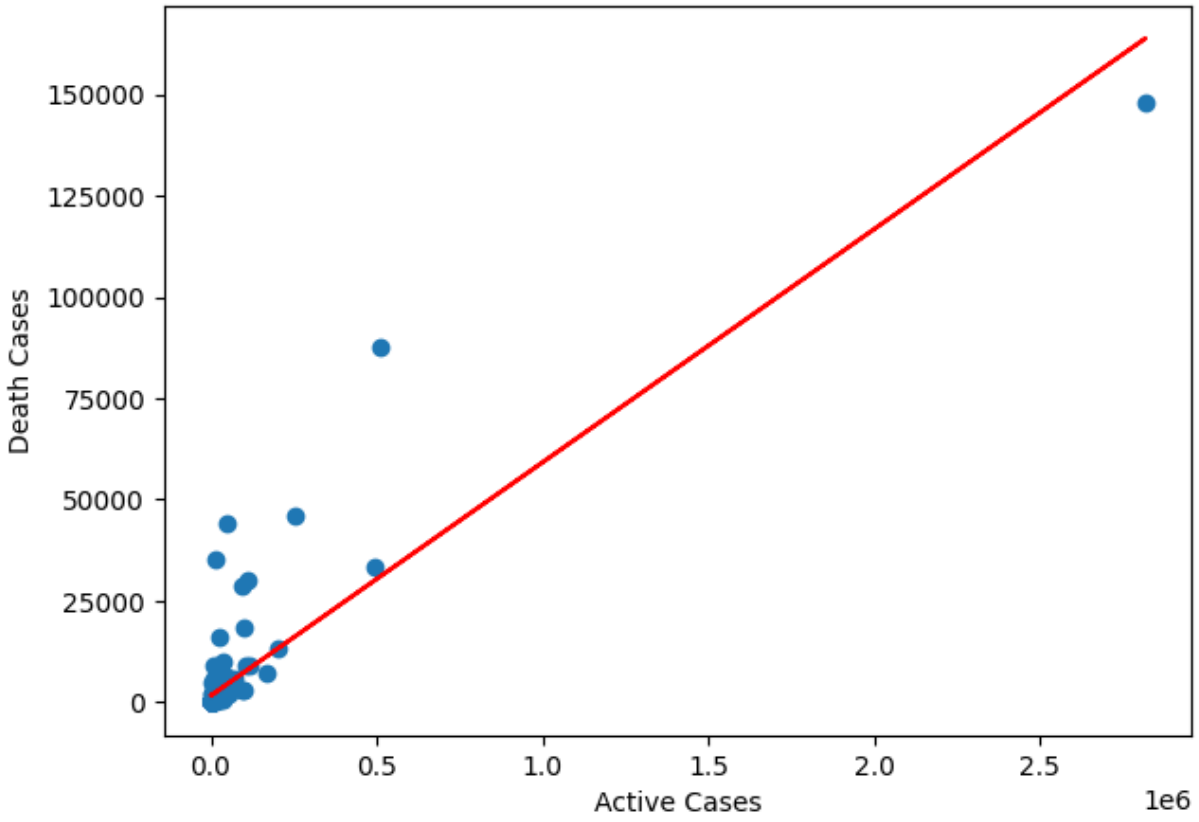


In [14]:
```
alpha = model.intercept_
beta = model.coef_
print("alpha = {}".format(alpha))
print("beta = {}".format(beta))
```

```
alpha = 1538.7241343664857
beta = [0.05760833]
```

In [23]:
```
plt.figure(figsize = (7,5))
plt.scatter(x,y)
plt.xlabel('Active Cases')
plt.ylabel('Death Cases')
plt.plot(x,alpha + beta*x,'r')
plt.show
```

Out[23]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```


150000
125000
100000
75000
50000
25000
0
Death Cases
0.0
0.5
1.0
1.5
2.0
2.5
Active Cases
1e6

In [ ]: