In [1]:
```
# Assignment 1, Problem 1
#-------------------------------------------------------------------------

import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
from my_functions import *
```

In [2]:
```
df = pd.read_csv('./HW1.csv')
df.head()
m = len(df)
m
```

Out[2]: 100

In [3]:
```
df
```

Out[3]:

	X1	X2	X3	Y
0	0.000000	3.440000	0.440000	4.387545
1	0.040404	0.134949	0.888485	2.679650
2	0.080808	0.829899	1.336970	2.968490
3	0.121212	1.524848	1.785455	3.254065
4	0.161616	2.219798	2.233939	3.536375
...	...	...	...	...
95	3.838384	1.460202	3.046061	-4.440595
96	3.878788	2.155152	3.494545	-4.458663
97	3.919192	2.850101	3.943030	-4.479995
98	3.959596	3.545051	0.391515	-3.304593
99	4.000000	0.240000	0.840000	-5.332455

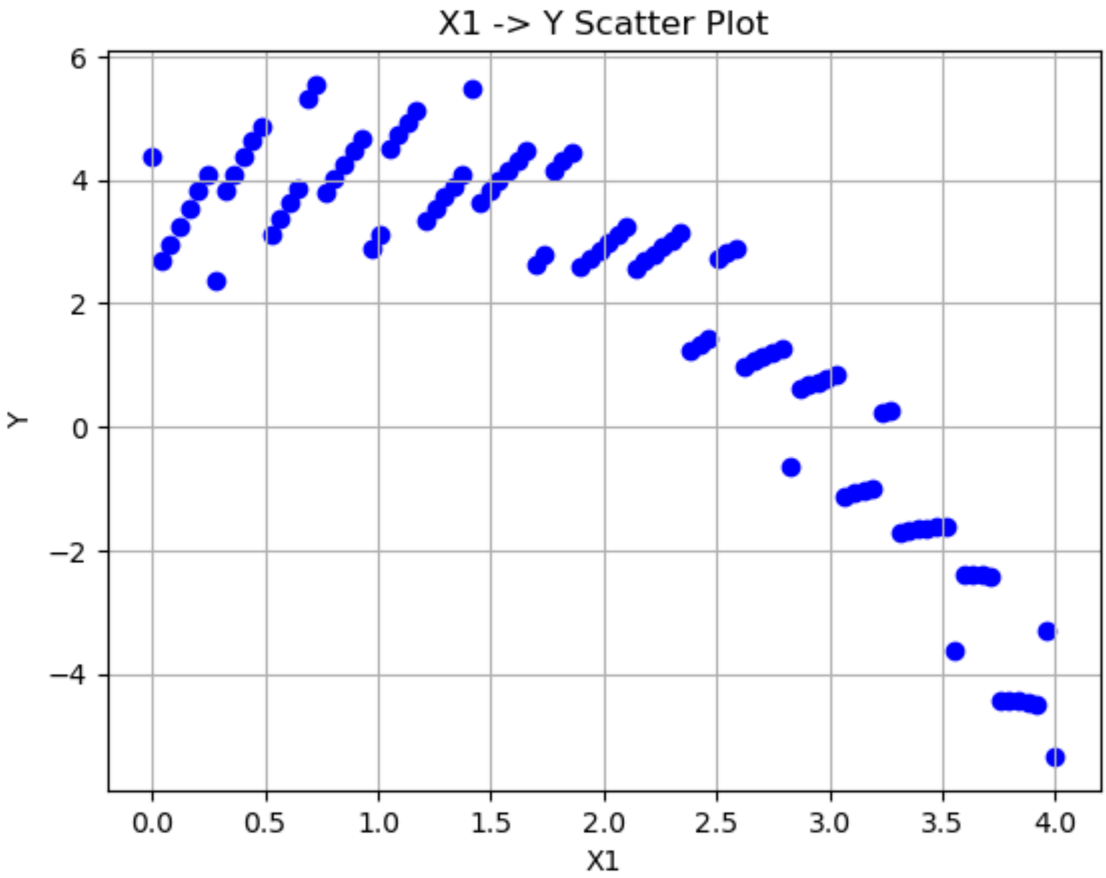
100 rows × 4 columns

In [4]:
```
# Seperate features and labels
X1 = df.values[:,0]
X2 = df.values[:,1]
X3 = df.values[:,2]
Y = df.values[:,3]
m = len(Y)
n = len(X1)

print('X1 = ', X1[: 5])
print('X2 = ', X2[: 5])
print('X3 = ', X3[: 5])
print('Y = ', Y[: 5])
print('m = ',m)
print('n = ',n)
```

```
X1 =  [0.         0.04040404 0.08080808 0.12121212 0.16161616]
X2 =  [3.44       0.1349495  0.82989899 1.52484848 2.21979798]
X3 =  [0.44       0.88848485 1.3369697  1.78545454 2.23393939]
Y =  [4.38754501 2.6796499  2.96848981 3.25406475 3.53637472]
m =  100
n =  100
```

In [5]:

```
# X1 Scatter Plot
plt.scatter(X1,Y,color='b')
plt.grid()
plt.xlabel('X1')
plt.ylabel('Y')
plt.title('X1 -> Y Scatter Plot');
```



In [6]:

```
# Build linear regression model
X1_0 = np.ones((m,1))                      # Bias column (same size as data
X1_0[:5]
```

Out[6]:

```
array([[1.],
       [1.],
       [1.],
       [1.],
       [1.]])
```

In [7]:

```
X1_1 = X1.reshape(m,1)                     # Reshaping X1 to fit matrix ope
X1_1[:10]
```

Out[7]:
```
array([[0.        ],
       [0.04040404],
       [0.08080808],
       [0.12121212],
       [0.16161616],
       [0.2020202 ],
       [0.24242424],
       [0.28282828],
       [0.32323232],
       [0.36363636]])
```

In [8]:
```
X1_feat = np.hstack((X1_0,X1_1))          # Concatonate bias and training
X1_feat[:5]
```

Out[8]:
```
array([[1.        , 0.        ],
       [1.        , 0.04040404],
       [1.        , 0.08080808],
       [1.        , 0.12121212],
       [1.        , 0.16161616]])
```

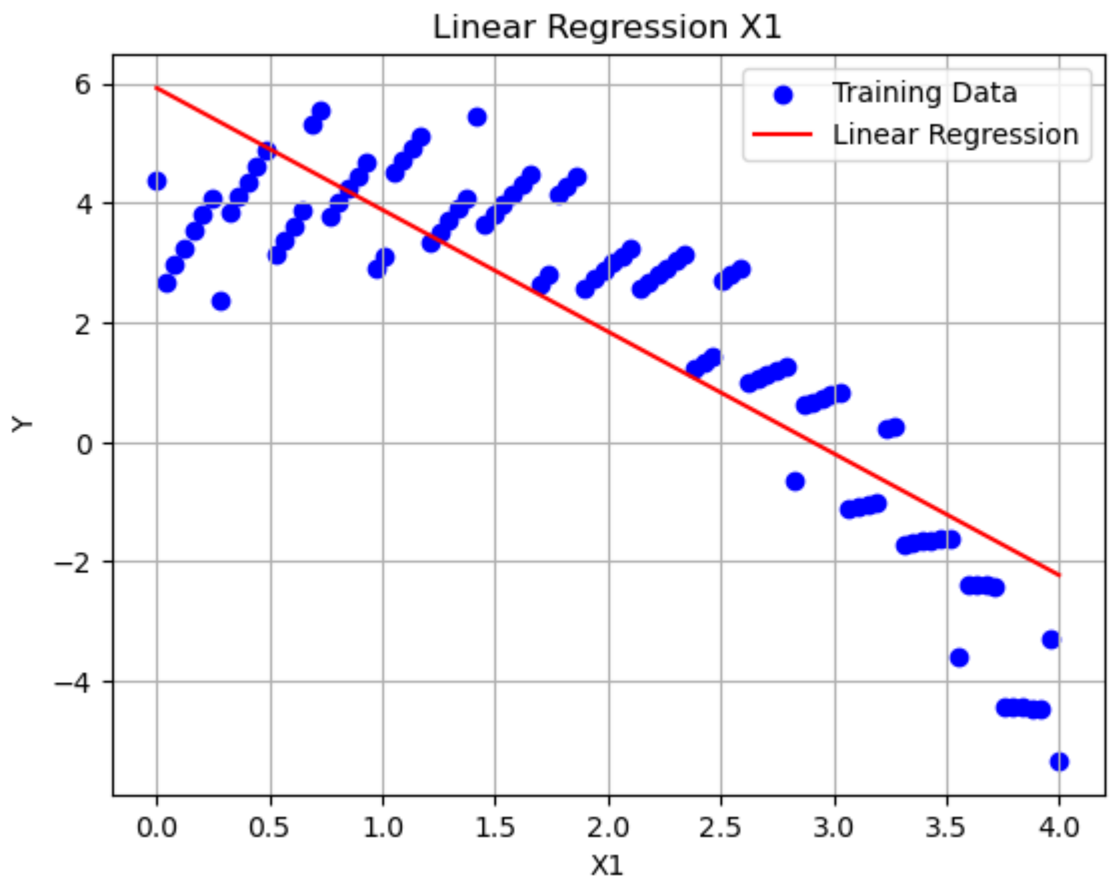
In [9]:
```
theta = np.zeros(2)                       # Create 2-column vector for th
theta
```

Out[9]:
```
array([0., 0.])
```

In [10]:
```
# Computing J(theta_0, theta_1)
iterations = 1500;
alpha = 0.05;

theta1, cost_history = gradient_descent(X1_feat,Y,theta,alpha,iterations,
print('Final value of [Theta_0, Theta_1] = ', theta1)
print('cost_history =', cost_history)
```

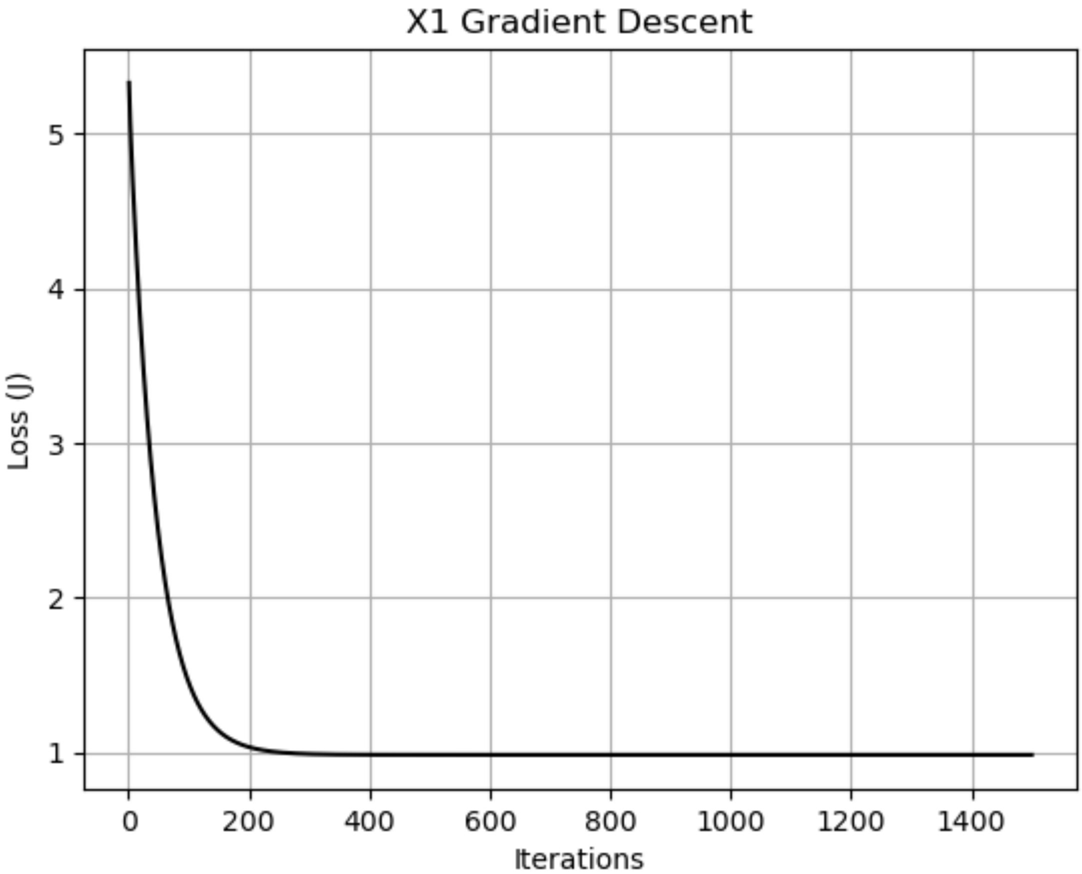
```
Final value of [Theta_0, Theta_1] =  [ 5.9279486  -2.03833651]
cost_history = [5.32852962 5.18676104 5.07204859 ... 0.98499308 0.98499308
0.98499308]
```

In [11]:
```
# X1 Linear Regression Graph
plt.scatter(X1_feat[:,1], Y, color='b', label='Training Data')
plt.plot(X1_feat[:,1], X1_feat.dot(theta1), color='r', label='Linear Regr
plt.xlabel('X1')
plt.ylabel('Y')
plt.title('Linear Regression X1')
plt.legend(); plt.grid()
```

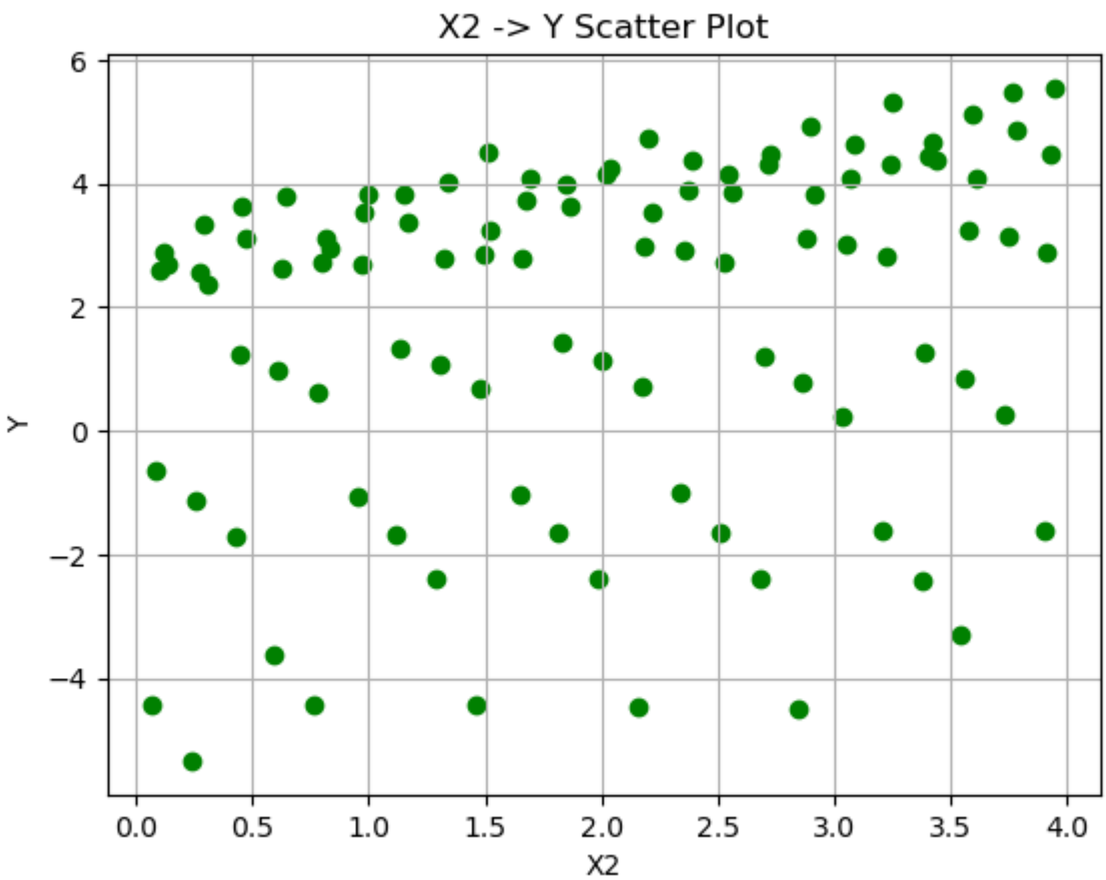


In [12]:

```
# X1 Loss Graph
plt.plot(range(1, iterations + 1), cost_history, color='k')
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Loss (J)')
plt.title('X1 Gradient Descent');
```



In [13]:
```
# X2 Scatter Plot
plt.scatter(X2,Y,color='g')
plt.grid()
plt.xlabel('X2')
plt.ylabel('Y')
plt.title('X2 -> Y Scatter Plot');
```

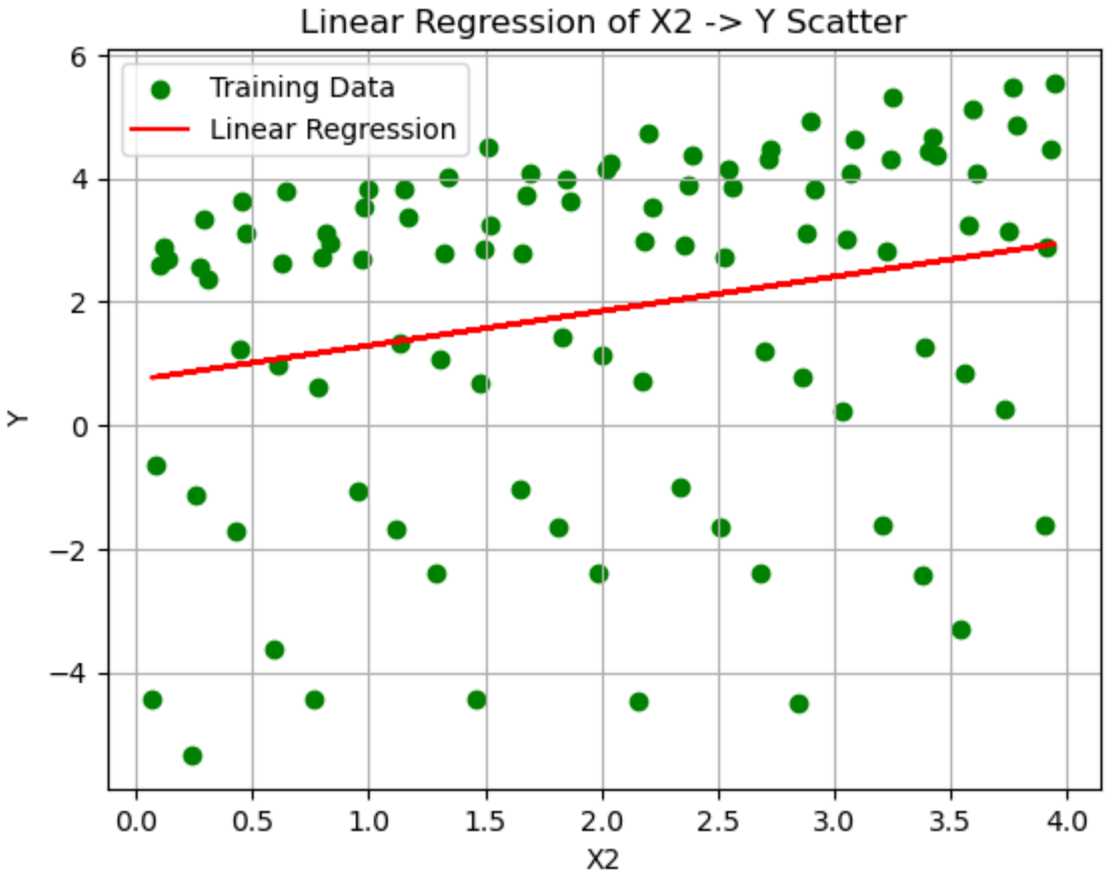


In [14]:
```
# Building X2 Linear Regression Model
X2_0 = np.ones((m,1))
X2_1 = X2.reshape(m,1)
X2_feat = np.hstack((X2_0,X2_1))
```

In [15]:
```
theta = np.zeros(2)
```

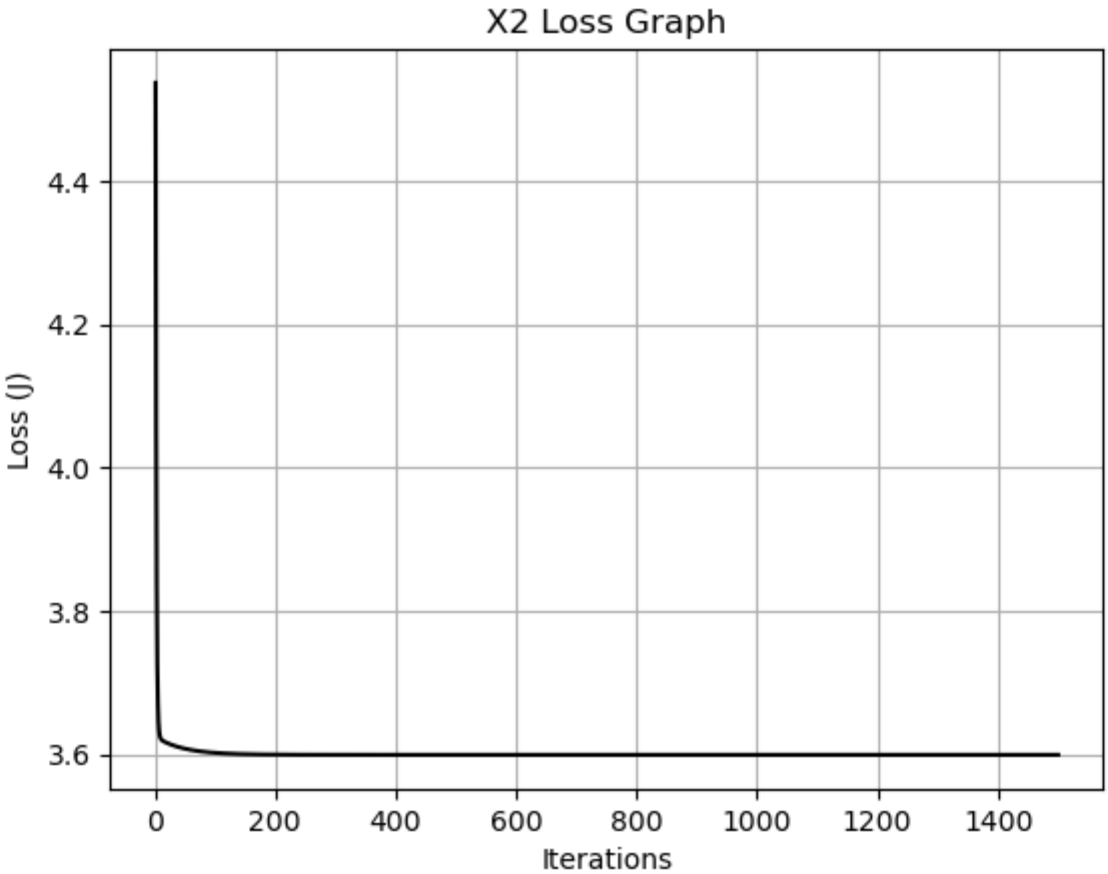
In [16]:
```
# Calculating J(theta_0,theta_1)
iterations = 1500;
alpha = 0.05;

theta2, cost_history = gradient_descent(X2_feat,Y,theta,alpha,iterations,
```

In [17]:
```
# X2 Linear Regression Plot
plt.scatter(X2_feat[:,1], Y, color='g', label='Training Data')
plt.plot(X2_feat[:,1], X2_feat.dot(theta2), color='r', label='Linear Regr
plt.xlabel('X2')
plt.ylabel('Y')
plt.title('Linear Regression of X2 -> Y Scatter')
plt.grid(); plt.legend();
```



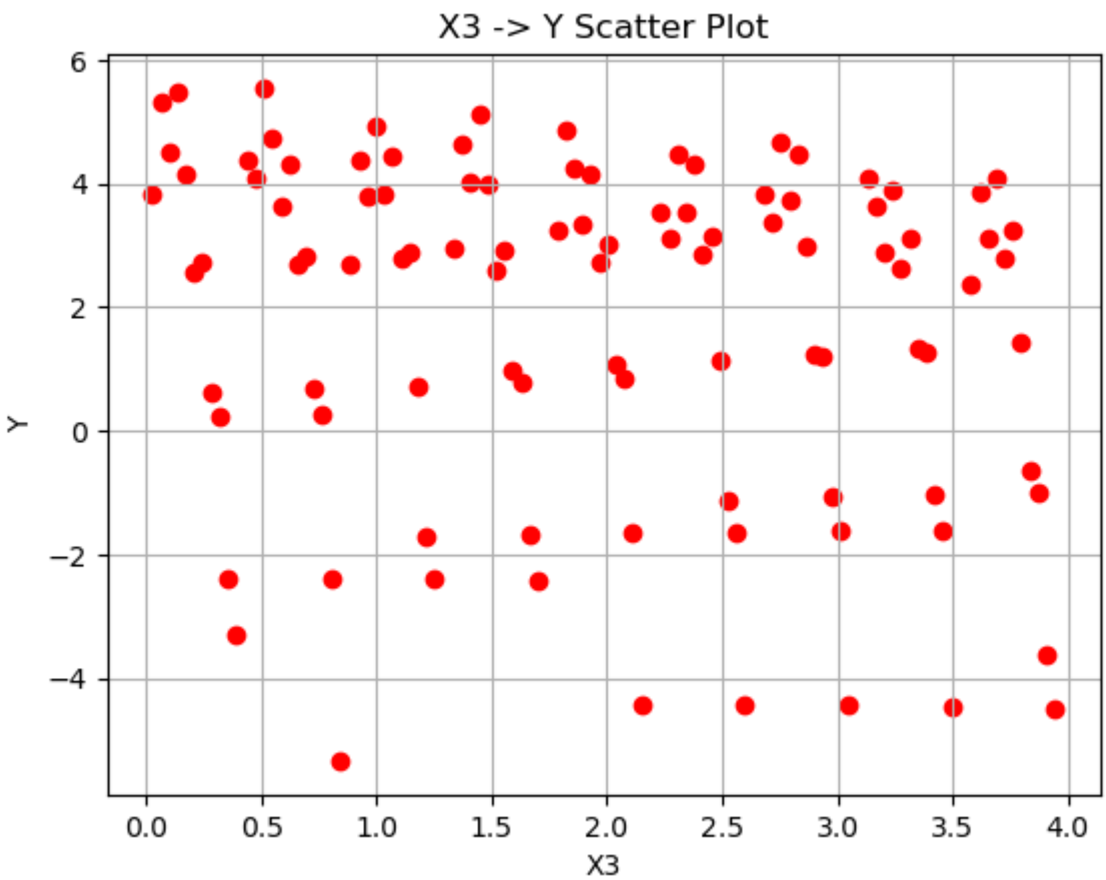
In [18]:

```
# X2 Loss Graph
plt.plot(range(1, iterations + 1), cost_history, color='k')
plt.xlabel('Iterations')
plt.ylabel('Loss (J)')
plt.title('X2 Loss Graph');
plt.grid()
```



In [19]:

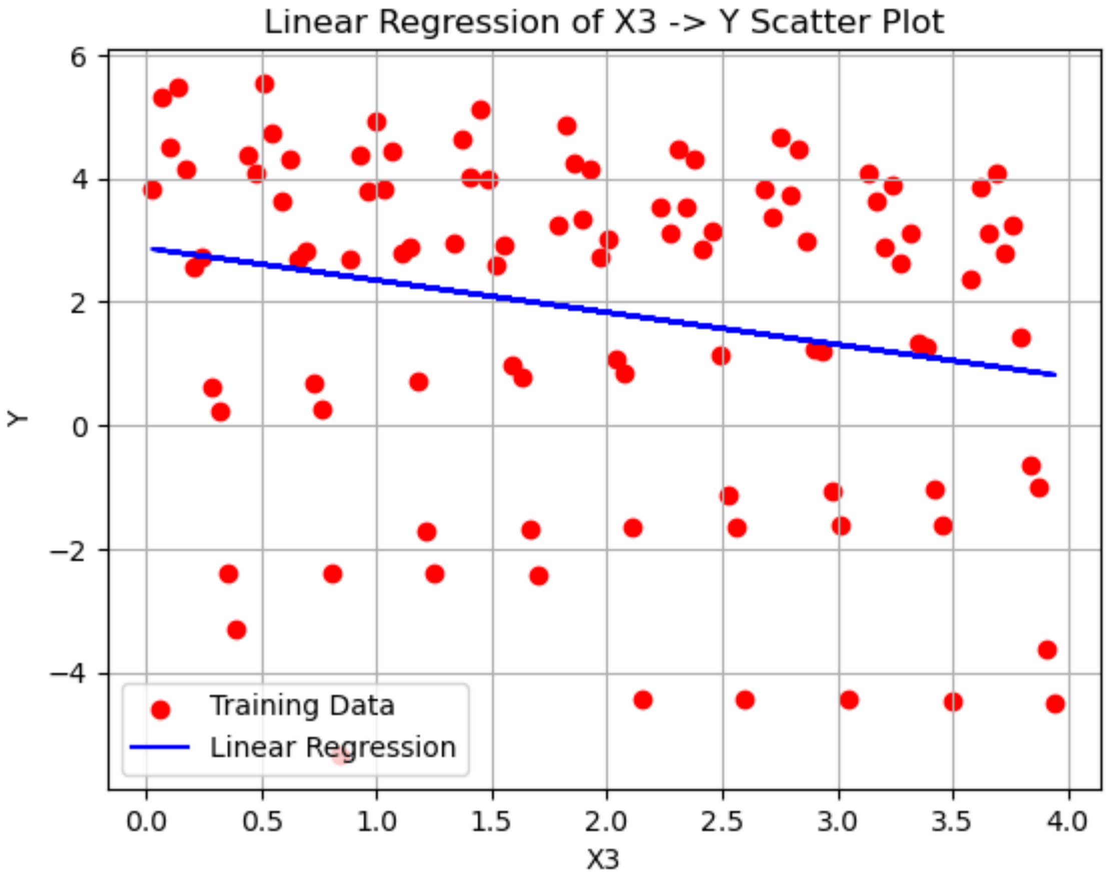
```
# X3 Scatter Plot
plt.scatter(X3, Y, color='r')
plt.xlabel('X3')
plt.ylabel('Y')
plt.title('X3 -> Y Scatter Plot')
plt.grid()
```



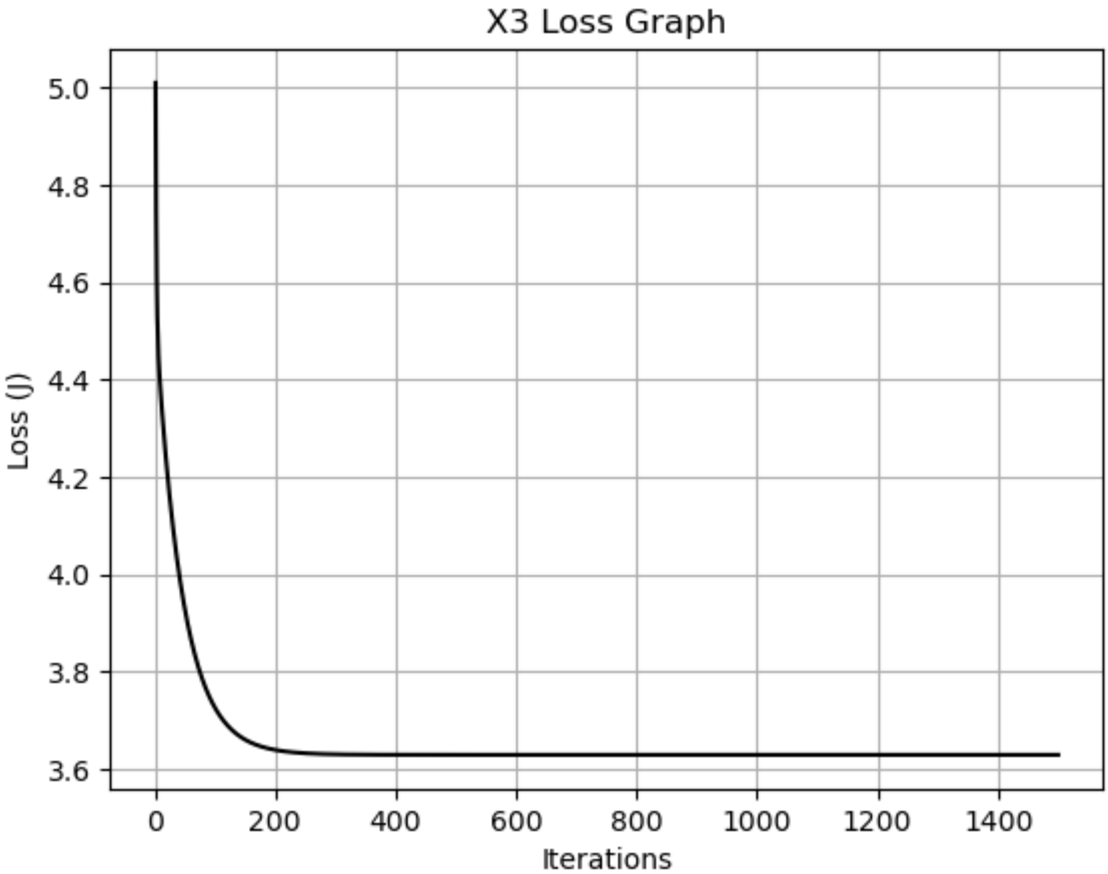
In [20]:
```
# Building X3 Linear Regression Model
X3_0 = np.ones((m,1))
X3_1 = X3.reshape(m,1)
X3_feat = np.hstack((X3_0,X3_1))
theta = np.zeros(2)
```

In [21]:
```
# Calculating Cost Function (J)
iterations = 1500;
alpha = 0.05;
theta3, cost_history = gradient_descent(X3_feat,Y,theta,alpha,iterations,
```

In [22]:
```
# X3 Linear Regression Scatter Plot
plt.scatter(X3_feat[:,1], Y, color='r', label='Training Data')
plt.plot(X3_feat[:,1],X3_feat.dot(theta3), color='b', label='Linear Regre
plt.xlabel('X3')
plt.ylabel('Y')
plt.title('Linear Regression of X3 -> Y Scatter Plot')
plt.grid(); plt.legend();
```



In [23]:
```
# X3 Loss Graph
plt.plot(range(1, iterations + 1), cost_history, color='k')
plt.title('X3 Loss Graph')
plt.xlabel('Iterations')
plt.ylabel('Loss (J)')
plt.grid();
```



In [24]:
```
# Problem 2: Multivariable Linear Regression
# Combine all three X's into one matrix
```

In [25]:
```
# Remove output column for easy-access
df1 = df.iloc[:,:-1]
df1
```

Out[25]:

	X1	X2	X3
0	0.000000	3.440000	0.440000
1	0.040404	0.134949	0.888485
2	0.080808	0.829899	1.336970
3	0.121212	1.524848	1.785455
4	0.161616	2.219798	2.233939
...	...	...	...
95	3.838384	1.460202	3.046061
96	3.878788	2.155152	3.494545
97	3.919192	2.850101	3.943030
98	3.959596	3.545051	0.391515
99	4.000000	0.240000	0.840000

100 rows × 3 columns

In [26]:
```
# Init X1, X2, X3
```

```
X1 = df1.values[:,0]
X2 = df1.values[:,1]
X3 = df1.values[:,2]
# Add Bias
X0 = np.ones((m,1))
# Ensure X's are in right shape
X1 = X1.reshape(m,1)
X2 = X2.reshape(m,1)
X3 = X3.reshape(m,1)
# Concatonate
X_full = np.hstack((X0,X1,X2,X3))
```

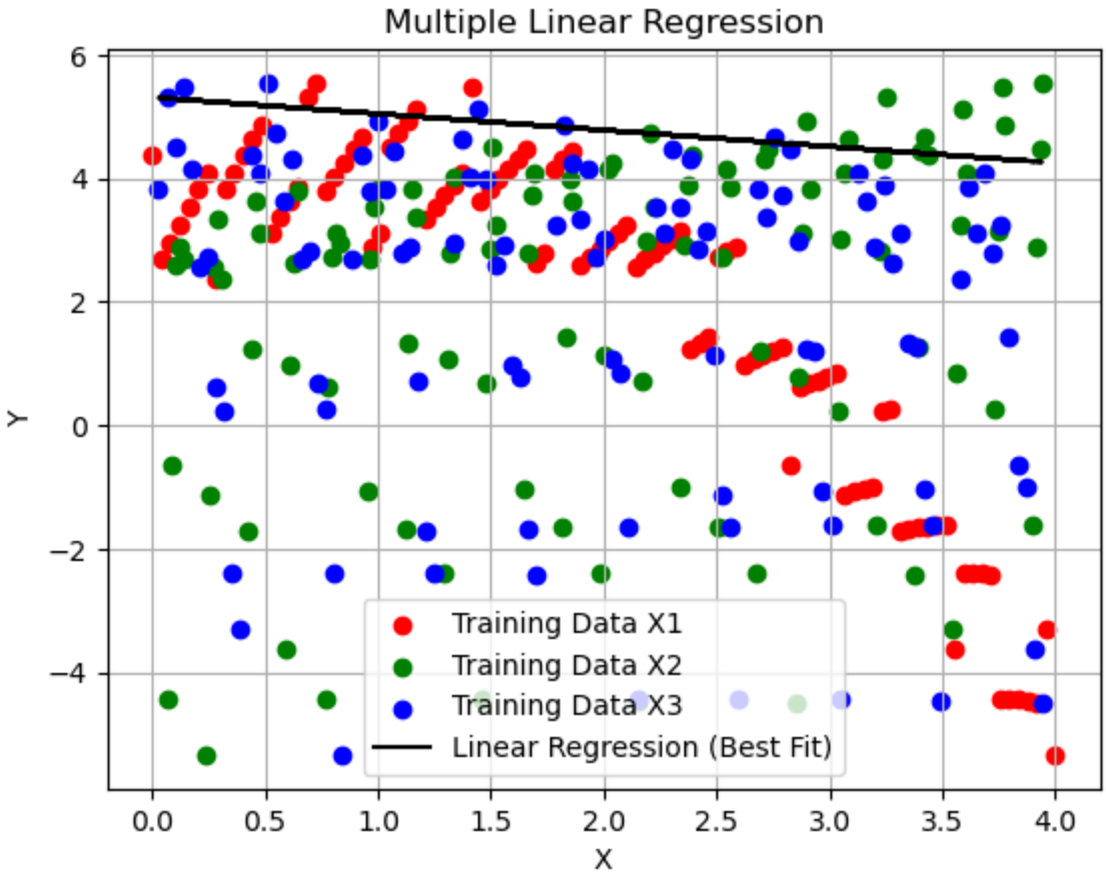
In [27]:
```
theta = np.zeros(4) # 3 variables --> 4 coefficients
theta
```

Out[27]: array([0., 0., 0., 0.])

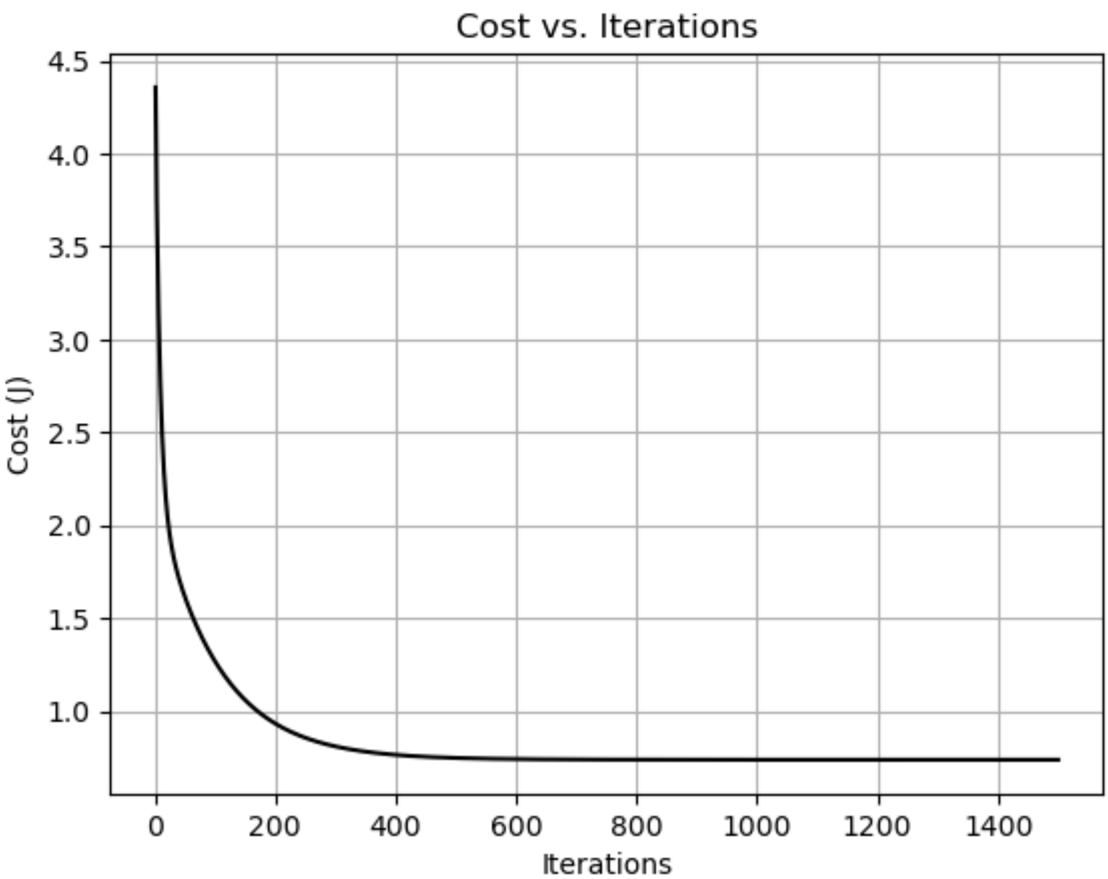
In [28]:
```
# Calculating Cost Function (J)
iterations = 1500; # Arbritrary
alpha = 0.05;      # Between 0.01 and 0.1
theta, cost_history = gradient_descent(X_full,Y,theta,alpha,iterations,m)
theta
```

Out[28]: array([5.31128136, -2.0033116 , 0.5330402 , -0.26517886])

In [29]:
```
plt.scatter(X_full[:,1], Y, color='r', label='Training Data X1')
plt.scatter(X_full[:,2], Y, color='g', label='Training Data X2')
plt.scatter(X_full[:,3], Y, color='b', label='Training Data X3')

# X1 Line with y-intercept
# plt.plot(X_full[:,1],X_full[:,0].dot(theta[0])+X_full[:,1].dot(theta[1]

# X2 Line with y-intercept
# plt.plot(X_full[:,2],X_full[:,0].dot(theta[0])+X_full[:,2].dot(theta[2]

# X3 Line with y-intercept (CHOSEN BEST FIT)
plt.plot(X_full[:,3],X_full[:,0].dot(theta[0])+X_full[:,3].dot(theta[3]),

plt.grid(); plt.legend();
plt.xlabel('X'); plt.ylabel('Y');
plt.title('Multiple Linear Regression');
```



In [30]:
```python
plt.plot(range(1, iterations + 1), cost_history, color='k');
plt.grid();
plt.xlabel('Iterations'); plt.ylabel('Cost (J)');
plt.title('Cost vs. Iterations');
```



```python
# Collection of necessary functions

import numpy as np

def compute_cost(x,y,theta,m):
"""
Input:
--------------
x: 2D array (input feature array)
m= number of training samples
n= number of input features (including the column of all 1's)
y: 1D array of target values for each sample. Dimensions (1 x m)
theta: 1D array of fitting weights. Dimensions (1 x n)
Output:
--------------
J: Scalar value 0 <= J <= 1
"""
predictions = x.dot(theta)
errors = np.subtract(predictions,y)
sqr_errors = np.square(errors)
J = 1 / (2*m) * np.sum(sqr_errors)
return J

def gradient_descent(x,y,theta,alpha,iterations,m):
"""
Input:
--------------
x: 2D array
m= number of training samples
n= number of features (including column full of ones)
y: 1D array of labels/target values for each training example. Dimensions (m x 1)
theta: 1D array of weights. Dimensions (1 x n)
alpha: Learning rate. Scalar value between 0.1 and 0.01
iterations: Num of iterations. Scalar value.
Output:
---------------
theta: Final Value. 1D array of fitting weights. Dimensions (1 x n)
cost_history: contains value of cost per iteration. 1D array, dimensions (m x 1)
"""
cost_history = np.zeros(iterations) # Init. Mat.
for i in range(iterations): # For each iteration...
predictions = x.dot(theta) # Change predict via current theta.
errors = np.subtract(predictions,y) # Calculate errors
sum_delta = (alpha / m) * x.transpose().dot(errors) # Compute the change in theta.
theta = theta - sum_delta # Calculate new theta value
cost_history[i] = compute_cost(x,y,theta,m) # Find new cost value, then repeat
return theta, cost_history
```