In [50]:

```
#PROBLEM 1

import numpy as np
import pandas as pd


data = pd.read_csv("D3.csv")


x1 = data['X1'].values
x2 = data['X2'].values
x3 = data['X3'].values
Y = data['Y'].values


num_iterations = 1000
learning_rates = 0.1


theta_x1 = np.zeros(2)  # Two parameters: theta0 and theta1
theta_x2 = np.zeros(2)
theta_x3 = np.zeros(2)

# Gradient Descent Function
def gradient_descent(x, y, theta, learning_rate, num_iterations):
    m = len(y)
    for iteration in range(num_iterations):
        error = np.dot(x, theta) - y
        gradient = (1/m) * np.dot(x.T, error)
        theta -= learning_rate * gradient
    return theta


   # For x1
X_x1 = np.column_stack((np.ones(len(x1)), x1))
theta_x1 = gradient_descent(X_x1, Y, theta_x1, lr, num_iterations)

    # For x2
X_x2 = np.column_stack((np.ones(len(x2)), x2))
theta_x2 = gradient_descent(X_x2, Y, theta_x2, lr, num_iterations)

    # For x3
X_x3 = np.column_stack((np.ones(len(x3)), x3))
theta_x3 = gradient_descent(X_x3, Y, theta_x3, lr, num_iterations)


print("Results for x1:")
print("Theta:", theta_x1)

print("\nResults for x2:")
print("Theta:", theta_x2)

print("\nResults for x3:")
print("Theta:", theta_x3)
```

```
Results for x1:
Theta: [ 5.29294728 -1.79119079]

Results for x2:
Theta: [0.68701995 0.57669462]

Results for x3:
Theta: [ 2.59118664 -0.40965135]
```

Regression Models ->

For x1:

Y = 5.29294728 - 1.79119079 * x1

For x2:

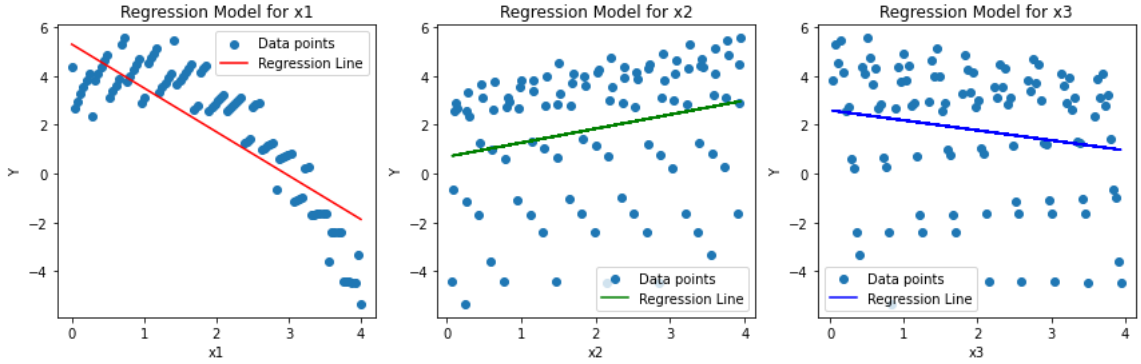
Y = 0.68701995 + 0.57669462 * x2

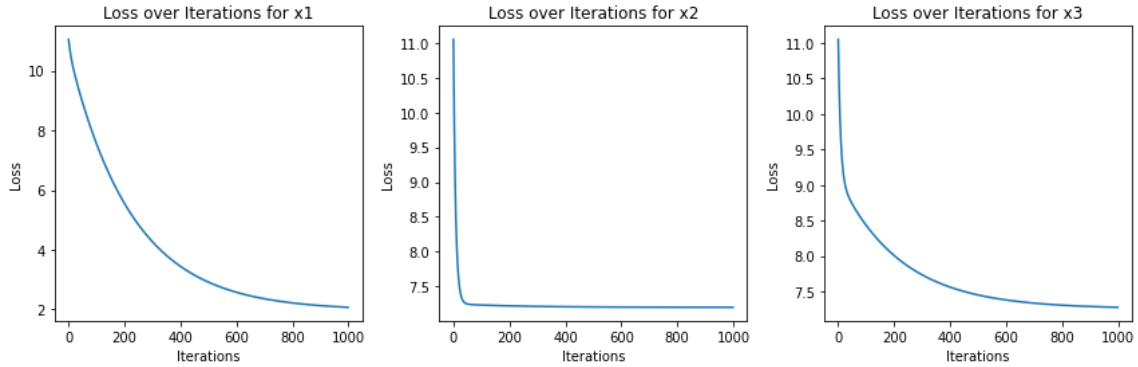
For x3:

Y = 2.59118664 - 0.4096513 * x3

In [52]:

```
# Plot the regression models
plt.figure(figsize=(12, 4))
plt.subplot(131)
plt.scatter(x1, Y, label='Data points')
plt.plot(x1, theta_x1[0] + theta_x1[1] * x1, color='red', label='Regression Line')
plt.xlabel('x1')
plt.ylabel('Y')
plt.title('Regression Model for x1')
plt.legend()

plt.subplot(132)
plt.scatter(x2, Y, label='Data points')
plt.plot(x2, theta_x2[0] + theta_x2[1] * x2, color='green', label='Regression Line')
plt.xlabel('x2')
plt.ylabel('Y')
plt.title('Regression Model for x2')
plt.legend()

plt.subplot(133)
plt.scatter(x3, Y, label='Data points')
plt.plot(x3, theta_x3[0] + theta_x3[1] * x3, color='blue', label='Regression Line')
plt.xlabel('x3')
plt.ylabel('Y')
plt.title('Regression Model for x3')
plt.legend()

plt.tight_layout()
plt.show()

# Plot the loss over iterations
plt.figure(figsize=(12, 4))
plt.subplot(131)
plt.plot(range(num_iterations), loss_x1)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss over Iterations for x1')

plt.subplot(132)
plt.plot(range(num_iterations), loss_x2)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss over Iterations for x2')

plt.subplot(133)
plt.plot(range(num_iterations), loss_x3)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss over Iterations for x3')

plt.tight_layout()
plt.show()
```





In [53]:

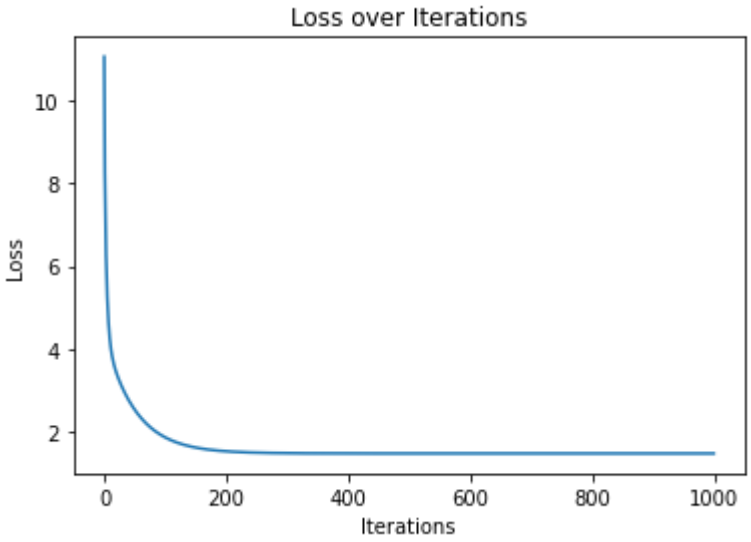
```
# Print the final loss values for each variable
final_loss_x1 = loss_x1[-1]
final_loss_x2 = loss_x2[-1]
final_loss_x3 = loss_x3[-1]

# Compare and report which variable has the lowest loss
if final_loss_x1 < final_loss_x2 and final_loss_x1 < final_loss_x3:
    print("x1 has the lowest loss for explaining Y.")
elif final_loss_x2 < final_loss_x1 and final_loss_x2 < final_loss_x3:
    print("x2 has the lowest loss for explaining Y.")
else:
    print("x3 has the lowest loss for explaining Y.")
```

```
x1 has the lowest loss for explaining Y.
```

In [55]:

```
#PROBLEM 2

# Combine all features into one matrix
X = np.column_stack((np.ones(len(x1)), x1, x2, x3))  # Add a column of ones for the bias

num_iterations = 1000
learning_rate = 0.1

theta = np.zeros(4)  # Four parameters: theta0, theta1, theta2, theta3

loss_values = []

# Gradient Descent Function
def gradient_descent(X, Y, theta, learning_rate, num_iterations, loss_list):
    m = len(Y)
    for iteration in range(num_iterations):
        error = np.dot(X, theta) - Y
        gradient = (1/m) * np.dot(X.T, error)
        theta -= learning_rate * gradient

        # Calculate the mean squared error (MSE) and append it to the loss list
        mse = np.mean((error ** 2))
        loss_list.append(mse)
    return theta


final_theta = gradient_descent(X, Y, theta, learning_rate, num_iterations, loss_values)

print("Final Parameters:", final_theta)

plt.plot(range(num_iterations), loss_values)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss over Iterations')
plt.show()
```

```
Final Parameters: [ 5.31393577 -2.00368658  0.53260157 -0.26556795]
```



In [56]:

```
loss_values[-1]
```

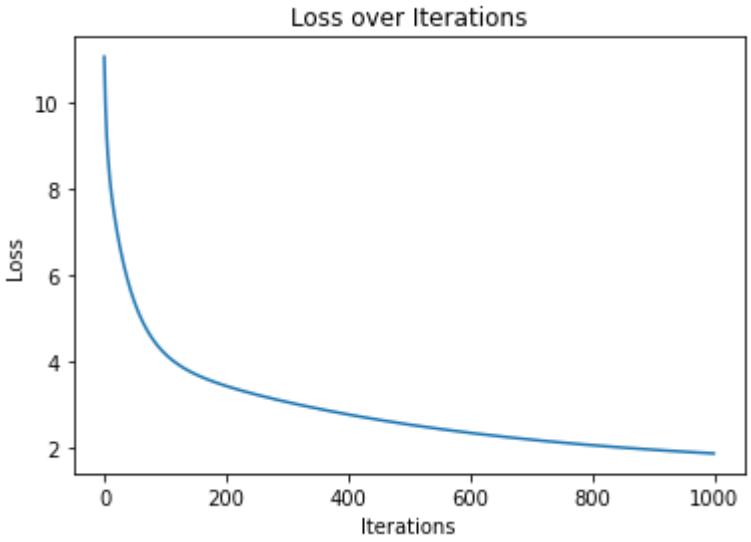
Out[56]:

```
1.4769284889567649
```

In [58]:

```
#PROBLEM 2

# Combine all features into one matrix
X = np.column_stack((np.ones(len(x1)), x1, x2, x3))  # Add a column of ones for the bias

num_iterations = 1000
learning_rate = 0.01

theta = np.zeros(4)  # Four parameters: theta0, theta1, theta2, theta3

loss_values = []

# Gradient Descent Function
def gradient_descent(X, Y, theta, learning_rate, num_iterations, loss_list):
    m = len(Y)
    for iteration in range(num_iterations):
        error = np.dot(X, theta) - Y
        gradient = (1/m) * np.dot(X.T, error)
        theta -= learning_rate * gradient

        # Calculate the mean squared error (MSE) and append it to the loss list
        mse = np.mean((error ** 2))
        loss_list.append(mse)
    return theta


final_theta = gradient_descent(X, Y, theta, learning_rate, num_iterations, loss_values)

print("Final Parameters:", final_theta)

plt.plot(range(num_iterations), loss_values)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss over Iterations')
plt.show()
```

```
Final Parameters: [ 3.39929705 -1.73320582  0.84898275  0.0150737 ]
```



In [47]:

```
loss_values[-1]
```

Out[47]:

```
1.8683471696613532
```

In [59]:

```
final_theta = np.array([5.31393577, -2.00368658, 0.53260157, -0.26556795])

new_x_values = np.array([1, 1, 1])

new_x_values = np.insert(new_x_values, 0, 1)

prediction = np.dot(new_x_values, final_theta)

print(f"Prediction Y for (X1=1, X2=1, X3=1): {prediction}")
```

```
Prediction Y for (X1=1, X2=1, X3=1): 3.5772828099999994
```

In [60]:

```
new_x_values = np.array([2, 0, 4])

new_x_values = np.insert(new_x_values, 0, 1)

prediction = np.dot(new_x_values, final_theta)

print(f"Prediction Y for (X1=2, X2=0, X3=4): {prediction}")
```

```
Prediction Y for (X1=2, X2=0, X3=4): 0.24429080999999941
```

In [61]:

```
new_x_values = np.array([3, 2, 1])

new_x_values = np.insert(new_x_values, 0, 1)

prediction = np.dot(new_x_values, final_theta)

print(f"Prediction Y for (X1=3, X2=2, X3=1): {prediction}")
```

```
Prediction Y for (X1=3, X2=2, X3=1): 0.10251121999999924
```

In [ ]:

```

```