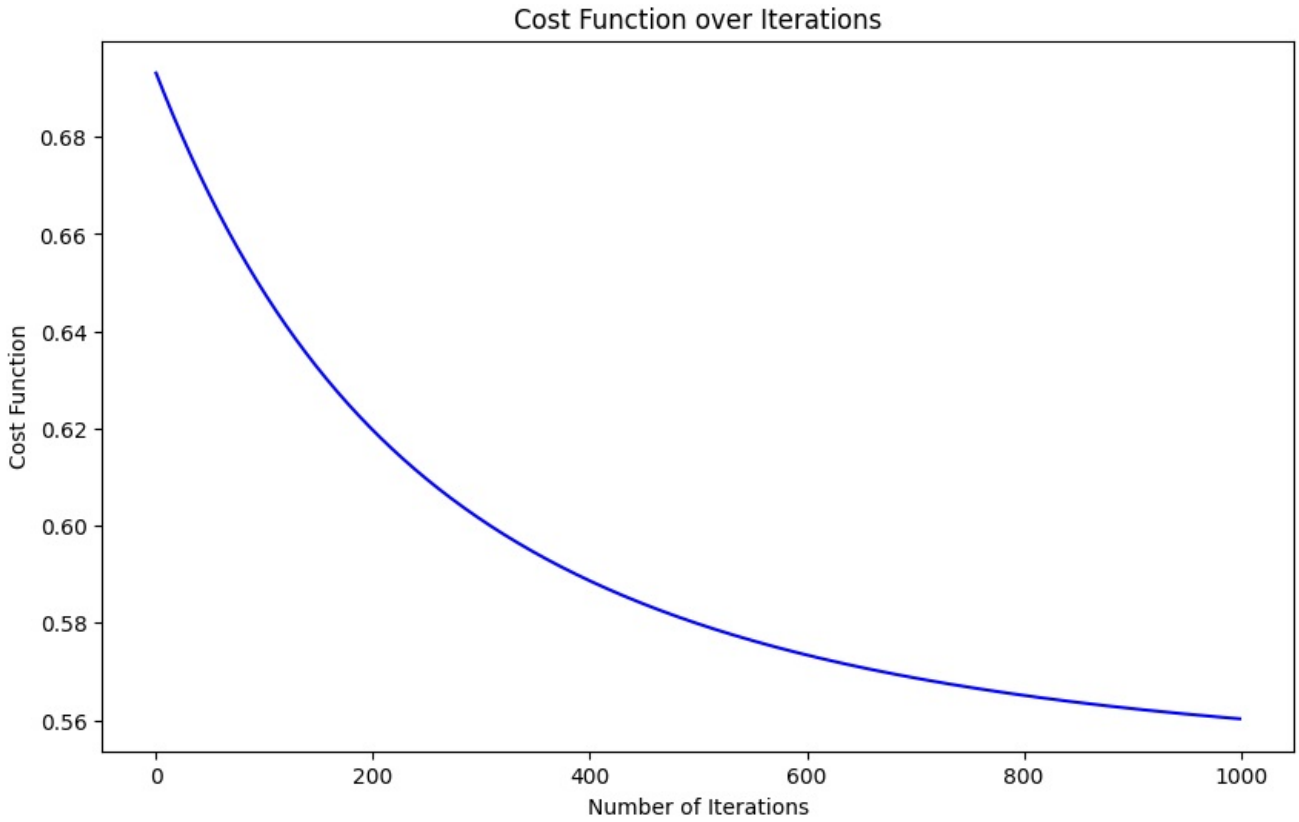
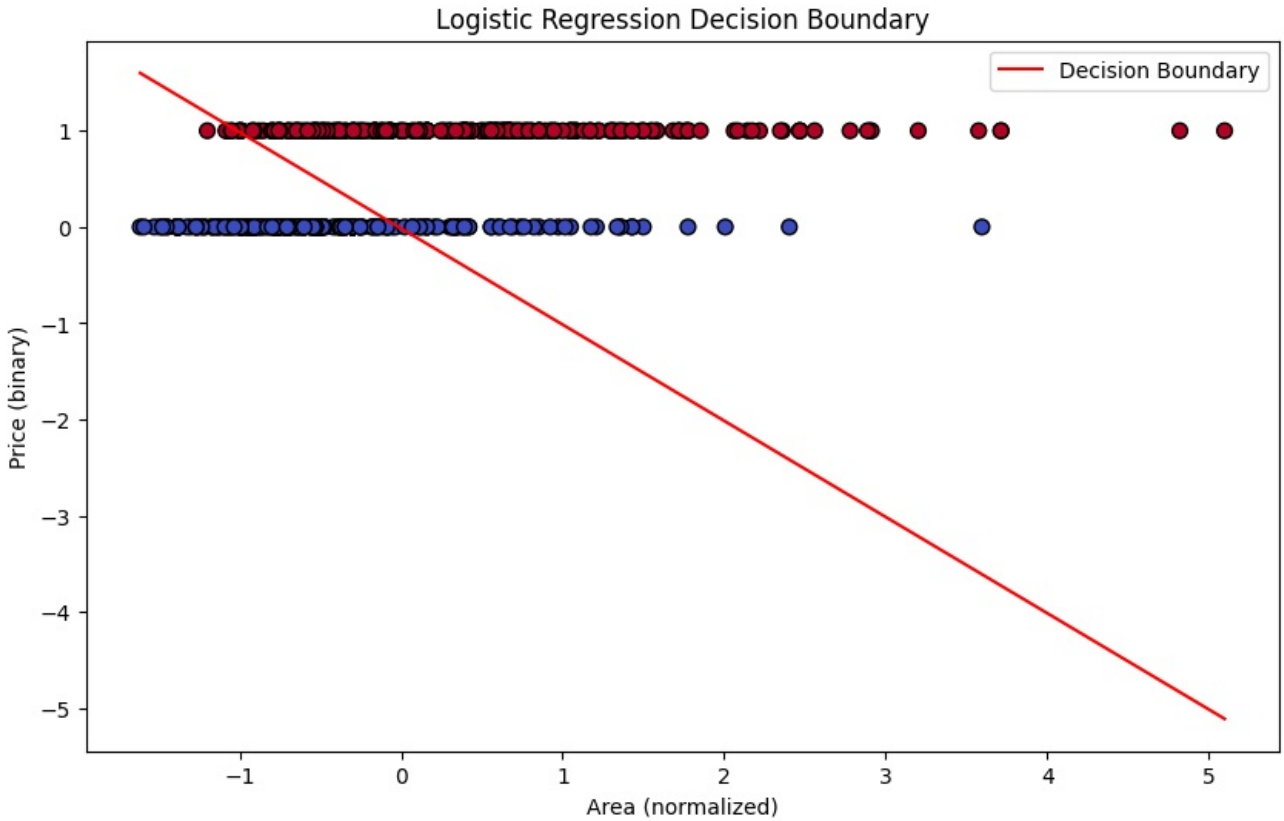
Experiment 4

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv(r'C:\Users\hp\OneDrive\Desktop\housing.csv')  # Update with your file path

# Feature and target selection
X = df[['area']].values  # Using 'area' as the feature
y = (df['price'] > df['price'].median()).astype(int).values  # Binary target based on median price

# Normalize feature
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Add a column of ones to include the intercept (bias) in the model
X_b = np.c_[np.ones((X.shape[0], 1)), X]

# Sigmoid function with clipping
def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -250, 250)))  # Clipping to avoid overflow

# Cost function with clipping
def cost_function(y, y_pred):
    y_pred = np.clip(y_pred, 1e-10, 1 - 1e-10)  # Clipping to avoid log(0)
    m = len(y)
    return (-1/m) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))

# Gradient Descent
def gradient_descent(X, y, learning_rate, num_iterations):
    m, n = X.shape
    theta = np.zeros(n)
    costs = []

    print(f"{'Iteration':<10}{'Theta0':<15}{'Theta1':<15}{'Cost Function'}")
    print("-" * 45)

    for i in range(num_iterations):
        predictions = sigmoid(X.dot(theta))
        errors = predictions - y
        gradients = (1/m) * X.T.dot(errors)
        theta -= learning_rate * gradients
        cost = cost_function(y, predictions)
        costs.append(cost)

        if i % 100 == 0:  # Print every 100 iterations
            print(f"{i:<10}{theta[0]:<15.4f}{theta[1]:<15.4f}{cost:<20.4f}")

    return theta, costs

# Parameters
learning_rate = 0.01
num_iterations = 1000

# Run Gradient Descent
theta, costs = gradient_descent(X_b, y, learning_rate, num_iterations)

print(f"\nOptimal parameters (theta): {theta}")

# Plotting cost function over iterations
plt.figure(figsize=(10, 6))
plt.plot(range(num_iterations), costs, color='blue')
plt.xlabel('Number of Iterations')
plt.ylabel('Cost Function')
plt.title('Cost Function over Iterations')
plt.show()

# Plotting decision boundary
plt.figure(figsize=(10, 6))
plt.scatter(X, y, c=y, cmap='coolwarm', edgecolors='k', s=50)
x_values = np.linspace(X.min(), X.max(), 100)
y_values = (-1 / theta[1]) * (theta[0] + theta[1] * x_values)
plt.plot(x_values, y_values, label='Decision Boundary', color='red')
plt.xlabel('Area (normalized)')
plt.ylabel('Price (binary)')
plt.title('Logistic Regression Decision Boundary')
```

```
plt.legend()
plt.show()
```

```
Iteration Theta0         Theta1         Cost Function
-----------------------------------------------
0         -0.0000        0.0024         0.6931
100       -0.0040        0.2142         0.6479
200       -0.0066        0.3816         0.6196
300       -0.0075        0.5170         0.6012
400       -0.0068        0.6287         0.5886
500       -0.0048        0.7225         0.5798
600       -0.0017        0.8023         0.5734
700       0.0020         0.8709         0.5686
800       0.0063         0.9304         0.5651
900       0.0108         0.9823         0.5623

Optimal parameters (theta): [0.01547946 1.02751984]
```





In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js