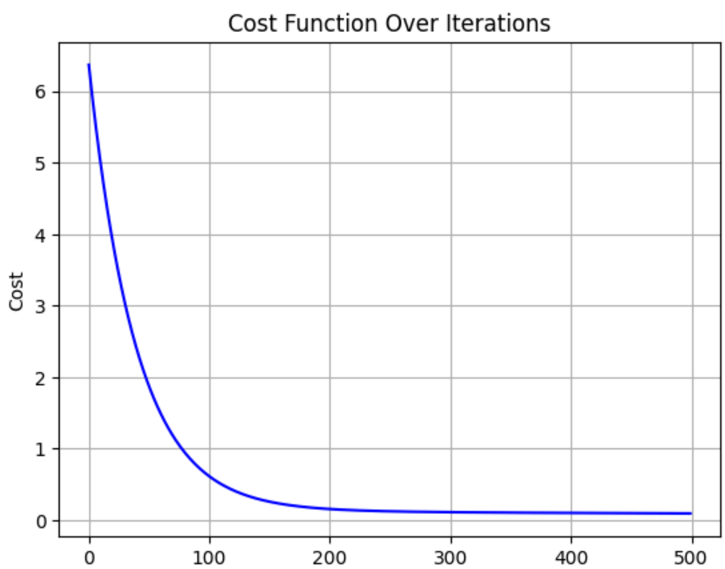
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
m = 2000
n = 2

X = np.random.rand(m, n)
X[:, 0] = 1
true_theta = np.array([2, 3])
y = X.dot(true_theta) + np.random.randn(m) * 0.3

theta = np.zeros(n)
learning_rate = 0.01
num_iterations = 500

def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    errors = predictions - y
    cost = (1 / (2 * m)) * np.sum(errors ** 2)
    return cost
```

Batch GD

```
def batch_gradient_descent(X, y, theta, learning_rate, num_iterations):
    m = len(y)
    cost_history = np.zeros(num_iterations)
    for i in range(num_iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        gradients = (1 / m) * X.T.dot(errors)
        theta = theta - learning_rate * gradients
        cost_history[i] = compute_cost(X, y, theta)
    return theta, cost_history

theta, cost_history = batch_gradient_descent(X, y, theta, learning_rate, num_iterations)

print("Theta:", theta)
print("Final cost:", cost_history[-1])

plt.plot(range(num_iterations), cost_history, color='blue')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
plt.title('Cost Function Over Iterations')
plt.grid(True)
plt.show()
```

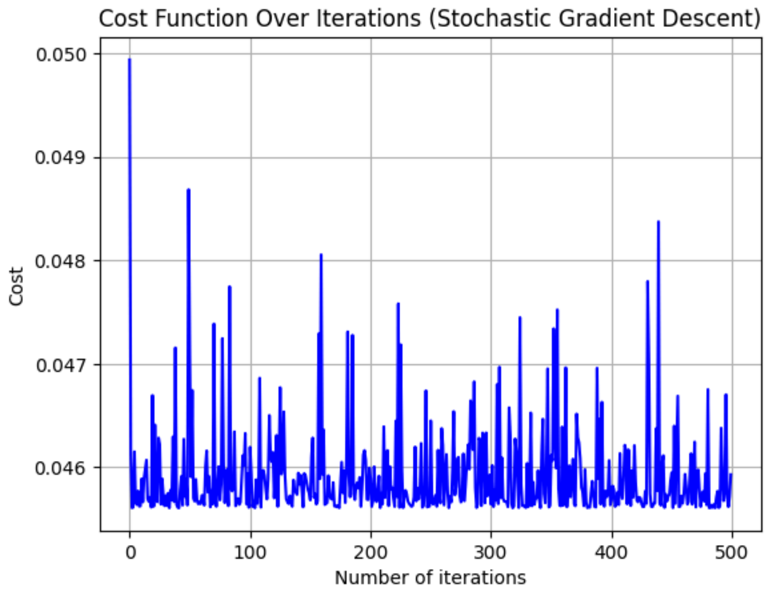
```
Theta: [2.56986733 1.91440276]
Final cost: 0.09499329707514159
```



SGD

```
def stochastic_gradient_descent(X, y, theta, learning_rate, num_iterations):
    m = len(y)
    cost_history = np.zeros(num_iterations)

    for i in range(num_iterations):
        for j in range(m):
            index = np.random.randint(m)
            X_i = X[index:index + 1]
            y_i = y[index:index + 1]

            predictions = X_i.dot(theta)
            errors = predictions - y_i
            gradients = X_i.T.dot(errors)
            theta = theta - learning_rate * gradients

        cost_history[i] = compute_cost(X, y, theta)

    return theta, cost_history

theta, cost_history = stochastic_gradient_descent(X, y, theta, learning_rate, num_iterations)

print("Theta:", theta)
print("Final cost:", cost_history[-1])

plt.plot(range(num_iterations), cost_history, color='blue')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
plt.title('Cost Function Over Iterations (Stochastic Gradient Descent)')
plt.grid(True)
plt.show()

```

```
Theta: [1.96770767 3.00259837]
Final cost: 0.0459303724047596
```



Mini Batch GD

```
def mini_batch_gradient_descent(X, y, theta, learning_rate, num_iterations, batch_size):
    m = len(y)
    cost_history = np.zeros(num_iterations)
    for i in range(num_iterations):
        indices = np.random.permutation(m)
        X_shuffled = X[indices]
        y_shuffled = y[indices]
        for start in range(0, m, batch_size):
            end = min(start + batch_size, m)
            X_mini_batch = X_shuffled[start:end]
            y_mini_batch = y_shuffled[start:end]
            predictions = X_mini_batch.dot(theta)
```

```
            errors = predictions - y_mini_batch
            gradients = (1 / len(y_mini_batch)) * X_mini_batch.T.dot(errors)
            theta = theta - learning_rate * gradients
        cost_history[i] = compute_cost(X, y, theta)
    return theta, cost_history

batch_size = 64
theta, cost_history = mini_batch_gradient_descent(X, y, theta, learning_rate, num_iterations, batch_size)
print("Theta:", theta)
print("Final cost:", cost_history[-1])

plt.plot(range(num_iterations), cost_history, color='blue')
plt.xlabel('Number of iterations')
plt.ylabel('Cost')
plt.title('Cost Function Over Iterations (Mini-Batch Gradient Descent)')
plt.grid(True)
plt.show()
```

```
Theta: [1.99291503 3.00228372]
Final cost: 0.04560672672255235
```