```
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt


# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Use only the first two features
y = iris.target


# Normalize the input features
X_norm = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Add a column of ones for the intercept term
X_norm = np.hstack((np.ones((X_norm.shape[0], 1)), X_norm))
df=X_norm


df
```

```
array([[ 1.        , -0.05250608, -1.05276654],
       [ 1.        , -1.02184904, -1.74335684],
       [ 1.        , -0.29484182, -0.82256978],
       [ 1.        , -0.17367395, -0.13197948],
       [ 1.        , -0.17367395, -0.36217625],
       [ 1.        ,  0.4321654 , -0.36217625],
       [ 1.        , -0.90068117, -1.28296331],
       [ 1.        , -0.17367395, -0.59237301],
       [ 1.        ,  0.55333328,  0.55861082],
       [ 1.        , -0.05250608, -0.82256978],
       [ 1.        ,  1.52267624, -0.13197948],
       [ 1.        ,  0.55333328, -0.36217625],
       [ 1.        ,  0.79566902, -0.13197948],
       [ 1.        ,  2.12851559, -0.13197948],
       [ 1.        , -1.14301691, -1.28296331],
       [ 1.        ,  1.76501198, -0.36217625],
       [ 1.        ,  1.03800476, -1.28296331],
       [ 1.        ,  1.64384411,  1.24920112],
       [ 1.        ,  0.79566902,  0.32841405],
       [ 1.        ,  0.67450115, -0.82256978],
       [ 1.        ,  1.15917263, -0.13197948],
       [ 1.        , -0.17367395, -1.28296331],
       [ 1.        , -0.05250608, -0.59237301],
       [ 1.        ,  0.67450115,  0.32841405],
       [ 1.        ,  0.79566902, -0.13197948],
       [ 1.        ,  2.24968346,  1.70959465],
       [ 1.        ,  2.24968346, -1.05276654],
       [ 1.        ,  0.18982966, -1.97355361],
       [ 1.        ,  1.2803405 ,  0.32841405],
       [ 1.        , -0.29484182, -0.59237301],
       [ 1.        ,  2.24968346, -0.59237301],
       [ 1.        ,  0.55333328, -0.82256978],
       [ 1.        ,  1.03800476,  0.55861082],
       [ 1.        ,  1.64384411,  0.32841405],
       [ 1.        ,  0.4321654 , -0.59237301],
       [ 1.        ,  0.31099753, -0.13197948],
       [ 1.        ,  0.67450115, -0.59237301],
       [ 1.        ,  1.64384411, -0.13197948],
       [ 1.        ,  1.88617985, -0.59237301],
       [ 1.        ,  2.4920192 ,  1.70959465],
       [ 1.        ,  0.67450115, -0.59237301],
       [ 1.        ,  0.55333328, -0.59237301],
       [ 1.        ,  0.31099753, -1.05276654],
       [ 1.        ,  2.24968346, -0.13197948],
       [ 1.        ,  0.55333328,  0.78880759],
       [ 1.        ,  0.67450115,  0.09821729],
       [ 1.        ,  0.18982966, -0.13197948],
       [ 1.        ,  1.2803405 ,  0.09821729],
       [ 1.        ,  1.03800476,  0.09821729],
       [ 1.        ,  1.2803405 ,  0.09821729],
       [ 1.        , -0.05250608, -0.82256978],
       [ 1.        ,  1.15917263,  0.32841405],
       [ 1.        ,  1.03800476,  0.55861082],
       [ 1.        ,  1.03800476, -0.13197948],
       [ 1.        ,  0.55333328, -1.28296331],
       [ 1.        ,  0.79566902, -0.13197948],
       [ 1.        ,  0.4321654 ,  0.78880759],
       [ 1.        ,  0.06866179, -0.13197948]])
```

```
# Initialize parameters
theta = np.zeros((3, 1))  # 3 parameters including intercept
alpha = 0.01  # learning rate
num_iterations = 1000
```

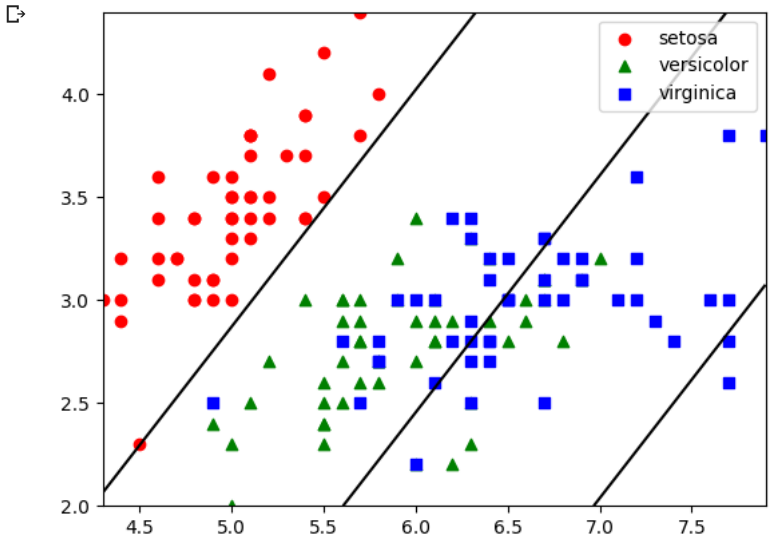
```
# Perform gradient descent
m = len(y)
for i in range(num_iterations):
    # Calculate predicted values
    y_pred = X_norm.dot(theta)

    # Calculate the error (difference between predicted and actual values)
    error = y_pred - y.reshape(-1, 1)

    # Update parameters
    theta -= alpha * (1/m) * X_norm.T.dot(error)

    # Print the cost (mean squared error) every 100 iterations
    if i % 100 == 0:
        cost = (1/(2*m)) * np.sum((y_pred - y.reshape(-1, 1))**2)
        print(f"Cost after iteration {i}: {cost}")
```

```
Cost after iteration 0: 0.8333333333333334
Cost after iteration 100: 0.18543549255118577
Cost after iteration 200: 0.10344460021642783
Cost after iteration 300: 0.09292227028873937
Cost after iteration 400: 0.09155360126443841
Cost after iteration 500: 0.09137324876850643
Cost after iteration 600: 0.09134917989390438
Cost after iteration 700: 0.0913459270241712
Cost after iteration 800: 0.09134548175626253
Cost after iteration 900: 0.09134542000074793
```

```
# Print final parameters
print(f"Final parameters: {theta.ravel()}")
```

```
Final parameters: [ 0.99995683  0.60635416 -0.27708957]
```

```
# Plot the data and the regression line
fig, ax = plt.subplots()
colors = ['red', 'green', 'blue']
markers = ['o', '^', 's']
for i in range(3):
    ax.scatter(X[y==i, 0], X[y==i, 1], color=colors[i], marker=markers[i], label=iris.target_names[i])
x1_range = np.linspace(X[:, 0].min(), X[:, 0].max(), num=10)
x2_range = np.linspace(X[:, 1].min(), X[:, 1].max(), num=10)
X1, X2 = np.meshgrid(x1_range, x2_range)
Y_pred = theta[0] + theta[1]*((X1 - np.mean(X[:, 0]))/np.std(X[:, 0])) + theta[2]*((X2 - np.mean(X[:, 1]))/np.std(X[:, 1]))
ax.contour(X1, X2, Y_pred, levels=[0.5, 1.5, 2.5], colors='black')
ax.legend()
plt.show()
```



✓ 0s completed at 18:34