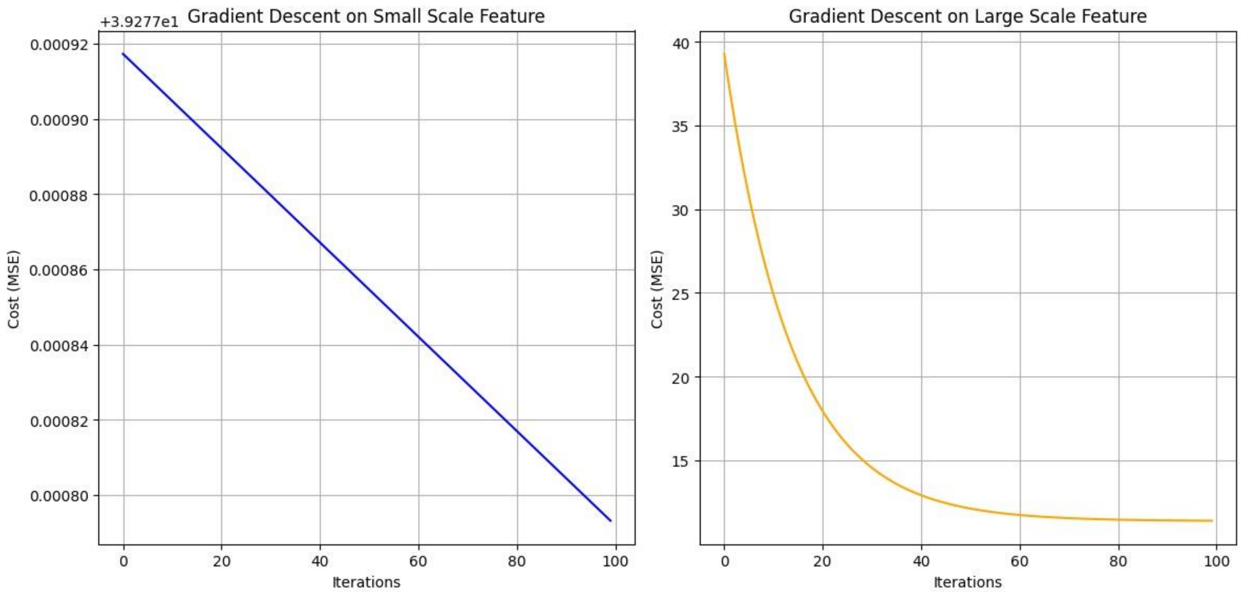
```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(0)

# Generate synthetic dataset with features on different scales
X_small = np.random.rand(100, 1)                    # Small scale
feature (0-1)
X_large = np.random.rand(100, 1) * 1000             # Large scale
feature (0-1000)
X = np.hstack((X_small, X_large))                   # Combine features

# True coefficients for linear regression (for generating target
values)
true_coef = np.array([5, 0.01])
y = X @ true_coef + np.random.randn(100) * 5        # Generate target
variable with noise

# Gradient Descent function def gradient_descent(X, y,
learning_rate, iterations):     n_samples = X.shape[0]
theta = np.zeros(X.shape[1])     cost_history = []
for i in range(iterations):
        y_pred = X @ theta
error = y_pred - y
        theta -= (learning_rate / n_samples) * (X.T @ error)
cost = (1 / (2 * n_samples)) * np.sum(error ** 2)
cost_history.append(cost)     return theta, cost_history

# Run gradient descent on unscaled data with a small learning rate
theta_unscaled, cost_history_unscaled = gradient_descent(X, y,
learning_rate=0.0000001, iterations=100)

# Run gradient descent on the small dataset
theta_small, cost_history_small = gradient_descent(X_small, y,
learning_rate=0.0000001, iterations=100)

# Run gradient descent on the large dataset
theta_large, cost_history_large = gradient_descent(X_large, y,
learning_rate=0.0000001, iterations=100)

# Plot cost history for both datasets
plt.figure(figsize=(12, 6)) # Plot
for small dataset
```

```python
plt.subplot(1, 2, 1)
plt.plot(cost_history_small, color="blue")
plt.title("Gradient Descent on Small Scale Feature")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

# Plot for large dataset
plt.subplot(1, 2, 2)
plt.plot(cost_history_large, color="orange")
plt.title("Gradient Descent on Large Scale Feature")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

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

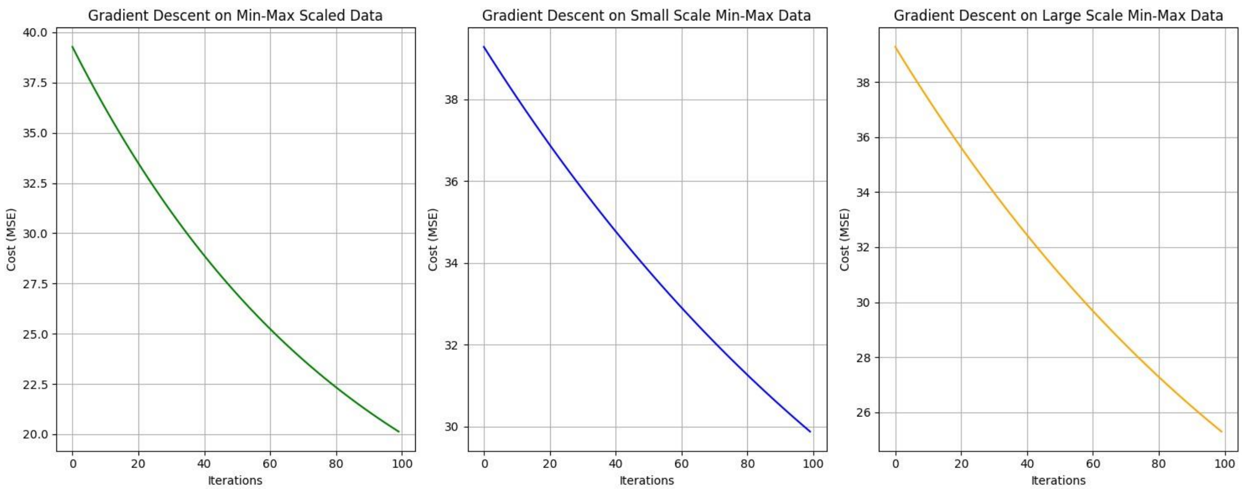
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Set random seed for reproducibility
np.random.seed(0)

# Generate synthetic dataset with features on different scales
X_small = np.random.rand(100, 1)                    # Small scale
feature (0-1)
X_large = np.random.rand(100, 1) * 1000             # Large scale
feature (0-1000)
X = np.hstack((X_small, X_large))                   # Combine features
```


Gradient Descent on Small Scale Feature
+3.9277e1
0.00092
0.00090
0.00088
0.00086
0.00084
0.00082
0.00080
Cost (MSE)
0
20
40
60
80
100
Iterations
Gradient Descent on Large Scale Feature
40
35
30
25
20
15
Cost (MSE)
0
20
40
60
80
100
Iterations

```
# True coefficients for linear regression (for generating target
values)
true_coef = np.array([5, 0.01])
y = X @ true_coef + np.random.randn(100) * 5      # Generate target
variable with noise

# Apply Min-Max scaling to the entire dataset
scaler_minmax = MinMaxScaler()
X_minmax = scaler_minmax.fit_transform(X)

# Apply Min-Max scaling separately to small and large features
scaler_small = MinMaxScaler()
X_small_minmax = scaler_small.fit_transform(X_small)

scaler_large = MinMaxScaler()
X_large_minmax = scaler_large.fit_transform(X_large)

# Gradient Descent function def gradient_descent(X, y,
learning_rate, iterations):     n_samples = X.shape[0]
theta = np.zeros(X.shape[1])     cost_history = []
for i in range(iterations):
        y_pred = X @ theta
error = y_pred - y
        theta -= (learning_rate / n_samples) * (X.T @ error)
cost = (1 / (2 * n_samples)) * np.sum(error ** 2)
cost_history.append(cost)     return theta, cost_history

# Run gradient descent on Min-Max scaled data
theta_minmax, cost_history_minmax = gradient_descent(X_minmax, y,
learning_rate=0.01, iterations=100)
theta_small_minmax, cost_history_small_minmax =
gradient_descent(X_small_minmax, y, learning_rate=0.01,
iterations=100)
theta_large_minmax, cost_history_large_minmax =
gradient_descent(X_large_minmax, y, learning_rate=0.01,
iterations=100)

# Plot cost history for all datasets
plt.figure(figsize=(15, 6))

# Plot for all Min-Max scaled data
plt.subplot(1, 3, 1)
plt.plot(cost_history_minmax, color="green")
plt.title("Gradient Descent on Min-Max Scaled Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
```

```
plt.grid(True)

# Plot for Min-Max scaled small dataset
plt.subplot(1, 3, 2)
plt.plot(cost_history_small_minmax, color="blue")
plt.title("Gradient Descent on Small Scale Min-Max Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

# Plot for Min-Max scaled large dataset
plt.subplot(1, 3, 3)
plt.plot(cost_history_large_minmax, color="orange")
plt.title("Gradient Descent on Large Scale Min-Max Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Set random seed for reproducibility
np.random.seed(0)

# Generate synthetic dataset with features on different scales
X_small = np.random.rand(100, 1)                    # Small scale
feature (0-1)
X_large = np.random.rand(100, 1) * 1000             # Large scale
feature (0-1000)
X = np.hstack((X_small, X_large))                   # Combine features
```

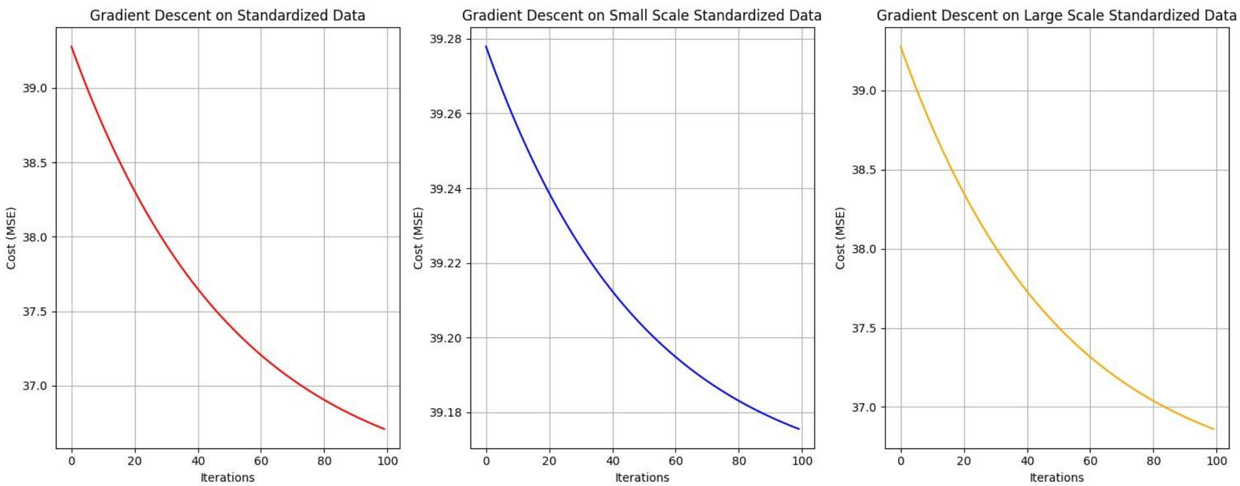


Gradient Descent on Min-Max Scaled Data
Cost (MSE)
Iterations
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
0
20
40
60
80
100
Gradient Descent on Small Scale Min-Max Data
Cost (MSE)
Iterations
38
36
34
32
30
0
20
40
60
80
100
Gradient Descent on Large Scale Min-Max Data
Cost (MSE)
Iterations
38
36
34
32
30
28
26
0
20
40
60
80
100

```
# True coefficients for linear regression (for generating target
values)
true_coef = np.array([5, 0.01])
y = X @ true_coef + np.random.randn(100) * 5      # Generate target
variable with noise

# Apply Standardization to the entire dataset
scaler_standard = StandardScaler()
X_standard = scaler_standard.fit_transform(X)

# Apply Standardization separately to small and large features
scaler_small = StandardScaler()
X_small_standard = scaler_small.fit_transform(X_small)

scaler_large = StandardScaler()
X_large_standard = scaler_large.fit_transform(X_large)

# Gradient Descent function def gradient_descent(X, y,
learning_rate, iterations):     n_samples = X.shape[0]
theta = np.zeros(X.shape[1])     cost_history = []
for i in range(iterations):
        y_pred = X @ theta
error = y_pred - y
        theta -= (learning_rate / n_samples) * (X.T @ error)
cost = (1 / (2 * n_samples)) * np.sum(error ** 2)
cost_history.append(cost)     return theta, cost_history

# Run gradient descent on standardized data
theta_standard, cost_history_standard = gradient_descent(X_standard,
y, learning_rate=0.01, iterations=100)
theta_small_standard, cost_history_small_standard =
gradient_descent(X_small_standard, y, learning_rate=0.01,
iterations=100)
theta_large_standard, cost_history_large_standard =
gradient_descent(X_large_standard, y, learning_rate=0.01,
iterations=100)

# Plot cost history for all datasets
plt.figure(figsize=(15, 6))

# Plot for standardized entire dataset
plt.subplot(1, 3, 1)
plt.plot(cost_history_standard, color="red")
plt.title("Gradient Descent on Standardized Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
```

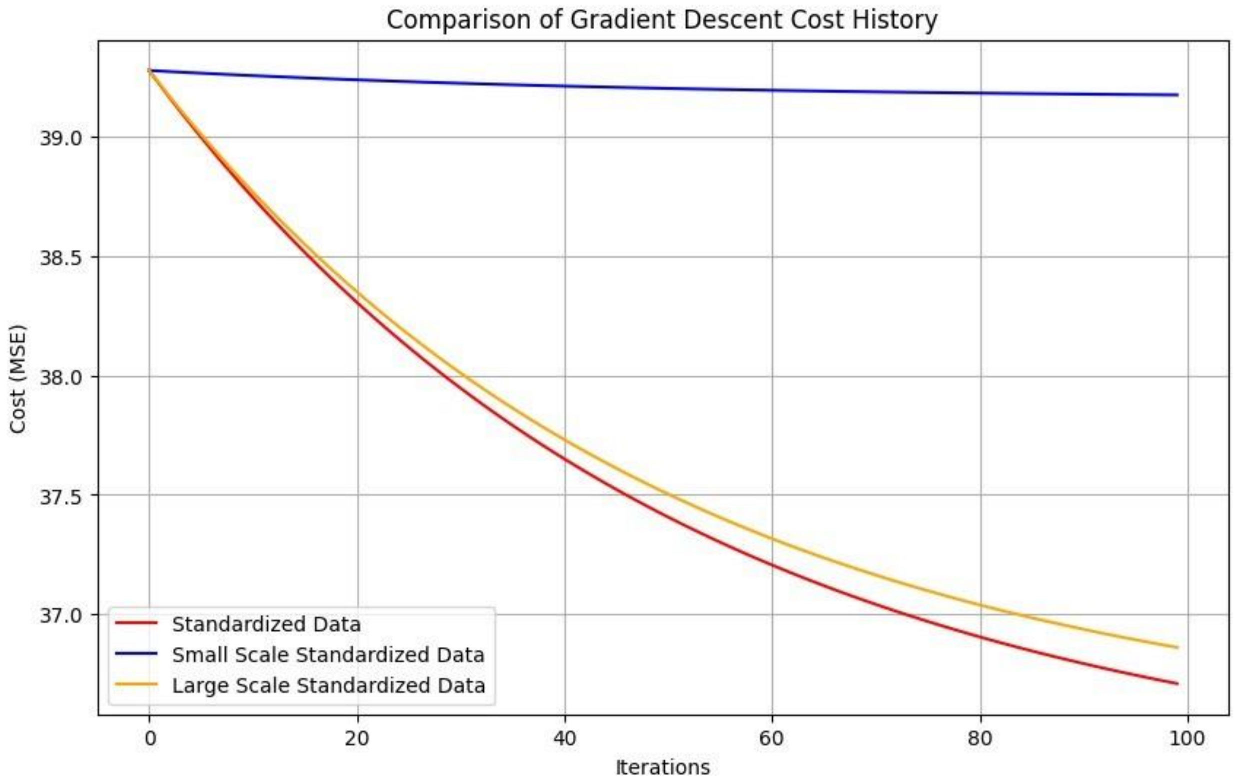
```
plt.grid(True)

# Plot for standardized small dataset
plt.subplot(1, 3, 2)
plt.plot(cost_history_small_standard, color="blue")
plt.title("Gradient Descent on Small Scale Standardized Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

# Plot for standardized large dataset
plt.subplot(1, 3, 3)
plt.plot(cost_history_large_standard, color="orange")
plt.title("Gradient Descent on Large Scale Standardized Data")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.grid(True)

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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Set random seed for reproducibility
np.random.seed(0)

# Generate synthetic dataset with features on different scales
X_small = np.random.rand(100, 1)                  # Small scale
feature (0-1)
X_large = np.random.rand(100, 1) * 1000           # Large scale
feature (0-1000)
X = np.hstack((X_small, X_large))                 # Combine features
```


Gradient Descent on Standardized Data
Gradient Descent on Small Scale Standardized Data
Gradient Descent on Large Scale Standardized Data
Cost (MSE)
Iterations
39.0
38.5
38.0
37.5
37.0
39.28
39.26
39.24
39.22
39.20
39.18
0
20
40
60
80
100

```
# True coefficients for linear regression (for generating target
values)
true_coef = np.array([5, 0.01])
y = X @ true_coef + np.random.randn(100) * 5      # Generate target
variable with noise

# Apply Standardization to the entire dataset
scaler_standard = StandardScaler()
X_standard = scaler_standard.fit_transform(X)

# Apply Standardization separately to small and large features
scaler_small = StandardScaler()
X_small_standard = scaler_small.fit_transform(X_small)

scaler_large = StandardScaler()
X_large_standard = scaler_large.fit_transform(X_large)

# Gradient Descent function def gradient_descent(X, y,
learning_rate, iterations):     n_samples = X.shape[0]
theta = np.zeros(X.shape[1])     cost_history = []
for i in range(iterations):
        y_pred = X @ theta
error = y_pred - y
        theta -= (learning_rate / n_samples) * (X.T @ error)
cost = (1 / (2 * n_samples)) * np.sum(error ** 2)
cost_history.append(cost)     return theta, cost_history

# Run gradient descent on standardized data
theta_standard, cost_history_standard = gradient_descent(X_standard,
y, learning_rate=0.01, iterations=100)
theta_small_standard, cost_history_small_standard =
gradient_descent(X_small_standard, y, learning_rate=0.01,
iterations=100)
theta_large_standard, cost_history_large_standard =
gradient_descent(X_large_standard, y, learning_rate=0.01,
iterations=100)

# Plot cost history for comparison
plt.figure(figsize=(10, 6))
plt.plot(cost_history_standard, label="Standardized Data",
color="red")
plt.plot(cost_history_small_standard, label="Small Scale Standardized
Data", color="blue")
plt.plot(cost_history_large_standard, label="Large Scale Standardized
Data", color="orange")
```

```
plt.title("Comparison of Gradient Descent Cost History")
plt.xlabel("Iterations") plt.ylabel("Cost (MSE)")
plt.legend() plt.grid(True) plt.show()
```




```
print("""
Explanation of Results:
Unscaled Data:

Gradient descent struggles to converge due to the large differences in
feature ranges (0-1 vs. 0-1000), and thus requires a tiny learning
rate. Convergence is very slow or may not happen in a reasonable
timeframe.
Min-Max Scaling:

Scales all features to the
[
0
,
1
]
[0,1] range, allowing gradient descent to converge steadily at a
reasonable learning rate.
This scaling method works well but can be sensitive to extreme values.
Standardization:
```

```
Standardization (mean=0, std=1) scales features more effectively when
large variances are present, stabilizing updates even with outliers.
It tends to be more robust than Min-Max scaling and is generally the
preferred choice for gradient descent.""")
```

```
Explanation of Results:
Unscaled Data:

Gradient descent struggles to converge due to the large differences in
feature ranges (0-1 vs. 0-1000), and thus requires a tiny learning
rate. Convergence is very slow or may not happen in a reasonable
timeframe.
Min-Max Scaling:

Scales all features to the
[
0
,
1
]
[0,1] range, allowing gradient descent to converge steadily at a
reasonable learning rate. This scaling method works well but can be
sensitive to extreme values.
Standardization:

Standardization (mean=0, std=1) scales features more effectively when
large variances are present, stabilizing updates even with outliers.
It tends to be more robust than Min-Max scaling and is generally the
preferred choice for gradient descent.
```