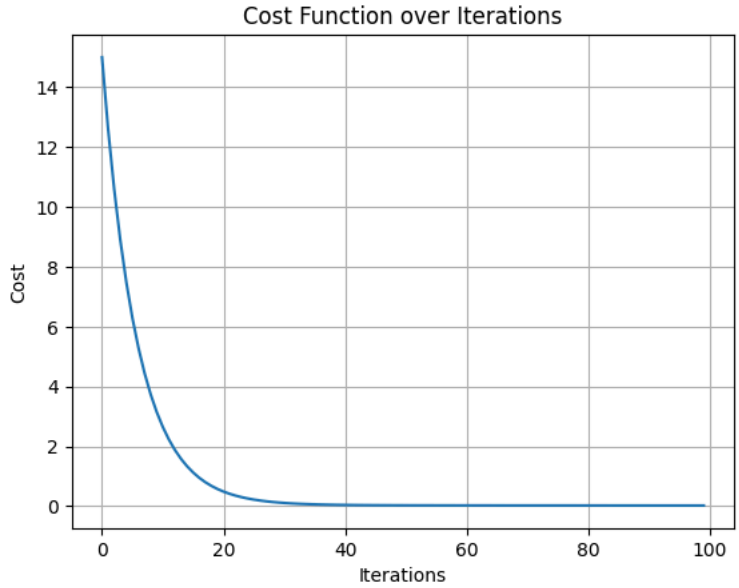
```
# Q1. Implement Vectorized Gradient Descent

import numpy as np

# Data
X = np.array([1, 2, 3, 4])
y = np.array([2, 4, 6, 8])
m = len(y)  # number of examples

# Initialize parameters
theta0 = 0.0
theta1 = 0.0
alpha = 0.01
iterations = 100

# Store cost for plotting
cost_history = []

for i in range(iterations):
    h = theta0 + theta1 * X  # Vectorized prediction
    loss = h - y             # Vectorized error
    cost = (1 / (2 * m)) * np.sum(loss ** 2)
    cost_history.append(cost)

    # Gradient calculations (vectorized)
    grad_theta0 = (1 / m) * np.sum(loss)
    grad_theta1 = (1 / m) * np.sum(loss * X)

    # Update thetas
    theta0 -= alpha * grad_theta0
    theta1 -= alpha * grad_theta1

print(f"Final theta0: {theta0}")
print(f"Final theta1: {theta1}")



# Q2. Compare Iteration Speed

import time

# Vectorized version
start_vec = time.time()
for i in range(1):
    h = theta0 + theta1 * X
    loss = h - y
    grad_theta0 = (1 / m) * np.sum(loss)
    grad_theta1 = (1 / m) * np.sum(loss * X)
    theta0 -= alpha * grad_theta0
    theta1 -= alpha * grad_theta1
end_vec = time.time()

print("Vectorized Time:", end_vec - start_vec)

# Non-vectorized version
theta0_nv, theta1_nv = 0.0, 0.0
start_nv = time.time()
for i in range(1):
    total_error_0 = 0
    total_error_1 = 0
    for j in range(m):
        prediction = theta0_nv + theta1_nv * X[j]
        error = prediction - y[j]
        total_error_0 += error
        total_error_1 += error * X[j]
    theta0_nv -= alpha * (1 / m) * total_error_0
    theta1_nv -= alpha * (1 / m) * total_error_1
end_nv = time.time()

print("Non-Vectorized Time:", end_nv - start_nv)
# Observation: Vectorized is much faster because NumPy uses optimized C-based operations under the hood.


# Q3. Plot Cost Function
import matplotlib.pyplot as plt
```

```
plt.plot(range(iterations), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Function over Iterations")
plt.grid(True)
plt.show()
# The plot should show a smooth decrease in cost over time, meaning the model is learning.



# Q4. Prediction Accuracy python

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split

# Convert to 2D shape for sklearn
X_reshaped = X.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.25, random_state=42)

# Use the learned thetas for prediction
y_pred = theta0 + theta1 * X_test.flatten()

# Evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"MSE: {mse}")
print(f"R²: {r2}")
print(f"MAE: {mae}")



# Q5. Visualize Results
# Plotting best-fit line with training and test data
plt.scatter(X_train, y_train, color='blue', label='Training Data')
plt.scatter(X_test, y_test, color='red', label='Test Data')
x_line = np.linspace(0, 5, 100)
y_line = theta0 + theta1 * x_line
plt.plot(x_line, y_line, color='green', label='Best Fit Line')
plt.xlabel('Distance (km)')
plt.ylabel('Time (hrs)')
plt.title('Univariate Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

```
Final theta0: 0.5247844642513884
Final theta1: 1.8211825989712458
Vectorized Time: 0.00019121170043945312
Non-Vectorized Time: 0.00021338462829589844
```




```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined wit
  warnings.warn(msg, UndefinedMetricWarning)
MSE: 0.02787418556714109
R²: nan
MAE: 0.16695563951882875
```