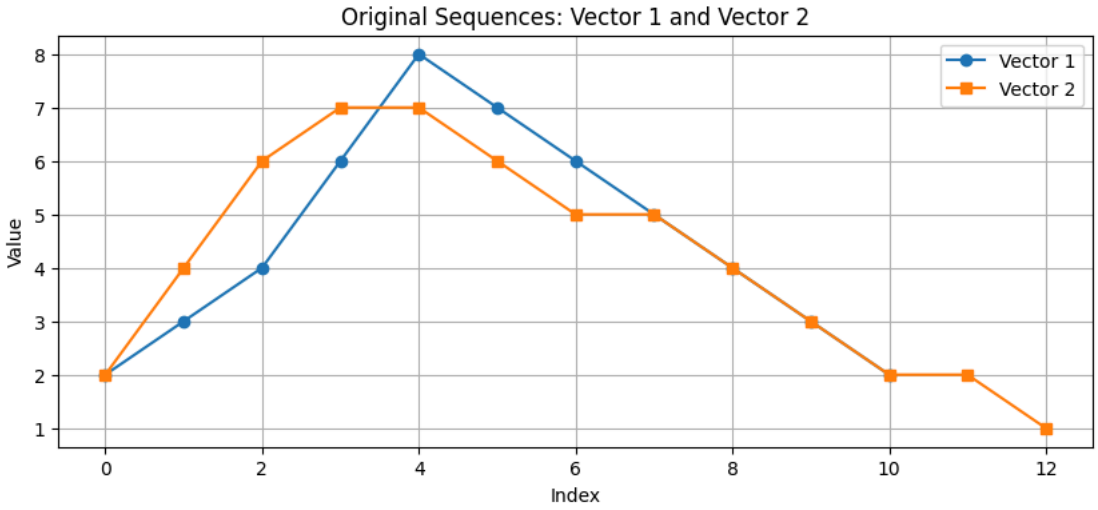
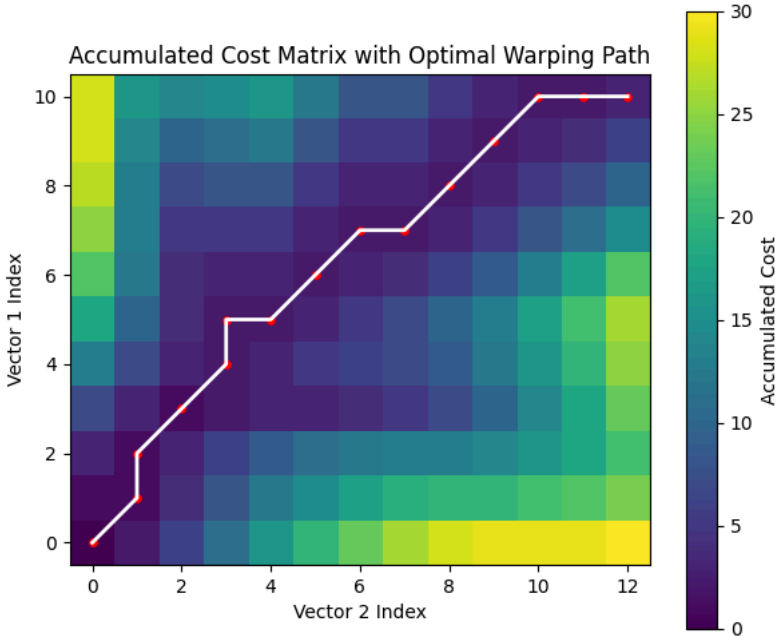
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Step 1: Given Data
vector1 = np.array([2, 3, 4, 6, 8, 7, 6, 5, 4, 3, 2])
vector2 = np.array([2, 4, 6, 7, 7, 6, 5, 5, 4, 3, 2, 2, 1])

len1, len2 = len(vector1), len(vector2)

# Step 2: Plot both vectors to visualize their patterns
plt.figure(figsize=(10, 4))
plt.plot(vector1, marker='o', label='Vector 1')
plt.plot(vector2, marker='s', label='Vector 2')
plt.title('Original Sequences: Vector 1 and Vector 2')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()

# Step 3: Implement the Dynamic Time Warping (DTW) algorithm
def dtw(x, y, dist_func=lambda a, b: abs(a - b)):
    n, m = len(x), len(y)
    cost = np.zeros((n, m))

    # Initialize first cell
    cost[0, 0] = dist_func(x[0], y[0])

    # Initialize first column
    for i in range(1, n):
        cost[i, 0] = cost[i - 1, 0] + dist_func(x[i], y[0])

    # Initialize first row
    for j in range(1, m):
        cost[0, j] = cost[0, j - 1] + dist_func(x[0], y[j])

    # Populate rest of the matrix
    for i in range(1, n):
        for j in range(1, m):
            choices = cost[i - 1, j], cost[i, j - 1], cost[i - 1, j - 1]
            cost[i, j] = dist_func(x[i], y[j]) + min(choices)

    # Traceback for optimal path
    i, j = n - 1, m - 1
    path = [(i, j)]

    while i > 0 or j > 0:
        if i == 0:
            j -= 1
        elif j == 0:
            i -= 1
        else:
            min_idx = np.argmin([cost[i - 1, j], cost[i, j - 1], cost[i - 1, j - 1]])
            if min_idx == 0:
                i -= 1
            elif min_idx == 1:
                j -= 1
            else:
                i -= 1
                j -= 1
        path.append((i, j))

    path.reverse()
    return cost, path, cost[-1, -1]

# Compute DTW
cost_matrix, path, dtw_distance = dtw(vector1, vector2)

# Step 4: Visualize the Accumulated Cost Matrix and Optimal Path
plt.figure(figsize=(7, 6))
plt.imshow(cost_matrix, origin='lower', cmap='viridis', interpolation='nearest')
plt.title('Accumulated Cost Matrix with Optimal Warping Path')
plt.xlabel('Vector 2 Index')
plt.ylabel('Vector 1 Index')
plt.colorbar(label='Accumulated Cost')

# Plot optimal path
path_x, path_y = zip(*path)
plt.plot(path_y, path_x, 'w-', linewidth=2)
plt.scatter(path_y, path_x, color='red', s=15)
```

```
plt.show()

# Step 5: Print the DTW Distance and Mapping Table
print(f"DTW Distance between Vector 1 and Vector 2: {dtw_distance:.4f}")

# Create a mapping DataFrame for clarity
mapping_df = pd.DataFrame(path, columns=["Vector1_Index", "Vector2_Index"])
mapping_df["Vector1_Value"] = [vector1[i] for i, _ in path]
mapping_df["Vector2_Value"] = [vector2[j] for _, j in path]
mapping_df["Abs_Difference"] = abs(mapping_df["Vector1_Value"] - mapping_df["Vector2_Value"])

# Display mapping
print("\nOptimal Warping Path Mapping:\n")
print(mapping_df.to_string(index=False))
```






```
DTW Distance between Vector 1 and Vector 2: 3.0000

Optimal Warping Path Mapping:

 Vector1_Index  Vector2_Index  Vector1_Value  Vector2_Value  Abs_Difference
             0              0              2              2               0
             1              1              3              4               1
             2              1              4              4               0
             3              2              6              6               0
             4              3              8              7               1
             5              3              7              7               0
             5              4              7              7               0
             6              5              6              6               0
             7              6              5              5               0
             7              7              5              5               0
             8              8              4              4               0
             9              9              3              3               0
            10             10              2              2               0
            10             11              2              2               0
            10             12              2              1               1
```