Solution 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import sqrtm
```

```
# Define the adjacency matrix for the graph
adjacency_matrix = np.matrix(np.array([
     [0, 1, 0, 0, 1, 1],
     [1, 0, 1, 1, 0, 0],
     [0, 1, 0, 1, 0, 0],
     [0, 1, 1, 0, 0, 0],
     [1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0]
]))
```

```
# Calculate the degree matrix
degree_matrix = np.matrix(np.diag(np.sum(np.array(adjacency_matrix), axis=0)))
degree_matrix
```

```
matrix([[3, 0, 0, 0, 0, 0],
        [0, 3, 0, 0, 0, 0],
        [0, 0, 2, 0, 0, 0],
        [0, 0, 0, 2, 0, 0],
        [0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 1]])
```

1.1: *Consider a binary classification problem. Write down the initial label vector P0 for this graph where v6 has observed label 1 and v4 and v5 have observed label 2.*

```
# Set the initial labels for the graph
initial_labels = np.matrix(np.array([0, 0, 0, -1, -1, 1])).T
initial_labels
```

```
matrix([[ 0],
        [ 0],
        [ 0],
        [-1],
        [-1],
        [ 1]])
```

```
# Normalize the similarity matrix
normalized_similarity_matrix = sqrtm(np.linalg.matrix_power(degree_matrix, -1)) @ adjacency_matrix @ sqrtm(np.linalg.matrix_pc
```

```
# Implement the label spreading algorithm
def label_spread(similarity_matrix, labels, decay=0.8, max_iter=20):
    m = 0
    for i in range(max_iter):
        aS = decay * similarity_matrix
        m += np.linalg.matrix_power(aS, i) @ labels
    aS = decay * similarity_matrix
    final_labels = (1 - decay) * m + np.linalg.matrix_power(aS, max_iter) @ labels
    return final_labels
```

```
# Function to get labels for nodes
def get_labels_vector(labels):
    labels_vector = ['' for _ in range(labels.shape[0])]
    for i in range(labels.shape[0]):
        if labels[i] < 0:
            labels_vector[i] = "label 2"
        elif labels[i] > 0:
            labels_vector[i] = "label 1"
        else:
            labels_vector[i] = "No label"
    return labels_vector

    # Function to print labels
def print_labels(labels_vector):
    print("v1:", str(labels_vector[0]), "\nv2:", labels_vector[1], "\nv3:", labels_vector[2])
```

1.2: *Perform 1 iteration of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P1 and provide the labels l1, l2, and l3 after 1 iteration.*

```
#Perform 1 iteration of the label spreading algorithm
labels_after_1_iteration = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 1)
labels_vector_1 = get_labels_vector(labels_after_1_iteration)
print_labels(labels_vector_1)
```

```
    v1: No label
    v2: label 2
    v3: label 2
```

1.3: *Perform 2 iterations of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P2 and provide the labels l1, l2, and l3 after 2 iterations.*

```
#Perform 2 iterations of the label spreading algorithm
labels_after_2_iterations = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 2)
labels_vector_2 = get_labels_vector(labels_after_2_iterations)
print_labels(labels_vector_2)
```

```
    v1: label 2
    v2: label 2
    v3: label 2
```

1.4: *Perform infinite iterations of the label spreading algorithm with the decay parameter α = 0.8 and determine the node labels for the unlabeled nodes v1, v2, and v3, i.e., compute P∞ and provide the labels l1, l2, and l3 after infinite iterations.*

```
#Perform infinite iterations of the label spreading algorithm
labels_after_infinite_iterations = label_spread(normalized_similarity_matrix, initial_labels, 0.8, 10000)
labels_vector_infinite = get_labels_vector(labels_after_infinite_iterations)
print_labels(labels_vector_infinite)
```

```
    v1: label 2
    v2: label 2
    v3: label 2
```

We will verify convergence through the utilization of the following formula:

F$_*$= $(I-\alpha S)$-1 * Y

```
# Confirm convergence
convergence_check = (1 - 0.8) * np.linalg.matrix_power((np.identity(normalized_similarity_matrix.shape[0]) - 0.8 * normalized_
convergence_check
```

```
    matrix([[-0.0972985 ],
            [-0.20919178],
            [-0.20910767],
            [-0.35196482],
            [-0.24494025],
            [ 0.15505975]])
```

we can confirm the convergence from the above result.

1.5: *Determine the node labels for the unlabeled nodes v1, v2, and v3 via the energy minimization algorithm.*

lets first calculate the laplacian matrix for the Similarity matrix:

```
laplacian_matrix = degree_matrix - adjacency_matrix
laplacian_matrix
```

```
    matrix([[ 3, -1,  0,  0, -1, -1],
            [-1,  3, -1, -1,  0,  0],
            [ 0, -1,  2, -1,  0,  0],
            [ 0, -1, -1,  2,  0,  0],
            [-1,  0,  0,  0,  1,  0],
            [-1,  0,  0,  0,  0,  1]])
```

```
# One-hot vector encoding of the labeled nodes
one_hot_labels = np.matrix([
     [0, 1],
     [0, 1],
     [1, 0]
])
```

We will use below formula to calculate the Fu

$$F_u = -L_{uu}^{-1} L_{uu} Y_l$$

```
# Calculate Fu using the energy minimization algorithm
Fu = -1 * np.linalg.inv(laplacian_matrix[0:3, 0:3]) @ laplacian_matrix[3:6, 0:3] @ one_hot_labels
unlabeled_node_labels = np.argmax(Fu, axis=1)

# Print the results
unlabeled_node_labels
```

```
    matrix([[1],
            [1],
            [1]])
```

The analysis reveals that all data points are categorized under the second label. Furthermore, it is observed that the algorithm successfully identifies unlabeled points without altering the labeled nodes, in contrast to the label spreading algorithm, which alters the labels of the nodes.