Bài tập thực hành-Khai thác dữ liệu-tuần 7

Phan Hồng Trâm - 21110414

May 2024

Mục lục

1 Trình bày tóm tắt phần code do em viết và so sánh với hàm có sẵn trong thư viện

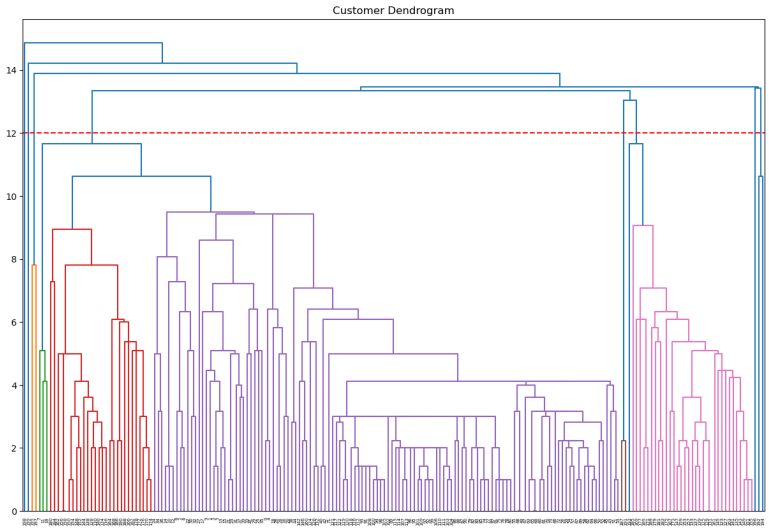
1.1 Code với hàm có sẵn trong thư viện:

Ta import thư viện cần thiết:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.cluster.hierarchy as shc
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
```

- **Single-linkage:**
  - Vẽ dendrogram và đường nằm ngang đi qua khoảng cách dài nhất:

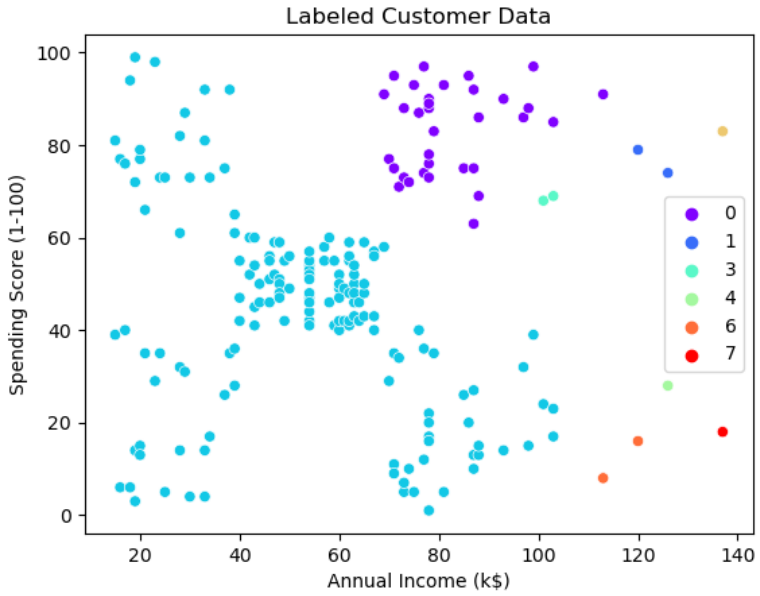
```
plt.figure(figsize=(15, 10))
plt.title('Customer Dendrogram')
selected_data = customer_data_oh.iloc[:, 1:3]
clusters = shc.linkage(selected_data, method='single', metric='euclidean')
shc.dendrogram(Z=clusters)
plt.axhline(y = 12, color = 'r', linestyle = '-')
plt.show()
```



Hình 1: Dendrogram với `method = 'single'`

  - **Vẽ biểu đồ phân tán clusters:**

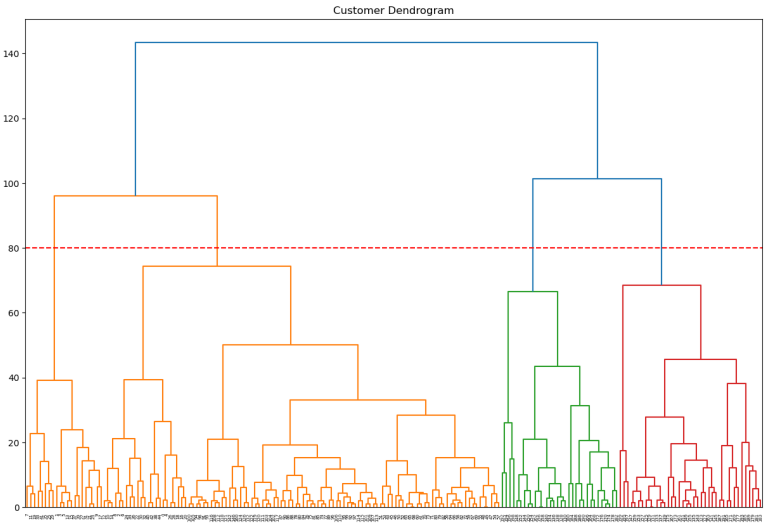
```
clustering_model = AgglomerativeClustering(n_clusters=8, affinity='euclidean', linkage='single')
clustering_model.fit(selected_data)
data_labels = clustering_model.labels_
sns.scatterplot(x = 'Annual Income (k$)', y = 'Spending Score (1-100)', data=selected_data, hue=
    data_labels, palette='rainbow')
plt.title('Labeled Customer Data')
plt.show()
```



Hình 2: Scatter plot với `method = 'single', n_clusters = 8`

- **Complete-linkage:**
  - Vẽ dendrogram và đường nằm ngang đi qua khoảng cách dài nhất:

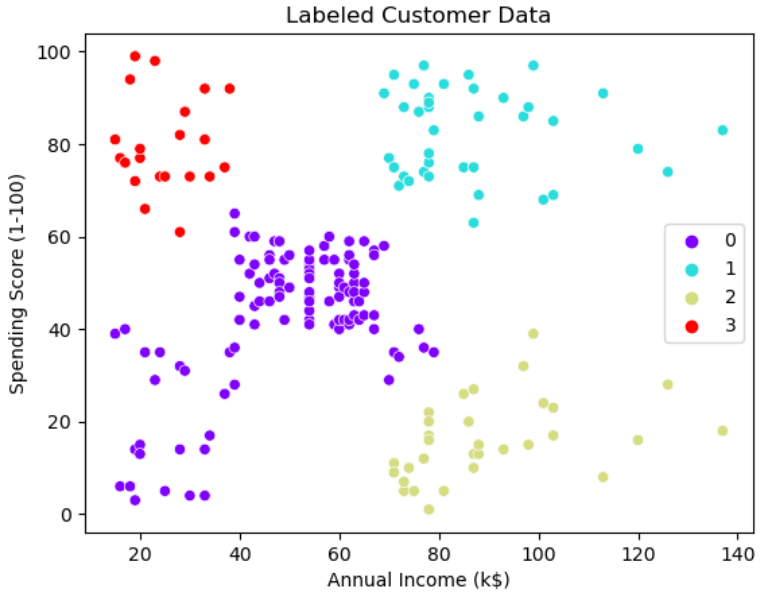
```
plt.figure(figsize=(15, 10))
plt.title('Customer Dendrogram')
selected_data = customer_data_oh.iloc[:, 1:3]
clusters = shc.linkage(selected_data, method='complete', metric='euclidean')
shc.dendrogram(Z=clusters)
plt.axhline(y = 80, color = 'r', linestyle = '-')
plt.show()
```



Hình 3: Dendrogram với `method = 'complete'`

– Vẽ biểu đồ phân tán clusters:

```
clustering_model = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='complete'
    )
clustering_model.fit(selected_data)
data_labels = clustering_model.labels_
sns.scatterplot(x = 'Annual Income (k$)', y = 'Spending Score (1-100)', data=selected_data, hue=
    data_labels, palette='rainbow')
plt.title('Labeled Customer Data')
plt.show()
```

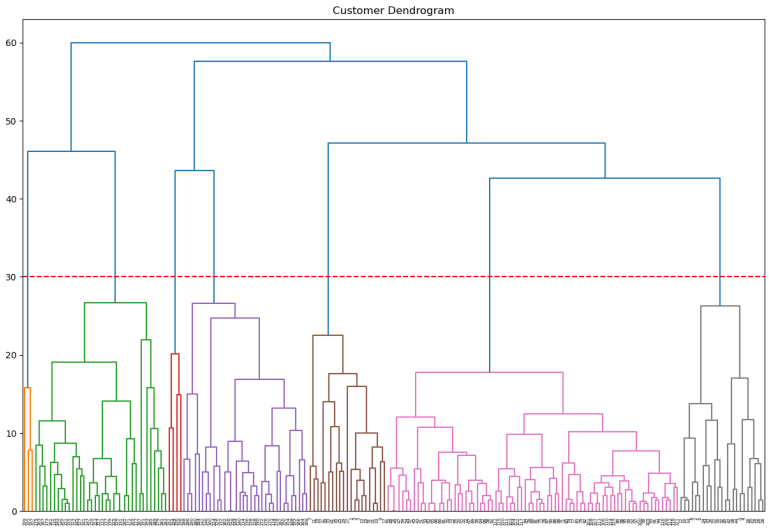


Hình 4: Scatter plot với `method = 'complete', n_clusters = 4`

- **average-linkage:**

– Vẽ dendrogram và đường nằm ngang đi qua khoảng cách dài nhất:

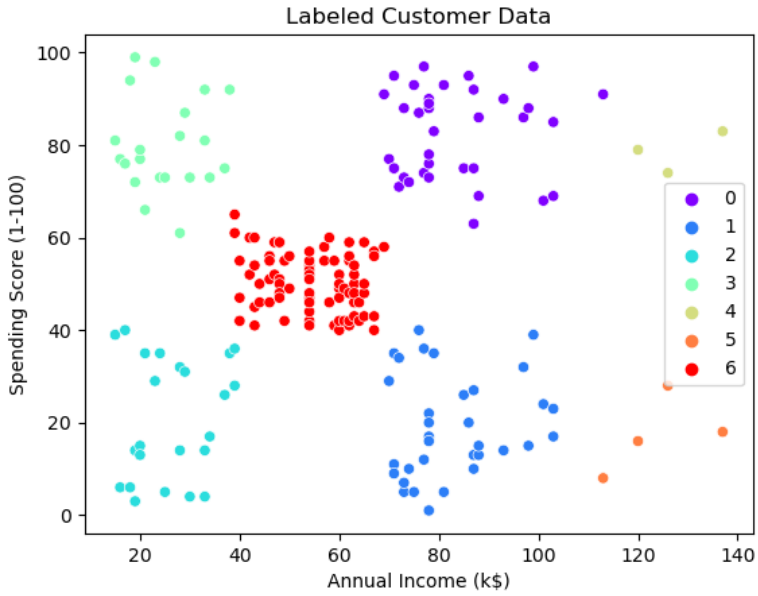
```
plt.figure(figsize=(15, 10))
plt.title('Customer Dendrogram')
selected_data = customer_data_oh.iloc[:, 1:3]
clusters = shc.linkage(selected_data, method='average', metric='euclidean')
shc.dendrogram(Z=clusters)
plt.axhline(y = 30, color = 'r', linestyle = '-')
plt.show()
```



Hình 5: Dendrogram với `method = 'average'`

– Vẽ biểu đồ phân tán clusters:

```
clustering_model = AgglomerativeClustering(n_clusters=7, affinity='euclidean', linkage='average')
clustering_model.fit(selected_data)
data_labels = clustering_model.labels_
sns.scatterplot(x = 'Annual Income (k$)', y = 'Spending Score (1-100)', data=selected_data, hue=
    data_labels, palette='rainbow')
plt.title('Labeled Customer Data')
plt.show()
```



Hình 6: Scatter plot với `method = 'average', n_clusters=7`

1.2 Code không sử dụng thư viện:

Import các thư viện cần thiết:

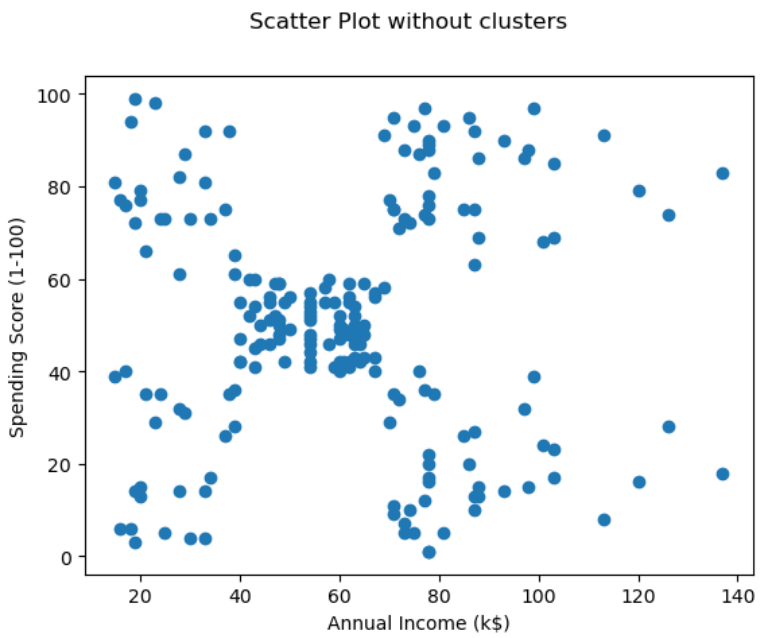
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
import sys
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Chuyển data về dạng mảng:

```
data = np.array(selected_data.values)
print(data)
```

Vẽ biểu đồ phân tán khi chưa phân cụm:

```
fig = plt.figure()
fig.suptitle('Scatter Plot without clusters')
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Annual Income (k$)')
ax.set_ylabel('Spending Score (1-100)')
ax.scatter(data[:,0],data[:,1])
```



Hình 7: Biểu đồ phân tán của dữ liệu khi chưa phân cụm

- Hàm `find_clusters` tìm kiếm các cluster từ ma trận khoảng cách đầu vào theo phương pháp single, complete và average:

```
def find_clusters(input, linkage):
    clusters = {}
    row_index = -1
    col_index = -1
    array = []
```

```
    for n in range(input.shape[0]):
        array.append(n)

    clusters[0] = array.copy()

    #finding minimum value from the distance matrix
    #note that this loop will always return minimum value from bottom triangle of matrix
    for k in range(1, input.shape[0]):
        min_val = sys.maxsize

        for i in range(0, input.shape[0]):
            for j in range(0, input.shape[1]):
                if(input[i][j]<=min_val):
                    min_val = input[i][j]
                    row_index = i
                    col_index = j

        #once we find the minimum value, we need to update the distance matrix
        #updating the matrix by calculating the new distances from the cluster to all points

        #for Single Linkage
        if(linkage == "single" or linkage =="Single"):
            for i in range(0,input.shape[0]):
                if(i != col_index):
                    #we calculate the distance of every data point from newly formed cluster and
    update the matrix.
                    temp = min(input[col_index][i],input[row_index][i])
                    #we update the matrix symmetrically as our distance matrix should always be
    symmetric
                    input[col_index][i] = temp
                    input[i][col_index] = temp
        #for Complete Linkage
        elif(linkage=="Complete" or linkage == "complete"):
             for i in range(0,input.shape[0]):
                if(i != col_index and i!=row_index):
                    temp = max(input[col_index][i],input[row_index][i])
                    input[col_index][i] = temp
                    input[i][col_index] = temp
        #for Average Linkage
        elif(linkage=="Average" or linkage == "average"):
             for i in range(0,input.shape[0]):
                if(i != col_index and i != row_index):
                    temp = (input[col_index][i]+input[row_index][i])/2
                    input[col_index][i] = temp
                    input[i][col_index] = temp


        #set the rows and columns for the cluster with higher index i.e. the row index to infinity
        #Set input[row_index][for_all_i] = infinity
        #set input[for_all_i][row_index] = infinity
        for i in range (0,input.shape[0]):
            input[row_index][i] = sys.maxsize
            input[i][row_index] = sys.maxsize

        #Manipulating the dictionary to keep track of cluster formation in each step
        #if k=0,then all datapoints are clusters

        minimum = min(row_index,col_index)
        maximum = max(row_index,col_index)
        for n in range(len(array)):
            if(array[n]==maximum):
```

```
                array[n] = minimum
        clusters[k] = array.copy()

    return clusters
```

– **Giải thích chi tiết hàm `find_clusters`:**

* Khởi tạo dictionary `clusters` để lưu trữ các cluster tại mỗi bước.
* `row_index=-1` và `col_index=-1` để lưu trữ chỉ số của các điểm tạo nên khoảng cách nhỏ nhất.
* `array` lưu trữ các chỉ số của các điểm dữ liệu ban đầu.
* `clusters[0]` khởi tạo với tất cả các điểm dữ liệu là các cluster riêng biệt.
* `for k in range(1, input.shape[0])`: lặp qua mỗi bước để tìm cluster mới, trong đó biến `min_val` khởi tạo với giá trị lớn nhất có thể.
* Hai vòng lặp lồng nhau để tìm khoảng cách nhỏ nhất trong ma trận khoảng cách `input`.

```
        for i in range(0, input.shape[0]):
            for j in range(0, input.shape[1]):
                if(input[i][j] <= min_val):
                    min_val = input[i][j]
                    row_index = i
                    col_index = j
```

* `if(linkage == "single" or linkage == "Single")`: cập nhật khoảng cách giữa cluster mới và các điểm khác bằng khoảng cách nhỏ nhất (min).

$$D(C_i, C_j) = \min\{d(v_p, v_q) \mid v_p \in C_i, v_q \in C_j\}$$

* `elif(linkage == "Complete" or linkage == "complete")`: cập nhật khoảng cách bằng khoảng cách lớn nhất (max).

$$D(C_i, C_j) = \max\{d(v_p, v_q) \mid v_p \in C_i, v_q \in C_j\}$$

* `elif(linkage=="Average" or linkage == "average")`: cập nhật khoảng cách bằng khoảng cách trung bình (mean).

$$D(C_i, C_j) = mean\{d(v_p, v_q) \mid v_p \in C_i, v_q \in C_j\}$$

* Đặt giá trị khoảng cách của các phần tử đã được gộp thành cluster mới thành giá trị lớn nhất để loại bỏ chúng khỏi các lần tính toán tiếp theo:

```
        for i in range(0, input.shape[0]):
            input[row_index][i] = sys.maxsize
            input[i][row_index] = sys.maxsize
```

* Cập nhật `array` để giữ chỉ số của các điểm dữ liệu trong cluster mới và lưu trạng thái hiện tại của `array` vào `clusters`:

```
        minimum = min(row_index, col_index)
        maximum = max(row_index, col_index)
        for n in range(len(array)):
            if(array[n] == maximum):
                array[n] = minimum
        clusters[k] = array.copy()
```

* Trả về dictionary `clusters` chứa các trạng thái của các cluster tại mỗi bước.

- Hàm `hierarchical_clustering`:

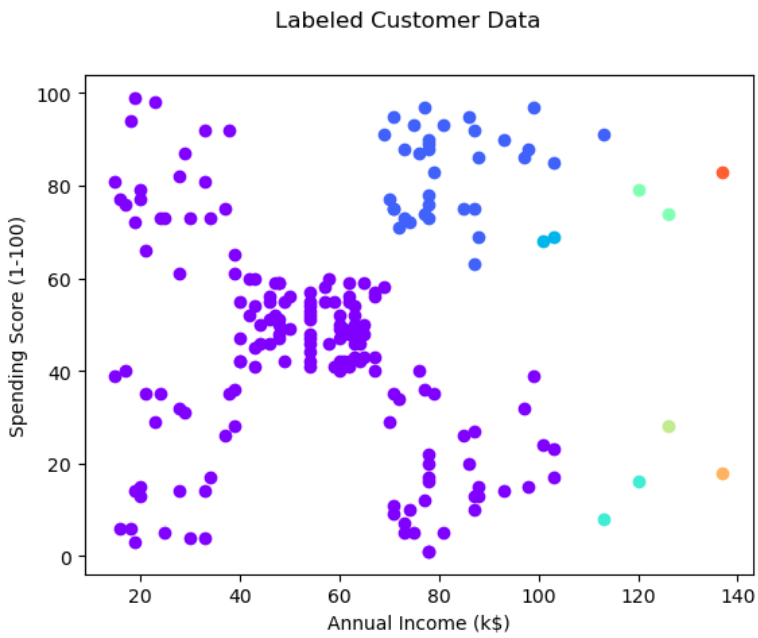
```
def hierarchical_clustering(data, linkage, no_of_clusters):
    #first step is to calculate the initial distance matrix
    #it consists distances from all the point to all the point
    rainbow_palette = plt.get_cmap('rainbow')
    initial_distances = pairwise_distances(data, metric='euclidean')
    #making all the diagonal elements infinity
    np.fill_diagonal(initial_distances,sys.maxsize)
    clusters = find_clusters(initial_distances,linkage)

    #plotting the clusters
    iteration_number = initial_distances.shape[0] - no_of_clusters
    clusters_to_plot = clusters[iteration_number]
    arr = np.unique(clusters_to_plot)

    indices_to_plot = []
    fig = plt.figure()
    fig.suptitle('Labeled Customer Data')
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel('Annual Income (k$)')
    ax.set_ylabel('Spending Score (1-100)')
    for x in np.nditer(arr):
        indices_to_plot.append(np.where(clusters_to_plot==x))
    p = 0

    for i in range(0, len(indices_to_plot)):
        for j in np.nditer(indices_to_plot[i]):
            scatter = ax.scatter(x = data[j,0],y = data[j,1], c=rainbow_palette(p/len(arr)))
        p = p + 1

    plt.show()
```

  - **Giải thích chi tiết hàm `hierarchical_clustering`:**
    - `rainbow_palette = plt.get_cmap('rainbow')`: lấy một colormap 'rainbow' từ thư viện Matplotlib.
    - `initial_distances = pairwise_distances(data, metric='euclidean')`: Tính toán ma trận khoảng cách ban đầu giữa các điểm dữ liệu bằng khoảng cách Euclidean.
    - `np.fill_diagonal(initial_distances,sys.maxsize)`: Đặt các phần tử đường chéo của ma trận (khoảng cách của mỗi điểm đến chính nó) thành giá trị lớn nhất.
    - `clusters = find_clusters(initial_distances, linkage)`: Gọi hàm `find_clusters` để tìm các cluster.
    - `iteration_number = initial_distances.shape[0] - no_of_clusters`: xác định số bước cần thực hiện để đạt được số lượng cluster mong muốn.
    - `clusters_to_plot = clusters[iteration_number]`: lấy trạng thái của các cluster tại bước `iteration_number`.
    - `arr = np.unique(clusters_to_plot)`: chứa các giá trị duy nhất (các chỉ số của các cluster).
    - Sau đó ta tạo một biểu đồ scatter plot để hiển thị các cluster.

- Chạy thử chương trình với các phương pháp khác nhau:

  - **Single-linkage:**

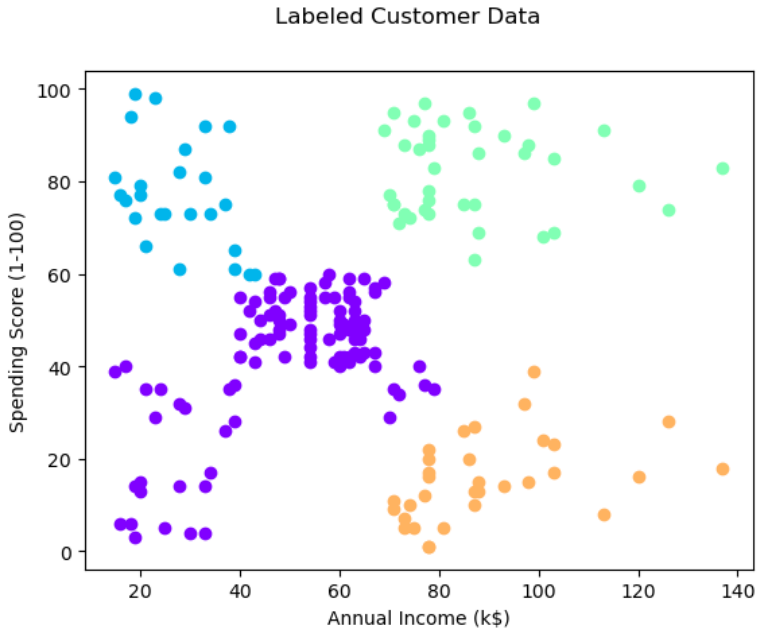
```
single_linkage = hierarchical_clustering(data = data, linkage = "single", no_of_clusters = 8)
```



Hình 8: Scatterplot với `method = 'single', n_clusters = 8` (không dùng thư viện)

– **Complete-linkage:**

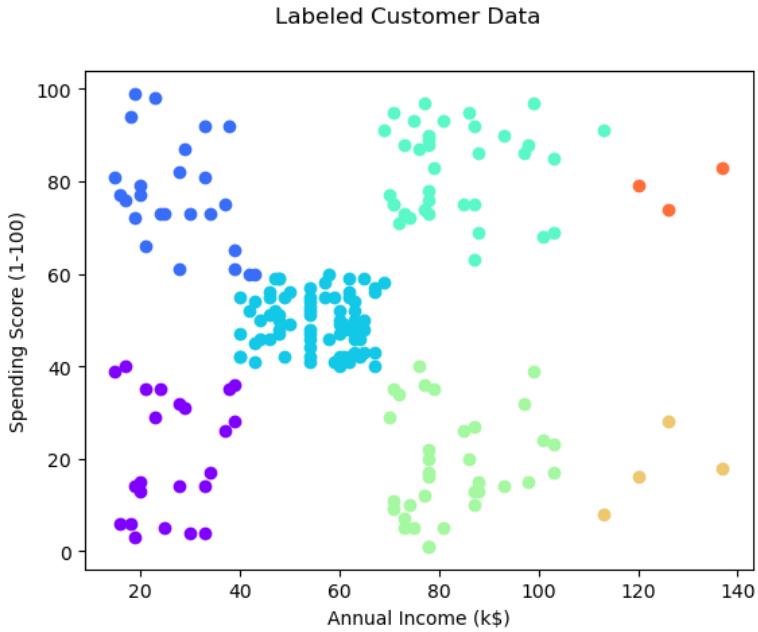
```
complete_linkage = hierarchical_clustering(data = data, linkage = "complete", no_of_clusters = 4)
```



Hình 9: Scatterplot với `method = 'complete', n_clusters = 4` (không dùng thư viện)

– **Average-linkage:**

```
average_linkage = hierarchical_clustering(data = data, linkage = "average", no_of_clusters = 7)
```

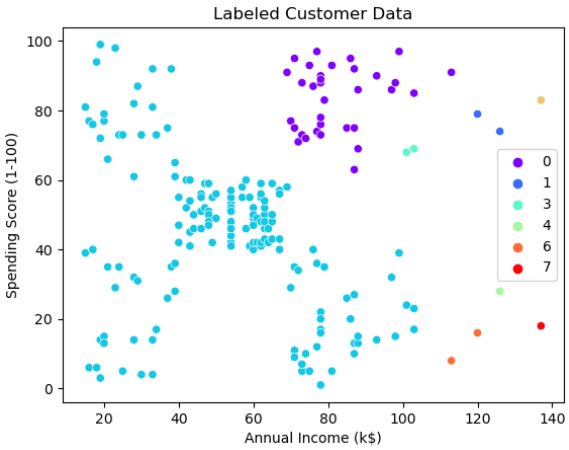


Hình 10: Scatterplot với `method = 'average', n_clusters = 7` (không dùng thư viện)

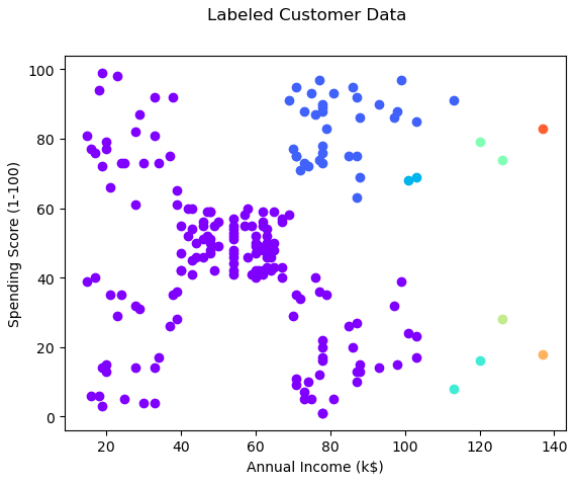
1.3 So sánh

Ta so sánh kết quả biểu đồ scatter khi dùng thư viện và không dùng thư viện:

- **Single-linkage:**



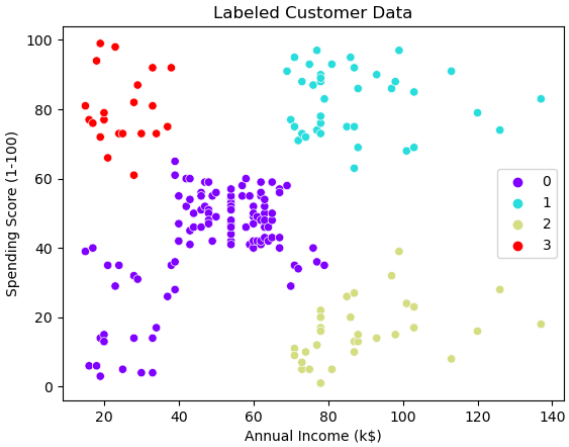
((a)) Scatter plot với single-linkage dùng thư viện



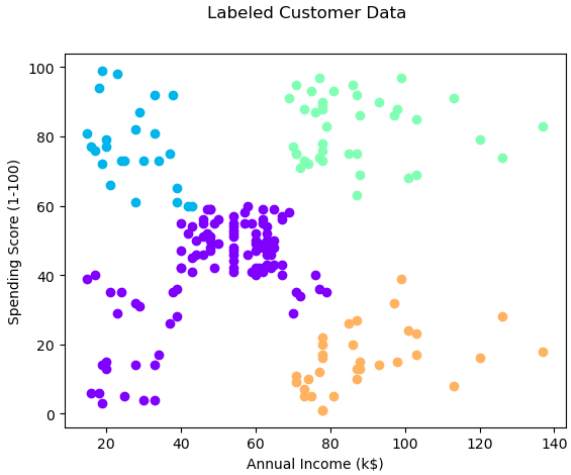
((b)) Scatter plot với single-linkage không dùng thư viện

Hình 11: Kết quả so sánh phương pháp Single-linkage

- **Complete-linkage:**



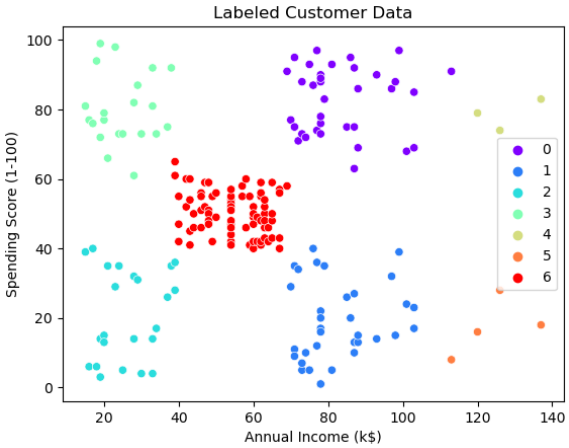
((a)) Scatter plot với complete-linkage dùng thư viện



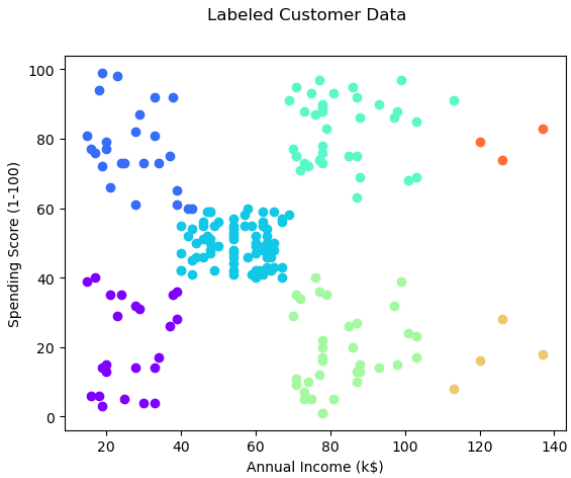
((b)) Scatter plot với complete-linkage không dùng thư viện

Hình 12: Kết quả so sánh phương pháp Complete-linkage

- **Average-linkage:**



((a)) Scatter plot với average-linkage dùng thư viện



((b)) Scatter plot với average-linkage không dùng thư viện

Hình 13: Kết quả so sánh phương pháp Average-linkage

- Nhận xét: Sau khi chạy code, ta có thể thấy kết quả của thuật toán gom cụm phân cấp `hierarchical_clustering` khi dùng thư viện và không dùng thư viện khá là tương đồng với nhau.