# 6.2

## 最短距离法

import numpy as np

from scipy.cluster.hierarchy import linkage, dendrogram

from scipy.spatial.distance import squareform

# 给定样本相关矩阵

correlation\_matrix = np.array([

[1.000, 0.366, 0.242, 0.280, 0.360, 0.282, 0.245, 0.448, 0.486, 0.648, 0.679, 0.486, 0.133, 0.376],

[0.366, 1.000, 0.233, 0.194, 0.324, 0.263, 0.265, 0.345, 0.367, 0.662, 0.681, 0.636, 0.153, 0.252],

[0.242, 0.233, 1.000, 0.590, 0.476, 0.483, 0.540, 0.452, 0.365, 0.216, 0.243, 0.174, 0.732, 0.676],

[0.280, 0.194, 0.590, 1.000, 0.435, 0.470, 0.478, 0.404, 0.357, 0.316, 0.313, 0.243, 0.477, 0.581],

[0.360, 0.324, 0.476, 0.435, 1.000, 0.452, 0.535, 0.431, 0.429, 0.429, 0.430, 0.375, 0.339, 0.441],

[0.282, 0.263, 0.483, 0.470, 0.452, 1.000, 0.633, 0.322, 0.283, 0.283, 0.302, 0.290, 0.392, 0.447],

[0.245, 0.265, 0.540, 0.478, 0.535, 0.633, 1.000, 0.266, 0.287, 0.263, 0.294, 0.255, 0.446, 0.440],

[0.448, 0.345, 0.452, 0.404, 0.431, 0.322, 0.266, 1.000, 0.820, 0.527, 0.520, 0.403, 0.266, 0.424],

[0.486, 0.367, 0.365, 0.357, 0.429, 0.283, 0.287, 0.820, 1.000, 0.547, 0.558, 0.417, 0.241, 0.372],

[0.648, 0.662, 0.216, 0.316, 0.429, 0.283, 0.263, 0.527, 0.547, 1.000, 0.957, 0.852, 0.054, 0.363],

[0.679, 0.681, 0.243, 0.313, 0.430, 0.302, 0.294, 0.520, 0.558, 0.957, 1.000, 0.852, 0.099, 0.376],

[0.486, 0.636, 0.174, 0.243, 0.375, 0.290, 0.255, 0.403, 0.417, 0.857, 0.852, 1.000, 0.055, 0.321],

[0.133, 0.153, 0.732, 0.477, 0.339, 0.392, 0.446, 0.266, 0.241, 0.054, 0.099, 0.055, 1.000, 0.627],

[0.376, 0.252, 0.676, 0.581, 0.441, 0.447, 0.440, 0.424, 0.372, 0.363, 0.376, 0.321, 0.627, 1.000]

])

# 使用 sqrt(2 \* (1 - correlation)) 作为距离矩阵

distance\_matrix = np.sqrt(2 \* (1 - correlation\_matrix))

# 强制对称化以确保数值稳定性

distance\_matrix = (distance\_matrix + distance\_matrix.T) / 2

# 将距离矩阵转换为1D形式以适用于linkage函数

condensed\_distance\_matrix = squareform(distance\_matrix)

# 使用单链接法进行层次聚类

Z = linkage(condensed\_distance\_matrix, method='single')

# 打印每一步的距离矩阵

n = len(correlation\_matrix)

clusters = {i: [i] for i in range(n)}

print("Initial distance matrix:")

print(distance\_matrix)

print("\n")

for step in range(Z.shape[0]):

cluster1 = int(Z[step, 0])

cluster2 = int(Z[step, 1])

dist = Z[step, 2]

new\_cluster = max(clusters) + 1

# 更新簇

clusters[new\_cluster] = clusters.pop(cluster1) + clusters.pop(cluster2)

# 计算新的距离矩阵

new\_distance\_matrix = np.full((len(clusters), len(clusters)), np.inf)

cluster\_keys = list(clusters.keys())

for i, key\_i in enumerate(cluster\_keys):

for j, key\_j in enumerate(cluster\_keys):

if i != j:

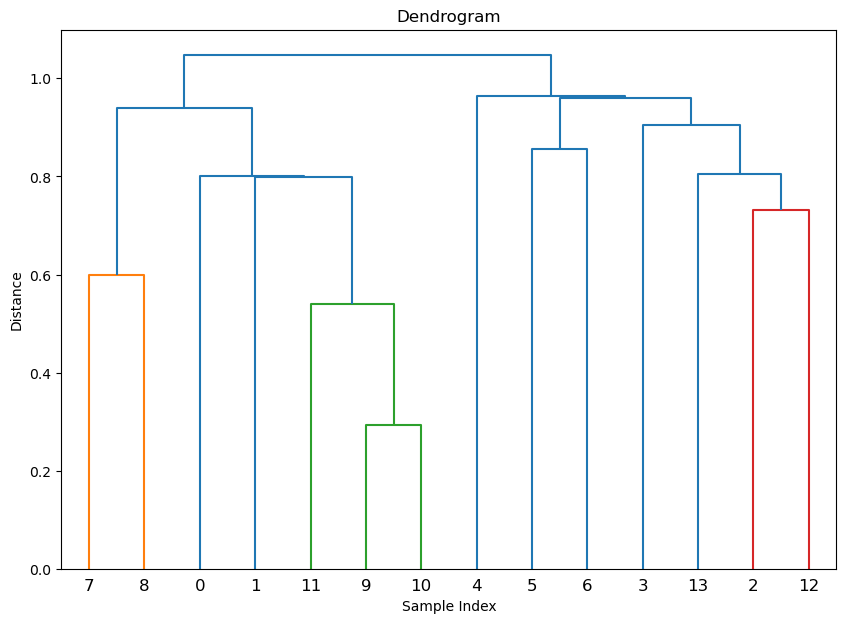
# 取最小距离

new\_distance\_matrix[i, j] = min(distance\_matrix[x, y] for x in clusters[key\_i] for y in clusters[key\_j])

print(f"Step {step + 1}:")

print(new\_distance\_matrix)

print("\n")



## 最长距离法

import numpy as np

from scipy.cluster.hierarchy import linkage, dendrogram

from scipy.spatial.distance import squareform

# 给定样本相关矩阵

correlation\_matrix = np.array([

[1.000, 0.366, 0.242, 0.280, 0.360, 0.282, 0.245, 0.448, 0.486, 0.648, 0.679, 0.486, 0.133, 0.376],

[0.366, 1.000, 0.233, 0.194, 0.324, 0.263, 0.265, 0.345, 0.367, 0.662, 0.681, 0.636, 0.153, 0.252],

[0.242, 0.233, 1.000, 0.590, 0.476, 0.483, 0.540, 0.452, 0.365, 0.216, 0.243, 0.174, 0.732, 0.676],

[0.280, 0.194, 0.590, 1.000, 0.435, 0.470, 0.478, 0.404, 0.357, 0.316, 0.313, 0.243, 0.477, 0.581],

[0.360, 0.324, 0.476, 0.435, 1.000, 0.452, 0.535, 0.431, 0.429, 0.429, 0.430, 0.375, 0.339, 0.441],

[0.282, 0.263, 0.483, 0.470, 0.452, 1.000, 0.633, 0.322, 0.283, 0.283, 0.302, 0.290, 0.392, 0.447],

[0.245, 0.265, 0.540, 0.478, 0.535, 0.633, 1.000, 0.266, 0.287, 0.263, 0.294, 0.255, 0.446, 0.440],

[0.448, 0.345, 0.452, 0.404, 0.431, 0.322, 0.266, 1.000, 0.820, 0.527, 0.520, 0.403, 0.266, 0.424],

[0.486, 0.367, 0.365, 0.357, 0.429, 0.283, 0.287, 0.820, 1.000, 0.547, 0.558, 0.417, 0.241, 0.372],

[0.648, 0.662, 0.216, 0.316, 0.429, 0.283, 0.263, 0.527, 0.547, 1.000, 0.957, 0.852, 0.054, 0.363],

[0.679, 0.681, 0.243, 0.313, 0.430, 0.302, 0.294, 0.520, 0.558, 0.957, 1.000, 0.852, 0.099, 0.376],

[0.486, 0.636, 0.174, 0.243, 0.375, 0.290, 0.255, 0.403, 0.417, 0.857, 0.852, 1.000, 0.055, 0.321],

[0.133, 0.153, 0.732, 0.477, 0.339, 0.392, 0.446, 0.266, 0.241, 0.054, 0.099, 0.055, 1.000, 0.627],

[0.376, 0.252, 0.676, 0.581, 0.441, 0.447, 0.440, 0.424, 0.372, 0.363, 0.376, 0.321, 0.627, 1.000]

])

# 使用 sqrt(2 \* (1 - correlation)) 作为距离矩阵

distance\_matrix = np.sqrt(2 \* (1 - correlation\_matrix))

# 强制对称化以确保数值稳定性

distance\_matrix = (distance\_matrix + distance\_matrix.T) / 2

# 将距离矩阵转换为1D形式以适用于linkage函数

condensed\_distance\_matrix = squareform(distance\_matrix)

# 使用单链接法进行层次聚类

Z = linkage(condensed\_distance\_matrix, method='complete')

# 打印每一步的距离矩阵

n = len(correlation\_matrix)

clusters = {i: [i] for i in range(n)}

print("Initial distance matrix:")

print(distance\_matrix)

print("\n")

for step in range(Z.shape[0]):

cluster1 = int(Z[step, 0])

cluster2 = int(Z[step, 1])

dist = Z[step, 2]

new\_cluster = max(clusters) + 1

# 更新簇

clusters[new\_cluster] = clusters.pop(cluster1) + clusters.pop(cluster2)

# 计算新的距离矩阵

new\_distance\_matrix = np.full((len(clusters), len(clusters)), np.inf)

cluster\_keys = list(clusters.keys())

for i, key\_i in enumerate(cluster\_keys):

for j, key\_j in enumerate(cluster\_keys):

if i != j:

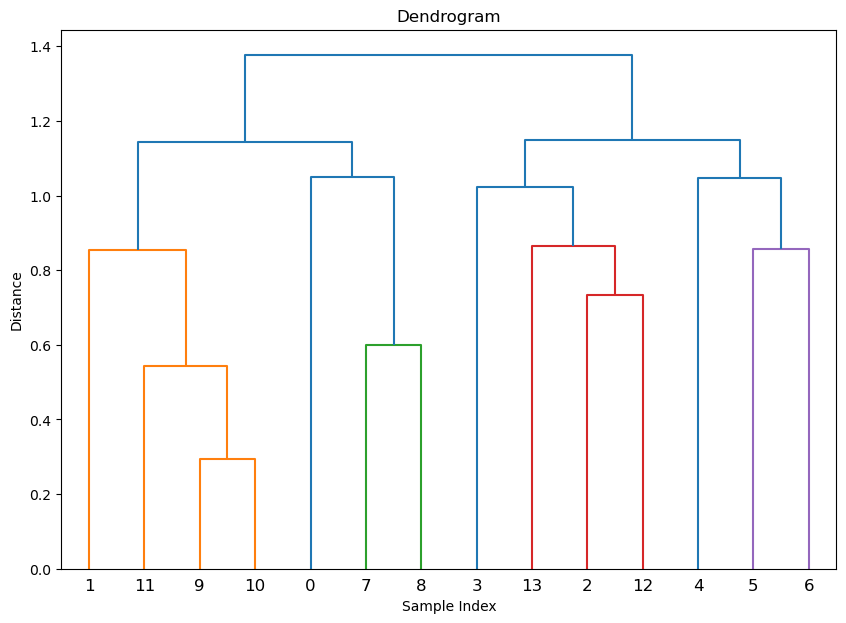
# 取最大距离

new\_distance\_matrix[i, j] = max(distance\_matrix[x, y] for x in clusters[key\_i] for y in clusters[key\_j])

print(f"Step {step + 1}:")

print(new\_distance\_matrix)

print("\n")



## 类平均法

import numpy as np

from scipy.cluster.hierarchy import linkage, dendrogram

from scipy.spatial.distance import squareform

# 给定样本相关矩阵

correlation\_matrix = np.array([

[1.000, 0.366, 0.242, 0.280, 0.360, 0.282, 0.245, 0.448, 0.486, 0.648, 0.679, 0.486, 0.133, 0.376],

[0.366, 1.000, 0.233, 0.194, 0.324, 0.263, 0.265, 0.345, 0.367, 0.662, 0.681, 0.636, 0.153, 0.252],

[0.242, 0.233, 1.000, 0.590, 0.476, 0.483, 0.540, 0.452, 0.365, 0.216, 0.243, 0.174, 0.732, 0.676],

[0.280, 0.194, 0.590, 1.000, 0.435, 0.470, 0.478, 0.404, 0.357, 0.316, 0.313, 0.243, 0.477, 0.581],

[0.360, 0.324, 0.476, 0.435, 1.000, 0.452, 0.535, 0.431, 0.429, 0.429, 0.430, 0.375, 0.339, 0.441],

[0.282, 0.263, 0.483, 0.470, 0.452, 1.000, 0.633, 0.322, 0.283, 0.283, 0.302, 0.290, 0.392, 0.447],

[0.245, 0.265, 0.540, 0.478, 0.535, 0.633, 1.000, 0.266, 0.287, 0.263, 0.294, 0.255, 0.446, 0.440],

[0.448, 0.345, 0.452, 0.404, 0.431, 0.322, 0.266, 1.000, 0.820, 0.527, 0.520, 0.403, 0.266, 0.424],

[0.486, 0.367, 0.365, 0.357, 0.429, 0.283, 0.287, 0.820, 1.000, 0.547, 0.558, 0.417, 0.241, 0.372],

[0.648, 0.662, 0.216, 0.316, 0.429, 0.283, 0.263, 0.527, 0.547, 1.000, 0.957, 0.852, 0.054, 0.363],

[0.679, 0.681, 0.243, 0.313, 0.430, 0.302, 0.294, 0.520, 0.558, 0.957, 1.000, 0.852, 0.099, 0.376],

[0.486, 0.636, 0.174, 0.243, 0.375, 0.290, 0.255, 0.403, 0.417, 0.857, 0.852, 1.000, 0.055, 0.321],

[0.133, 0.153, 0.732, 0.477, 0.339, 0.392, 0.446, 0.266, 0.241, 0.054, 0.099, 0.055, 1.000, 0.627],

[0.376, 0.252, 0.676, 0.581, 0.441, 0.447, 0.440, 0.424, 0.372, 0.363, 0.376, 0.321, 0.627, 1.000]

])

# 使用 sqrt(2 \* (1 - correlation)) 作为距离矩阵

distance\_matrix = np.sqrt(2 \* (1 - correlation\_matrix))

# 强制对称化以确保数值稳定性

distance\_matrix = (distance\_matrix + distance\_matrix.T) / 2

# 将距离矩阵转换为1D形式以适用于linkage函数

condensed\_distance\_matrix = squareform(distance\_matrix)

# 使用单链接法进行层次聚类

Z = linkage(condensed\_distance\_matrix, method='average')

# 打印每一步的距离矩阵

n = len(correlation\_matrix)

clusters = {i: [i] for i in range(n)}

print("Initial distance matrix:")

print(distance\_matrix)

print("\n")

for step in range(Z.shape[0]):

cluster1 = int(Z[step, 0])

cluster2 = int(Z[step, 1])

dist = Z[step, 2]

new\_cluster = max(clusters) + 1

# 更新簇

clusters[new\_cluster] = clusters.pop(cluster1) + clusters.pop(cluster2)

# 计算新的距离矩阵

new\_distance\_matrix = np.full((len(clusters), len(clusters)), np.inf)

cluster\_keys = list(clusters.keys())

for i, key\_i in enumerate(cluster\_keys):

for j, key\_j in enumerate(cluster\_keys):

if i != j:

# 使用类平均法计算两个簇的平均距离

total\_distance = sum(distance\_matrix[x, y] for x in clusters[key\_i] for y in clusters[key\_j])

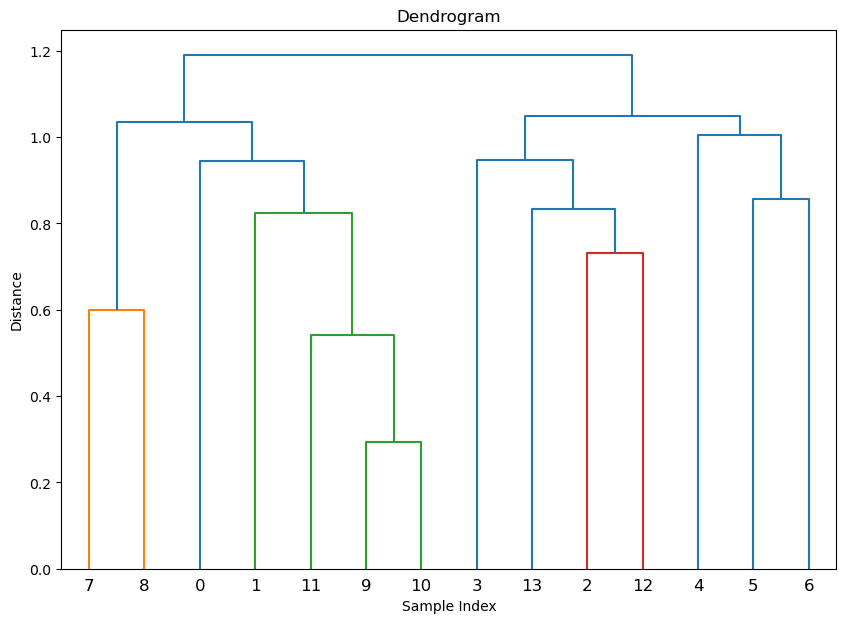
n\_i, n\_j = len(clusters[key\_i]), len(clusters[key\_j])

new\_distance\_matrix[i, j] = total\_distance / (n\_i \* n\_j)

print(f"Step {step + 1}:")

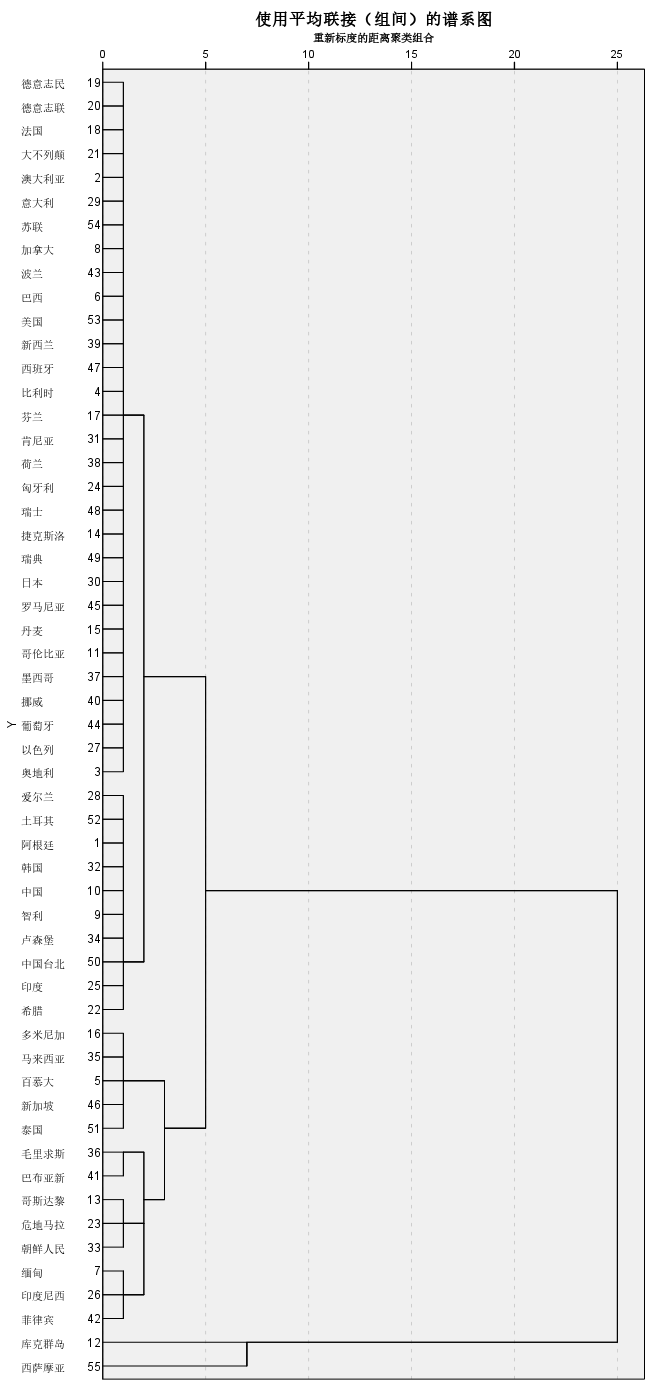
print(new\_distance\_matrix)

print("\n")

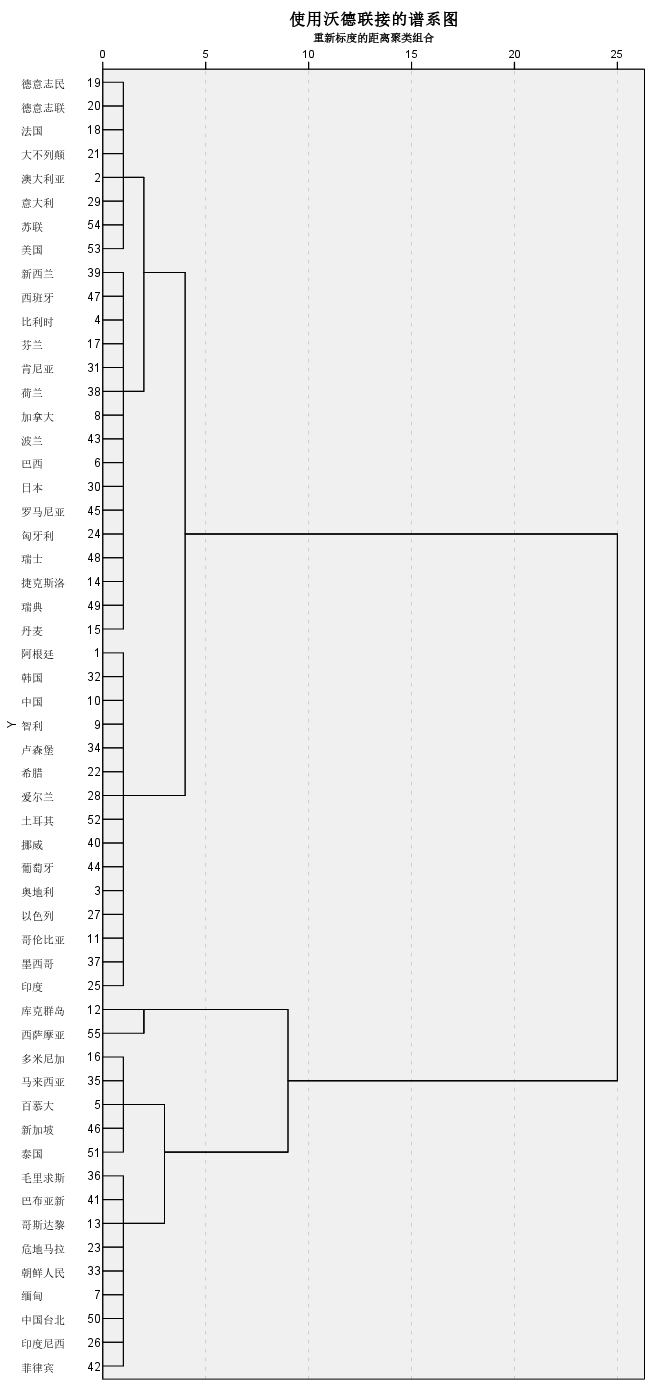


# 6.5

## 类平均法



## 离差平方和法



## k-means法

|  |  |  |
| --- | --- | --- |
| **初始聚类中心** | | |
|  | 聚类 | |
| 1 | 2 |
| 100米（秒） | 12.18 | 9.93 |
| 200米（秒） | 23.20 | 19.75 |
| 400米（秒） | 52.94 | 43.86 |
| 800米（分） | 2.02 | 1.73 |
| 1500米（分） | 4.24 | 3.53 |
| 5000米（分） | 16.70 | 13.20 |
| 10000米（分） | 35.38 | 27.43 |
| 马拉松（分） | 164.70 | 128.22 |

|  |  |  |
| --- | --- | --- |
| **迭代历史记录a** | | |
| 迭代 | 聚类中心中的变动 | |
| 1 | 2 |
| 1 | 11.838 | 5.581 |
| 2 | 1.661 | .595 |
| 3 | .000 | .000 |
| a. 由于聚类中心中不存在变动或者仅有小幅变动，因此实现了收敛。任何中心的最大绝对坐标变动为 .000。当前迭代为 3。初始中心之间的最小距离为 38.810。 | | |

|  |  |  |
| --- | --- | --- |
| **最终聚类中心** | | |
|  | 聚类 | |
| 1 | 2 |
| 100米（秒） | 10.74 | 10.40 |
| 200米（秒） | 21.54 | 20.79 |
| 400米（秒） | 47.81 | 46.10 |
| 800米（分） | 1.87 | 1.77 |
| 1500米（分） | 3.92 | 3.64 |
| 5000米（分） | 15.16 | 13.52 |
| 10000米（分） | 31.96 | 28.25 |
| 马拉松（分） | 153.05 | 132.52 |

|  |  |  |
| --- | --- | --- |
| **每个聚类中的个案数目** | | |
| 聚类 | 1 | 11.000 |
| 2 | 44.000 |
| 有效 | | 55.000 |
| 缺失 | | .000 |

# 7.1



# 7.5

import numpy as np

import pandas as pd

# 创建相关矩阵

correlation\_matrix = np.array([

[1, 0.846, 0.805, 0.859, 0.473, 0.398, 0.301, 0.382],

[0.846, 1, 0.881, 0.826, 0.376, 0.326, 0.277, 0.415],

[0.805, 0.881, 1, 0.801, 0.38, 0.319, 0.237, 0.345],

[0.859, 0.826, 0.801, 1, 0.436, 0.329, 0.327, 0.365],

[0.473, 0.376, 0.38, 0.436, 1, 0.762, 0.73, 0.629],

[0.398, 0.326, 0.319, 0.329, 0.762, 1, 0.583, 0.577],

[0.301, 0.277, 0.237, 0.327, 0.73, 0.583, 1, 0.539],

[0.382, 0.415, 0.345, 0.365, 0.629, 0.577, 0.539, 1]

])

# 计算特征值和特征向量

eigenvalues, eigenvectors = np.linalg.eig(correlation\_matrix)

# 将特征值按照降序排列

sorted\_indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[sorted\_indices]

eigenvectors = eigenvectors[:, sorted\_indices]

# 计算每个主成分的方差贡献率

explained\_variance\_ratio = eigenvalues / np.sum(eigenvalues)

# 打印特征值、特征向量和方差贡献率

print("Eigenvalues:", eigenvalues)

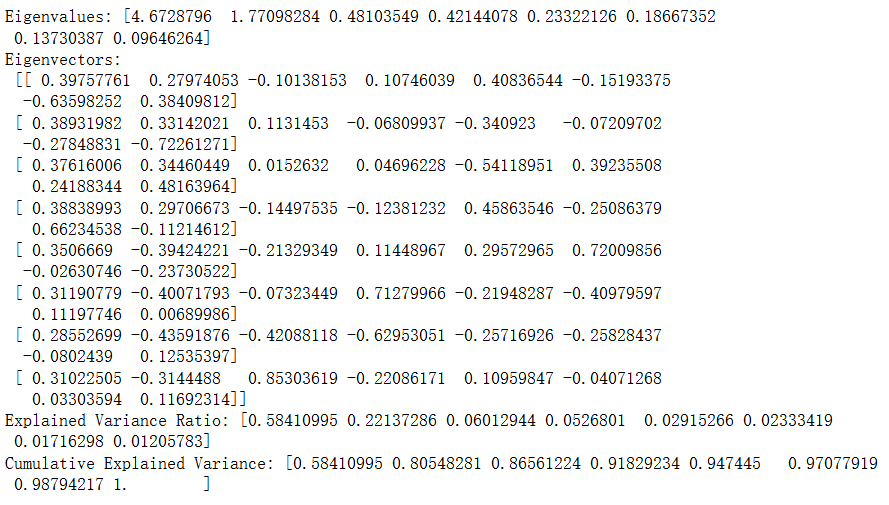
print("Eigenvectors:\n", eigenvectors)

print("Explained Variance Ratio:", explained\_variance\_ratio)

# 计算累计方差贡献率

cumulative\_explained\_variance = np.cumsum(explained\_variance\_ratio)

print("Cumulative Explained Variance:", cumulative\_explained\_variance)



# 7.6

import pandas as pd

# 读取 Excel 文件（请确保路径正确）

file\_path = 'exec7.6.xlsx' # 替换为你的文件路径

df = pd.read\_excel(file\_path)

# 查看数据

print("数据预览：")

print(df.head())

# 计算相关矩阵（忽略非数值列）

correlation\_matrix = df.iloc[:, 1:].corr()

# 输出相关矩阵

print("相关矩阵：")

print(correlation\_matrix)

# 计算特征值和特征向量

eigenvalues, eigenvectors = np.linalg.eig(correlation\_matrix)

# 将特征值按照降序排列

sorted\_indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[sorted\_indices]

eigenvectors = eigenvectors[:, sorted\_indices]

# 计算每个主成分的方差贡献率

explained\_variance\_ratio = eigenvalues / np.sum(eigenvalues)

# 打印特征值、特征向量和方差贡献率

print("Eigenvalues:", eigenvalues)

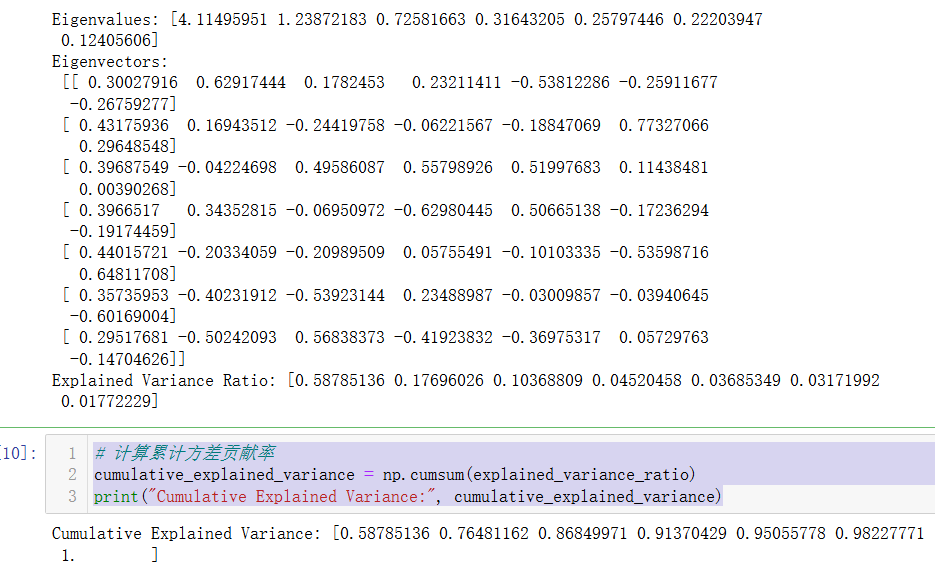
print("Eigenvectors:\n", eigenvectors)

print("Explained Variance Ratio:", explained\_variance\_ratio)

# 计算累计方差贡献率

cumulative\_explained\_variance = np.cumsum(explained\_variance\_ratio)

print("Cumulative Explained Variance:", cumulative\_explained\_variance)



# 8.4

import numpy as np

import pandas as pd

# 1. 准备数据

correlation\_matrix = np.array([

[1, 0.59, 0.35, 0.34, 0.63, 0.4, 0.28, 0.2, 0.11, -0.07],

[0.59, 1, 0.42, 0.51, 0.49, 0.52, 0.31, 0.36, 0.21, 0.09],

[0.35, 0.42, 1, 0.38, 0.19, 0.36, 0.73, 0.24, 0.44, -0.08],

[0.34, 0.51, 0.38, 1, 0.29, 0.46, 0.27, 0.39, 0.17, 0.18],

[0.63, 0.49, 0.19, 0.29, 1, 0.34, 0.17, 0.23, 0.13, 0.39],

[0.4, 0.52, 0.36, 0.46, 0.34, 1, 0.32, 0.33, 0.18, 0],

[0.28, 0.31, 0.73, 0.27, 0.17, 0.32, 1, 0.24, 0.34, -0.02],

[0.2, 0.36, 0.24, 0.39, 0.23, 0.33, 0.24, 1, 0.24, 0.17],

[0.11, 0.21, 0.44, 0.17, 0.13, 0.18, 0.34, 0.24, 1, 0],

[-0.07, 0.09, -0.08, 0.18, 0.39, 0, -0.02, 0.17, 0, 1]

])

# 2. 计算特征值和特征向量

def calculate\_eigenvalues\_and\_vectors(corr\_matrix):

eigenvalues, eigenvectors = np.linalg.eig(corr\_matrix)

return eigenvalues, eigenvectors

eigenvalues, eigenvectors = calculate\_eigenvalues\_and\_vectors(correlation\_matrix)

# 3. 选择前4个特征值和对应的特征向量

n\_factors = 4

indices = np.argsort(eigenvalues)[::-1][:n\_factors]

selected\_eigenvalues = eigenvalues[indices]

selected\_eigenvectors = eigenvectors[:, indices]

# 4. 计算因子载荷

loadings = np.dot(selected\_eigenvectors, np.diag(np.sqrt(selected\_eigenvalues)))

print("因子载荷:\n", loadings)

# 5. 进行最大方差旋转（Varimax）

def varimax(Phi, gamma=1.0, tol=1e-6, max\_iter=100):

"""Varimax rotation"""

p, k = Phi.shape

d = np.diag(Phi.T @ Phi)

last\_Phi = np.zeros\_like(Phi)

for \_ in range(max\_iter):

# 计算 U 矩阵

U = Phi @ np.diag(1.0 / np.sqrt(d + 1e-10)) # 加一个小值避免除零

V = np.linalg.inv(U.T @ U + 1e-10 \* np.eye(k)) # 加小值避免奇异矩阵

W = U.T @ Phi

d = np.diag(W @ W)

# 更新 Phi

Phi = U @ V @ W.T

# 检查收敛

if np.max(np.abs(Phi - last\_Phi)) < tol:

break

last\_Phi = Phi

return Phi

# 进行旋转

rotated\_loadings = varimax(loadings)

print("旋转后的因子载荷:\n", rotated\_loadings)

# 6. 计算共性方差，套用h的公式

communalities = np.sum(rotated\_loadings \*\* 2, axis=1)

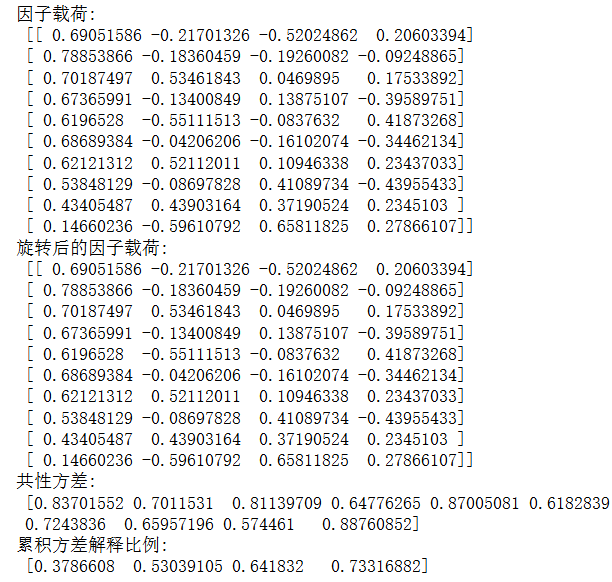
print("共性方差:\n", communalities)

# 7. 计算累积方差解释比例

explained\_variance = selected\_eigenvalues / np.sum(eigenvalues)

cumulative\_explained\_variance = np.cumsum(explained\_variance)

print("累积方差解释比例:\n", cumulative\_explained\_variance)



import numpy as np

from scipy.optimize import minimize

from scipy.linalg import eigh

# 1. 准备数据（相关矩阵）

correlation\_matrix = np.array([

[1, 0.59, 0.35, 0.34, 0.63, 0.4, 0.28, 0.2, 0.11, -0.07],

[0.59, 1, 0.42, 0.51, 0.49, 0.52, 0.31, 0.36, 0.21, 0.09],

[0.35, 0.42, 1, 0.38, 0.19, 0.36, 0.73, 0.24, 0.44, -0.08],

[0.34, 0.51, 0.38, 1, 0.29, 0.46, 0.27, 0.39, 0.17, 0.18],

[0.63, 0.49, 0.19, 0.29, 1, 0.34, 0.17, 0.23, 0.13, 0.39],

[0.4, 0.52, 0.36, 0.46, 0.34, 1, 0.32, 0.33, 0.18, 0],

[0.28, 0.31, 0.73, 0.27, 0.17, 0.32, 1, 0.24, 0.34, -0.02],

[0.2, 0.36, 0.24, 0.39, 0.23, 0.33, 0.24, 1, 0.24, 0.17],

[0.11, 0.21, 0.44, 0.17, 0.13, 0.18, 0.34, 0.24, 1, 0],

[-0.07, 0.09, -0.08, 0.18, 0.39, 0, -0.02, 0.17, 0, 1]

])

# 设置参数

n\_factors = 4 # 因子数量

n\_vars = correlation\_matrix.shape[0]

# 2. 初始化因子载荷和独特方差

loadings = np.random.rand(n\_vars, n\_factors)

unique\_variances = np.diag(correlation\_matrix) \* 0.5 # 初始误差方差

# 3. 迭代更新

tolerance = 1e-6

max\_iter = 1000

for iteration in range(max\_iter):

# 计算协方差矩阵

model\_cov = loadings @ loadings.T + np.diag(unique\_variances)

# 更新独特方差 D

unique\_variances = np.diag(correlation\_matrix - loadings @ loadings.T)

# 计算 A 和 D 的新值

A = loadings

D\_inv = np.diag(1 / unique\_variances)

A = correlation\_matrix @ D\_inv @ A @ np.linalg.inv(np.eye(n\_factors) + A.T @ D\_inv @ A)

# 确保 A'D^-1A 是对角矩阵

A\_diag = A.T @ D\_inv @ A

if np.allclose(A\_diag, np.diag(np.diag(A\_diag))):

break # 满足唯一性条件，停止迭代

# 4. 最大方差旋转

def varimax(Phi, gamma=1.0, tol=1e-5):

p, k = Phi.shape

d = np.ones((k,))

Phi\_ = Phi.copy()

for \_ in range(100):

d\_old = d.copy() # 避免直接修改d

d = np.sqrt(np.sum(Phi\_\*\*2, axis=0))

d[d < tol] = tol # 防止除以零

Phi\_ = Phi @ np.diag(d)

# 旋转

# 设置full\_matrices=False，确保维度与Phi\_兼容

u, \_, v = np.linalg.svd(Phi\_.T, full\_matrices=False)

Phi\_ = (u @ v).T

# 检查收敛

if np.allclose(d, d\_old, rtol=tol):

break

return Phi\_

# 进行最大方差旋转

rotated\_loadings = varimax(A)

# 输出旋转后的因子载荷矩阵

print("旋转后的因子载荷矩阵:\n", rotated\_loadings)

# 6. 计算共性方差，套用h的公式

communalities = np.sum(rotated\_loadings \*\* 2, axis=1)

print("共性方差:\n", communalities)

# 计算总的共性方差

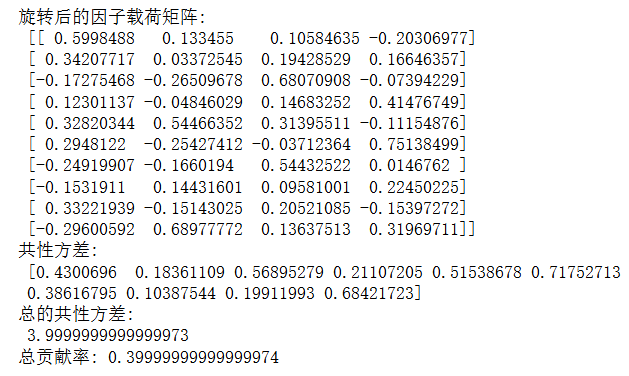
total\_h = np.sum(communalities)

print("总的共性方差:\n", total\_h)

# 计算累计贡献率

cumulative\_explained\_variance = total\_h / n\_vars

print("总贡献率:", cumulative\_explained\_variance)



# 8.5

# 看R满不满秩

import numpy as np

import pandas as pd

# 数据

data = np.array([

[5700, 12.8, 2500, 270, 25000],

[1000, 10.9, 600, 10, 10000],

[3400, 8.8, 1000, 10, 9000],

[3800, 13.6, 1700, 140, 25000],

[4000, 12.8, 1600, 140, 25000],

[8200, 8.3, 2600, 60, 12000],

[1200, 11.4, 400, 10, 16000],

[9100, 11.5, 3300, 60, 14000],

[9900, 12.5, 3400, 180, 18000],

[9600, 13.7, 3600, 390, 25000],

[9600, 9.6, 3300, 80, 12000],

[9400, 11.4, 4000, 100, 13000]

])

# 创建 DataFrame

columns = ['Population', 'Education', 'Servants', 'ServiceIndustry', 'HousePrice']

df = pd.DataFrame(data, columns=columns)

# 计算相关矩阵

correlation\_matrix = df.corr().values

print("相关矩阵:\n", correlation\_matrix)

# 计算秩

rank = np.linalg.matrix\_rank(correlation\_matrix)

print(f"相关矩阵的秩: {rank}")

print(f"相关矩阵的维数: {correlation\_matrix.shape[0]}")

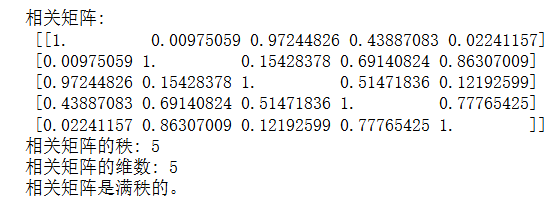
# 检查满秩条件

if rank == correlation\_matrix.shape[0]:

print("相关矩阵是满秩的。")

else:

print("相关矩阵不是满秩的。")





import numpy as np

import pandas as pd

# 准备数据

data = np.array([

[5700, 12.8, 2500, 270, 25000],

[1000, 10.9, 600, 10, 10000],

[3400, 8.8, 1000, 10, 9000],

[3800, 13.6, 1700, 140, 25000],

[4000, 12.8, 1600, 140, 25000],

[8200, 8.3, 2600, 60, 12000],

[1200, 11.4, 400, 10, 16000],

[9100, 11.5, 3300, 60, 14000],

[9900, 12.5, 3400, 180, 18000],

[9600, 13.7, 3600, 390, 25000],

[9600, 9.6, 3300, 80, 12000],

[9400, 11.4, 4000, 100, 13000]

])

# 创建 DataFrame

columns = ['Population', 'Education', 'Servants', 'ServiceIndustry', 'HousePrice']

df = pd.DataFrame(data, columns=columns)

# 计算相关矩阵

correlation\_matrix = df.corr()

print("相关矩阵:\n", correlation\_matrix)

# 满秩

# 计算特殊方差 D 的对角线元素为相关矩阵对角线元素的倒数

special\_variances = 1 / np.diag(correlation\_matrix)

D = np.diag(special\_variances)

print("特殊方差 D:\n", D)

# 计算约相关矩阵

R\_star = correlation\_matrix.values - D

print("约相关矩阵 R\*:\n", R\_star)

# 计算特征值和特征向量

eigenvalues, eigenvectors = np.linalg.eig(R\_star)

# 选择前 n\_factors 个特征值和特征向量

n\_factors = 2

indices = np.argsort(eigenvalues)[::-1][:n\_factors]

selected\_eigenvalues = eigenvalues[indices]

selected\_eigenvectors = eigenvectors[:, indices]

# 计算因子载荷

loadings = np.dot(selected\_eigenvectors, np.diag(np.sqrt(selected\_eigenvalues)))

print("因子载荷:\n", loadings)

# Varimax 旋转

def varimax(Phi, gamma=1.0, tol=1e-6, max\_iter=100):

"""Varimax rotation"""

p, k = Phi.shape

d = np.diag(Phi.T @ Phi)

last\_Phi = np.zeros\_like(Phi)

for \_ in range(max\_iter):

# 计算 U 矩阵

U = Phi @ np.diag(1.0 / np.sqrt(d + 1e-10)) # 加一个小值避免除零

V = np.linalg.inv(U.T @ U + 1e-10 \* np.eye(k)) # 加小值避免奇异矩阵

W = U.T @ Phi

d = np.diag(W @ W)

# 更新 Phi

Phi = U @ V @ W.T

# 检查收敛

if np.max(np.abs(Phi - last\_Phi)) < tol:

break

last\_Phi = Phi

return Phi

# 进行旋转

rotated\_loadings = varimax(loadings)

print("旋转后的因子载荷:\n", rotated\_loadings)

# 计算共性方差

communalities = np.sum(rotated\_loadings \*\* 2, axis=1)

print("共性方差:\n", communalities)

# 计算累积方差解释比例

explained\_variance = selected\_eigenvalues / np.sum(eigenvalues)

cumulative\_explained\_variance = np.cumsum(explained\_variance)

print("累积方差解释比例:\n", cumulative\_explained\_variance)

import numpy as np

import pandas as pd

# 假设有一个数据集和相关矩阵

data = np.array([

[5700, 12.8, 2500, 270, 25000],

[1000, 10.9, 600, 10, 10000],

[3400, 8.8, 1000, 10, 9000],

[3800, 13.6, 1700, 140, 25000],

[4000, 12.8, 1600, 140, 25000],

[8200, 8.3, 2600, 60, 12000],

[1200, 11.4, 400, 10, 16000],

[9100, 11.5, 3300, 60, 14000],

[9900, 12.5, 3400, 180, 18000],

[9600, 13.7, 3600, 390, 25000],

[9600, 9.6, 3300, 80, 12000],

[9400, 11.4, 4000, 100, 13000]

])

# 创建 DataFrame

columns = ['Population', 'Education', 'Servants', 'ServiceIndustry', 'HousePrice']

df = pd.DataFrame(data, columns=columns)

# 计算相关矩阵

correlation\_matrix = df.corr().values

# 初始化特殊方差矩阵

d = np.diag(1 / np.diag(correlation\_matrix)) # 设置初始特殊方差

# 迭代更新特殊方差

tol = 1e-6 # 收敛阈值

max\_iter = 100

for i in range(max\_iter):

# 计算主因子模型 R\* = R - D

R\_star = correlation\_matrix - np.diag(d)

# 计算特征值和特征向量

eigenvalues, eigenvectors = np.linalg.eig(R\_star)

# 选择前 n 个因子

n\_factors = 2

indices = np.argsort(eigenvalues)[::-1][:n\_factors]

selected\_eigenvalues = eigenvalues[indices]

selected\_eigenvectors = eigenvectors[:, indices]

# 计算因子载荷

loadings = np.dot(selected\_eigenvectors, np.diag(np.sqrt(selected\_eigenvalues)))

# 计算共性方差

communalities = np.sum(loadings \*\* 2, axis=1)

# 更新特殊方差 D

new\_d = 1 - communalities # 新的特殊方差

if np.all(np.abs(new\_d - d) < tol):

break # 收敛判断

d = new\_d # 更新特殊方差

# 输出最终结果

print("最终因子载荷矩阵:\n", loadings)

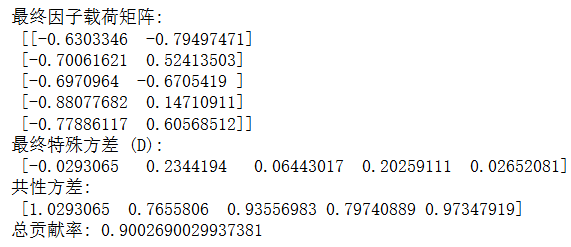
print("最终特殊方差 (D):\n", d)

print("共性方差:\n", communalities)

total\_h = sum(communalities)

贡献率 = total\_h/5

print("总贡献率:",贡献率)



# 8.6

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

# 读取 Excel 文件（替换路径）

file\_path = 'exec8.6.xlsx' # 替换为你的文件路径

df = pd.read\_excel(file\_path)

# 删除"应征者"这一列

df = df.drop(columns=['应征者'])

# 标准化数据

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df)

# 计算协方差矩阵

covariance\_matrix = np.cov(scaled\_data, rowvar=False)

# 对协方差矩阵进行特征值分解

eigenvalues, eigenvectors = np.linalg.eig(covariance\_matrix)

# 按照特征值大小排序，选择前 num\_factors 个

num\_factors = 5

indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[indices][:num\_factors]

eigenvectors = eigenvectors[:, indices][:, :num\_factors]

# 计算因子载荷矩阵 A

A = np.dot(eigenvectors, np.diag(np.sqrt(eigenvalues)))

# 计算特殊方差矩阵 D

D = np.diag(np.diag(covariance\_matrix) - np.sum(A \*\* 2, axis=1))

# 计算共性方差 h\_i^2

common\_variance = np.sum(A \*\* 2, axis=1)

# 计算特殊方差 sigma\_i^2

special\_variance = np.diag(D)

# 计算总方差

total\_variance = common\_variance + special\_variance

# 输出结果

print("特征值：", eigenvalues)

print("因子载荷矩阵（A）：\n", A)

print("特殊方差矩阵（D）：\n", D)

print("共性方差（h\_i^2）：", common\_variance)

print("特殊方差（σ\_i^2）：", special\_variance)

print("总方差：", total\_variance)

total\_h = sum(common\_variance)

贡献率 = total\_h/15

print("总贡献率:",贡献率)

特征值： [7.65891268 2.10132756 1.49356891 1.23305195 0.75485898]

因子载荷矩阵（A）：

[[ 0.44998065 -0.62406505 -0.37569566 -0.12050893 -0.10069419]

[ 0.58940075 0.04892565 0.0173516 0.29197602 -0.75631538]

[ 0.11012267 -0.34347376 0.50516996 0.71776727 0.18375417]

[ 0.6223045 0.18183575 -0.58121759 0.36522932 0.1108907 ]

[ 0.80774157 0.36185705 0.29784815 -0.17980356 -0.0044836 ]

[ 0.87415088 0.18967674 0.18398693 -0.07100576 0.17931331]

[ 0.43755671 0.58254062 -0.36463272 0.45272458 0.05468235]

[ 0.89081405 0.05695761 0.24732366 -0.23250916 -0.03164593]

[ 0.3685675 -0.80347394 -0.10026099 0.07101238 0.08984456]

[ 0.87318353 -0.06744913 0.1010269 -0.16686168 0.17496616]

[ 0.88195434 0.09934581 0.25838525 -0.20854327 -0.14124364]

[ 0.9172954 0.03162129 0.13606748 0.09267159 0.07190163]

[ 0.92188643 -0.03498679 0.07852521 0.2155423 0.11068981]

[ 0.71710359 0.11520099 -0.56554646 -0.23685717 0.09720696]

[ 0.65297572 -0.61109579 -0.10427992 -0.02797062 -0.07052043]]

特殊方差矩阵（D）：

[[0.26352787 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0.01392561 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0.08702324 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0.11744638 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0.11682668 0.

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.15011331

0. 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0.14956043 0. 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0.10825178 0. 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0.21669715 0. 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0.18561532 0. 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0.10336056 0.

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.14657362

0. 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0.105301 0. 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0.10837445 0. ]

[0. 0. 0. 0. 0. 0.

0. 0. 0. 0. 0. 0.

0. 0. 0.20483146]]

共性方差（h\_i^2）： [0.75774872 1.00735099 0.93425336 0.90383021 0.90444991 0.87116329

0.87171616 0.91302481 0.80457945 0.83566127 0.91791604 0.87470298

0.9159756 0.91290215 0.81644514]

特殊方差（σ\_i^2）： [0.26352787 0.01392561 0.08702324 0.11744638 0.11682668 0.15011331

0.14956043 0.10825178 0.21669715 0.18561532 0.10336056 0.14657362

0.105301 0.10837445 0.20483146]

总方差： [1.0212766 1.0212766 1.0212766 1.0212766 1.0212766 1.0212766 1.0212766

1.0212766 1.0212766 1.0212766 1.0212766 1.0212766 1.0212766 1.0212766

1.0212766]

总贡献率: 0.8827813388763588