CLUSTERING PRACTICAL

ARTIN

Mathieu Lagrange

reports are to be sent to [mathieu dot lagrange at ec-nantes dot fr] no later than a week after the lab

So Onishi
Send to: modan.tailleur@ls2n.fr

Import tools

please import others if needed

In [146…
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
```

Data: generate a synthetic dataset

Generate a set of 100 points in a 2 dimensional space split into 4 non overlapping clusters.

100点のデータを2次元空間に生成し、それを4つの非重複クラスタに分ける問題

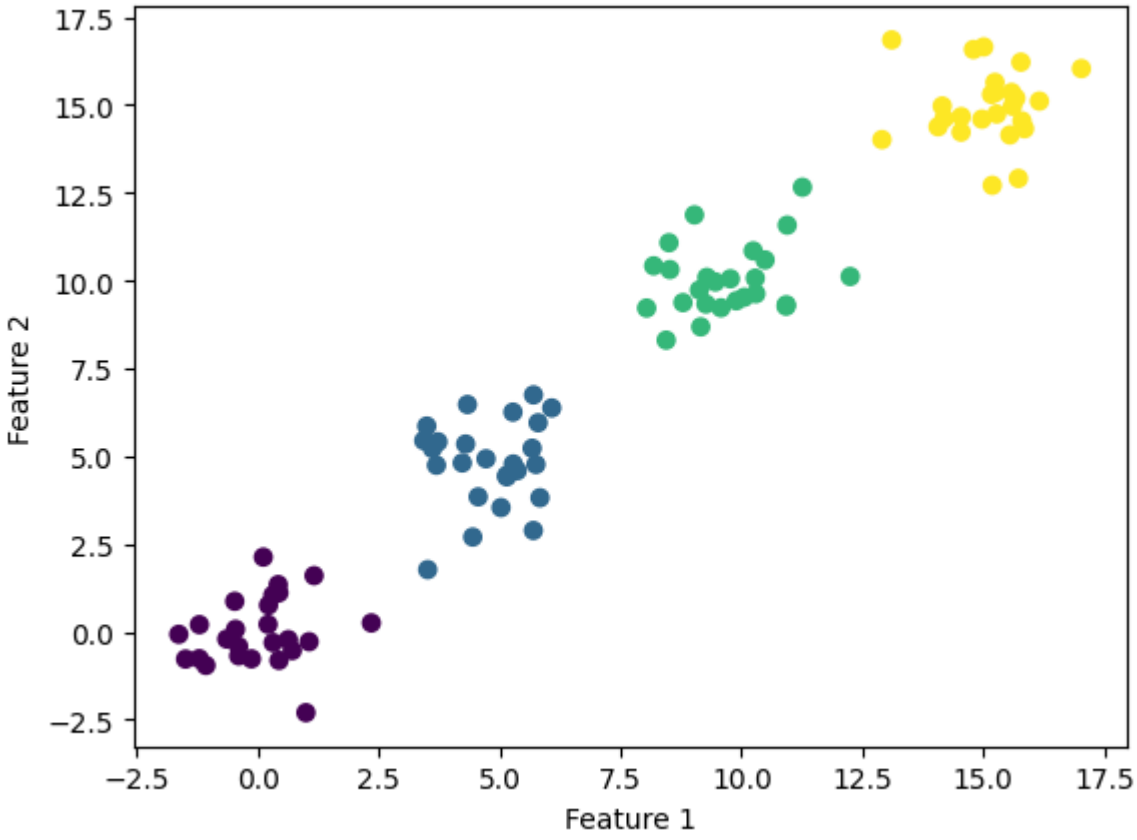
In [147…
```
# 適当なデータセットを作成
# np.random.seed(42)

points1 = np.random.randn(25, 2)
points2 = np.random.randn(25, 2) + np.array([5, 5])
points3 = np.random.randn(25, 2) + np.array([10, 10])
points4 = np.random.randn(25, 2) + np.array([15, 15])

# print("points1:\n" + str(points1))
```

In [148…
```
# 各クラスタにラベルを割り当てる
labels1 = np.zeros(25)  # points1に0のラベルを割り当てる
labels2 = np.ones(25)  # points2に1のラベルを割り当てる
labels3 = np.full(25, 2)  # points3に2のラベルを割り当てる（np.fullで全要素を2に
labels4 = np.full(25, 3)  # points4に3のラベルを割り当てる（np.fullで全要素を3に

# ラベル配列を結合する
truth_labels = np.r_[labels1, labels2, labels3, labels4]

# 確認用
# print(truth_labels)
```

In [149…
```
# データセット結合
points = np.r_[points1, points2, points3, points4]

# 確認用
# print(points)
# print(len(points))
```

Display the set with one color per cluster using the scatter function from matplotlib.pyplot

In [150…
```
plt.scatter(points[:, 0], points[:, 1], c=truth_labels)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



Baseline: the random clustering algorithm

Cluster this dataset into k clusters by assigning a random integer value between 0 and k-1 to each point.

機械学習における「ベースライン（baseline）」は、簡単かつ基本的な方法やモデルを指し、新しいモデルやアプローチの性能を評価するための基準点として使用されます。ベースラインは、複雑なモデルやアルゴリズムが実際に意味のある改善を提供しているかどうかを判断するのに役立ちます。

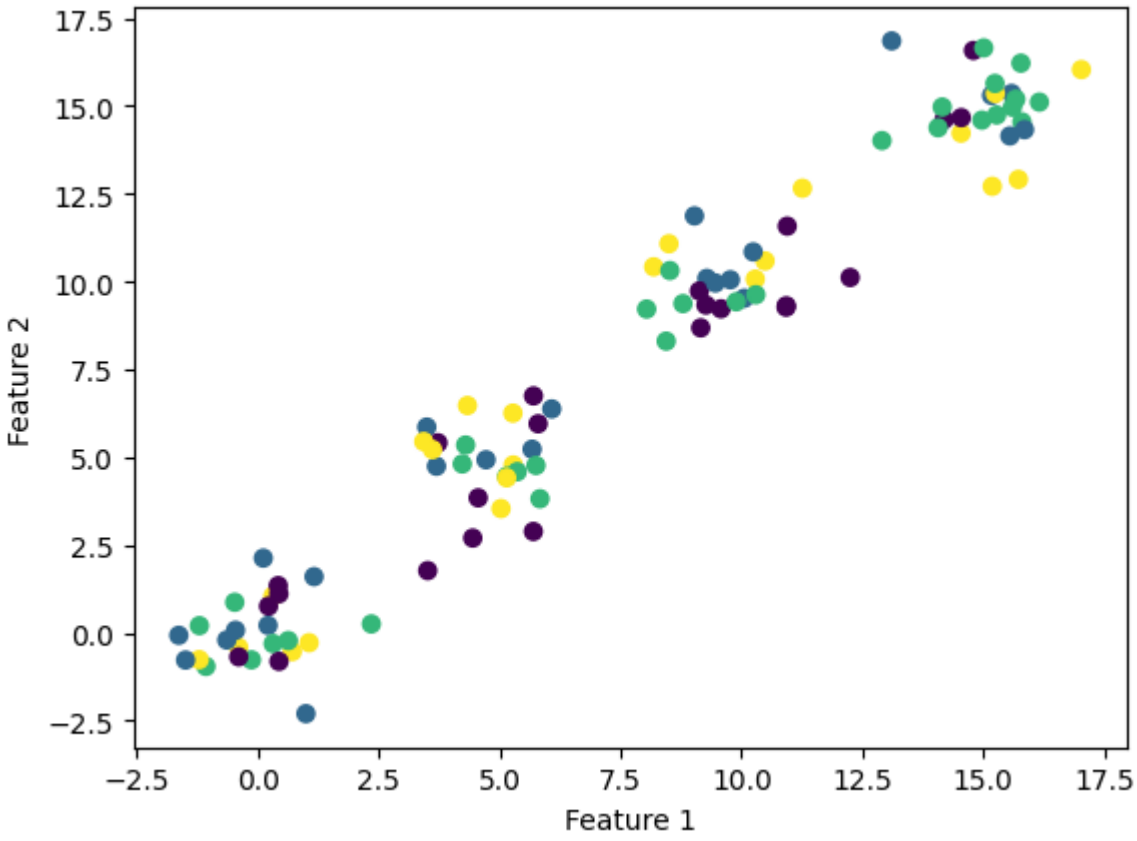
In [151…
```
# ランダムな予測ラベルを生成する関数
def generate_random_labels(num_points, num_classes):
    """
    指定された数のデータポイントに対して、0からnum_classes-1までのランダムなラベルを
```

```
    Parameters:
    num_points (int): ラベルを生成するデータポイントの数。
    num_classes (int): 生成するラベルのクラス数。

    Returns:
    numpy.array: 生成されたランダムラベルの配列。
    """
    return np.random.randint(0, num_points, num_classes)
```

In [152…
```
pred_labels = generate_random_labels(4, len(points))

# 確認用
# print(pred_labels)
```

In [153…
```
plt.scatter(points[:, 0], points[:, 1], c=pred_labels)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
```

Out[153… `Text(0, 0.5, 'Feature 2')`



Metric: the rand index

Implement the rand index criterion (see https://en.wikipedia.org/wiki/Rand_index for reference)

In [154…
```
# ランダムインデックスを計算する関数を定義する
def calculate_rand_index(truth_labels, pred_labels):
    n = len(truth_labels)
    tp = 0  # true positive
    tn = 0  # true negative
```

```
    for i in range(n):
        for j in range(i + 1, n):
            if truth_labels[i] == truth_labels[j] and pred_labels[i] == p
                tp += 1
            elif (
                truth_labels[i] != truth_labels[j] and pred_labels[i] !=
            ):
                tn += 1

    # ランダムインデックスの計算
    rand_index = (tp + tn) / (n * (n - 1) / 2)
    return rand_index
```

In [155…
```
rand_index = calculate_rand_index(truth_labels, pred_labels)  # ランダムイン
```

Compute the rand index between the reference clustering and 100 runs of the baseline algorithm.

基準クラスタリングとベースラインアルゴリズムの100回の実行の間のrandインデックスを計算する。

In [156…
```
def calculate_rand_indices(truth_labels, num_samples, label_size):
    rand_indices = []

    for i in range(num_samples):
        pred_labels = generate_random_labels(label_size, len(truth_labels
        index = calculate_rand_index(truth_labels, pred_labels)
        rand_indices.append(index)

    random_mean_rand_index = np.mean(rand_indices)
    random_std_rand_index = np.std(rand_indices)

    return rand_indices, random_mean_rand_index, random_std_rand_index

rand_indices, random_mean_rand_index, random_std_rand_index = calculate_r
```

Display results and compute the mean and standard deviation.

結果を表示し、平均と標準偏差を計算する。

In [157…
```
print("mean:", random_mean_rand_index)
print("standard deviation:", random_std_rand_index)
```

```
mean: 0.6294222222222222
standard deviation: 0.005988971941819901
```

Hierarchical Clustering

Compute the euclidean distance matrix using the pdist function from scipy.spatial.distance

scipy.spatial.distanceのpdist関数を使用してユークリッド距離行列を計算します。

In [158…
```
from scipy.spatial.distance import pdist, squareform
```

```
# ユークリッド距離を計算
euclidean_distances = pdist(points)
```

Display and interpret its shape

In [159…
```
# 距離行列を表示
print(euclidean_distances)
print(euclidean_distances.shape)
```

```
[0.14056876 0.80780326 0.29952944 ... 1.78928522 1.03713329 0.76765161]
(4950,)
```

answer here

The one-dimensional array representing the Euclidean distances between data points in a dataset has a length of `{n*(n-1)}/2`, where `n` is the number of data points in the dataset. This length corresponds to the number of unique pairwise combinations of points in the dataset. The two-dimensional distance matrix expands these distances into an `n * n` matrix, with the rows and columns representing the data points, and zeroes on the diagonal representing the distance of each point to itself.

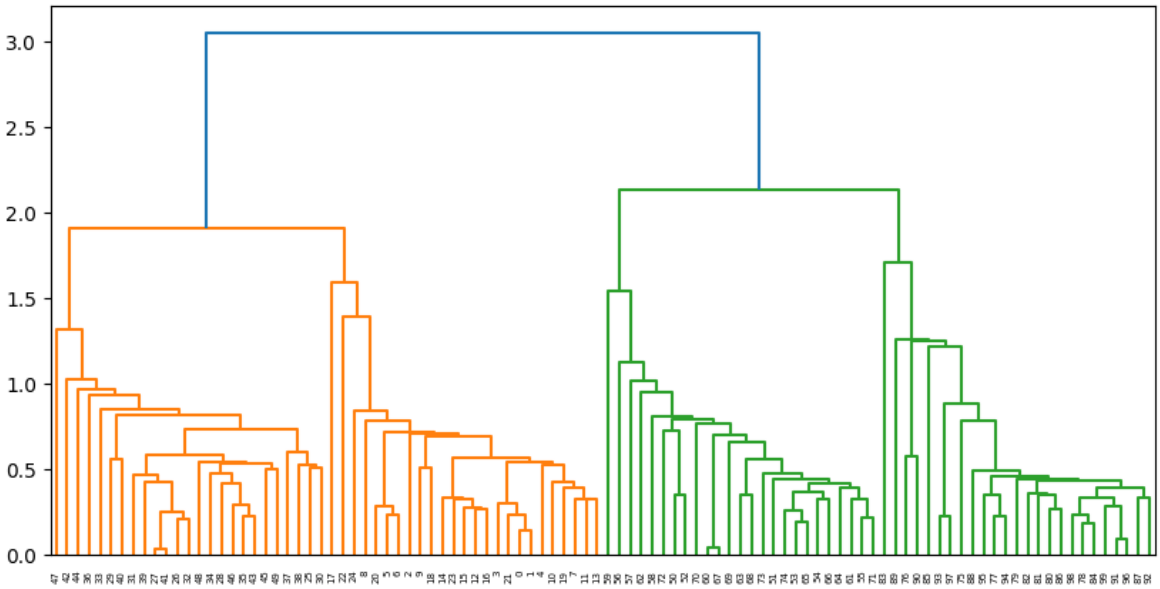
Compute the single link hierarchical clustering using the linkage function from scipy.cluster.hierarchy.

scipy.cluster.hierarchyのリンケージ関数を用いてシングルリンクの階層クラスタリングを計算します。

In [160…
```
from scipy.cluster.hierarchy import linkage, dendrogram

# 距離行列を用いて階層クラスタリングを行う
linkage_matrix = linkage(euclidean_distances, method="single")
```

Display the corresponding dendrogram using the dendrogram function from scipy.cluster.hierarchy.

In [161…
```
# 階層クラスタリングの樹形図を表示する
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix)
plt.show()
```



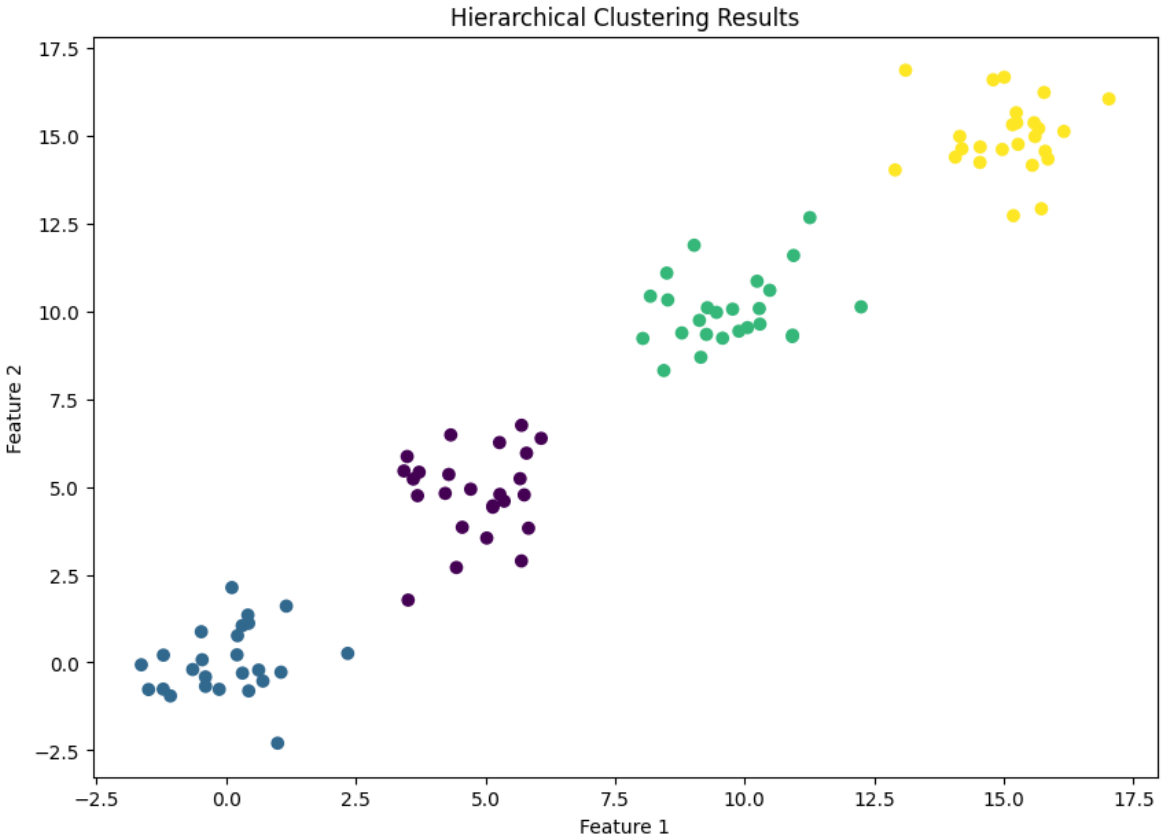
Implement a clustering algorithm that cuts the dendrogram in order to produce k clusters using the fcluster function from scipy.cluster.hierarchy.

scipy.cluster.hierarchyのfcluster関数を使用して、k個のクラスタを生成するためにデンドログラムを切断するクラスタリングアルゴリズムを実装します。

In [162…
```
from scipy.cluster.hierarchy import fcluster


def assign_clusters_by_number(linkage_matrix, num_clusters):
    return fcluster(linkage_matrix, num_clusters, criterion="maxclust")
```

In [163…
```
# クラスタの数
num_clusters = 4

clusters = assign_clusters_by_number(linkage_matrix, num_clusters)

# 確認用
# print(clusters)
```

In [164…
```
# 結果を表示
plt.figure(figsize=(10, 7))
plt.scatter(points[:, 0], points[:, 1], c=clusters)  # Color the points b
plt.title("Hierarchical Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



Compute the rand index between the reference clustering and 100 runs of this clustering algorithm.

参照クラスタリングとこのクラスタリングアルゴリズムの100回の実行の間のrandインデックスを計算する。

In [165…
```
def calculate_euclidean_indices_mean_std(truth_labels, num_samples, linka
    rand_indices = []

    for i in range(100):
        pred_labels = assign_clusters_by_number(linkage_matrix, num_clust

        rand_index = calculate_rand_index(truth_labels, pred_labels)
        rand_indices.append(rand_index)

    euclidean_mean_rand_index = np.mean(rand_indices)
    euclidean_std_rand_index = np.std(rand_indices)

    return rand_indices, euclidean_mean_rand_index, euclidean_std_rand_in

rand_indices, euclidean_mean_rand_index, euclidean_std_rand_index = calcu
```

Display results and compute the mean and standard deviation.

In [166…
```
print("mean:", euclidean_mean_rand_index)
print("standard deviation:", euclidean_std_rand_index)
```

```
mean: 1.0
standard deviation: 0.0
```

Explain why the standard deviation is 0.

answer here

Single-link hierarchical clustering is a deterministic algorithm, which means that it produces the same clustering results every time it is run on the same dataset. This algorithm does not incorporate any randomness, so the output remains unchanged as long as the input does not change. Therefore, the standard deviation of the Rand index over 100 runs is 0, indicating that the algorithm consistently provides the same results.

Partitional Clustering

Implement the k-means algorithm (see https://en.wikipedia.org/wiki/K-means_clustering section Standard algorithm for reference).

Hint: please consider the cdist function from scipy.spatial.distance to compute the distance of the points to the centroids.

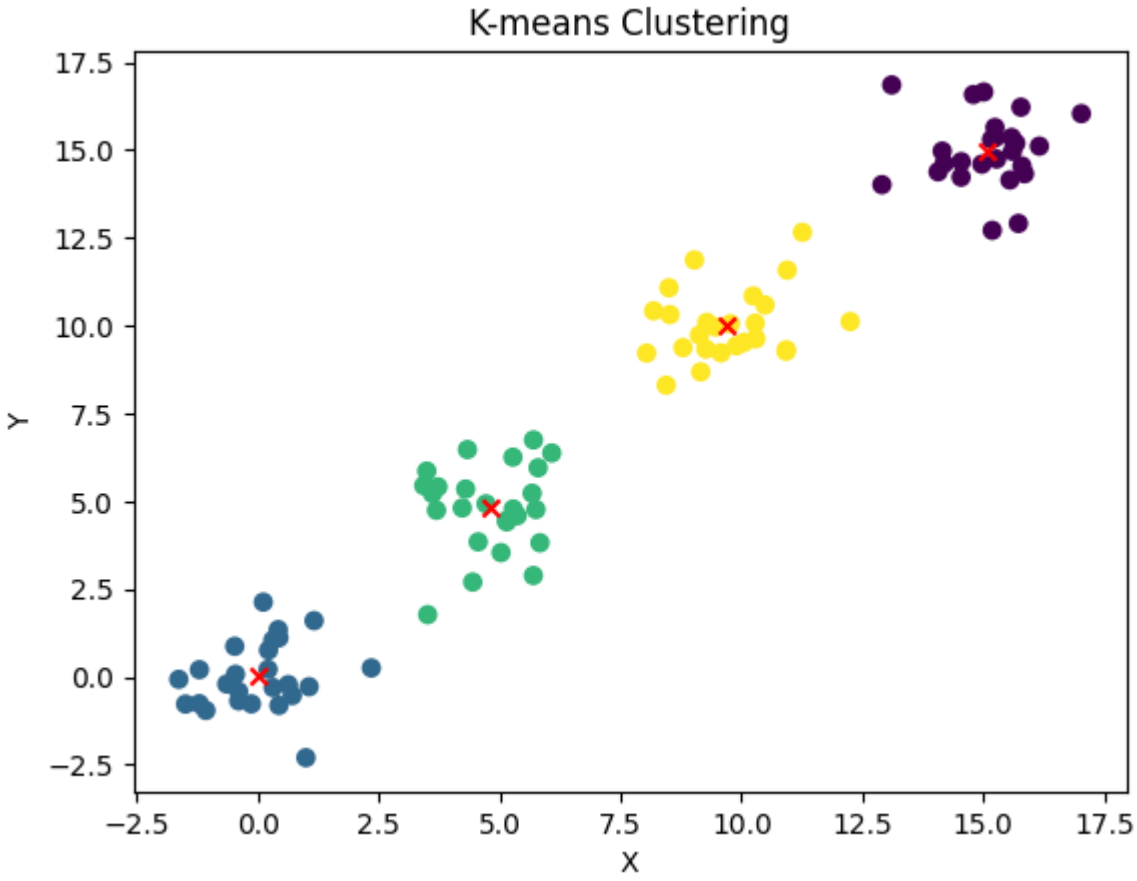
ヒント：セントロイド（中心）に対する点の距離を計算するために、scipy.spatial.distanceのcdist関数を考慮してください。

In [167…

```
from scipy.spatial.distance import cdist

# クラスタの数
num_clusters = 4
# 中心をランダムに選択
centroids = points[np.random.choice(points.shape[0], num_clusters, replac


def k_means(points, centroids, num_clusters):
    while True:
        # 各店に対して最も近い中心を割り当て
        distances = cdist(points, centroids, "euclidean")  # 各点と各中心の
        clusters = np.argmin(distances, axis=1)  # 最も近い中心のインデックス

        # 新しい中心の計算
        new_centroids = []  # クラスタの数に対して空のリストを用意

        # 各クラスタに対してループを実行します。
        for i in range(num_clusters):
            # クラスタiに属するすべてのポイントを選択します。
            cluster_points = points[clusters == i]

            # 選択したポイントの平均を計算します（セントロイドを計算）。
            centroid = cluster_points.mean(axis=0)

            # 計算したセントロイドをリストに追加します。
            new_centroids.append(centroid)

        # リストをNumPy配列に変換します。
        new_centroids = np.array(new_centroids)

        # 収束チェック
        if np.all(centroids == new_centroids):
            break
```

```
        centroids = new_centroids

    return clusters, centroids


clusters, centroids = k_means(points, centroids, num_clusters)
```

In [168…
```
plt.scatter(points[:, 0], points[:, 1], c=clusters, cmap="viridis", marke
plt.scatter(centroids[:, 0], centroids[:, 1], c="red", marker="x")  # セン
plt.title("K-means Clustering")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Compute the rand index between the reference clustering and 100 runs of this clustering algorithm.

参照クラスタリングとこのクラスタリングアルゴリズムの100回の実行の間のrandインデックスを計算する。

In [169…
```
def calculate_k_means_indices_mean_std(truth_labels, num_samples, feature

    rand_indices = []

    for i in range(num_samples):
        # 中心の初期化とk-meansの実行
        initial_centroids = features[np.random.choice(points.shape[0], nu
        pred_labels, _ = k_means(features, initial_centroids, num_cluster

        # ランド指数の計算
```

```
        rand_index = calculate_rand_index(truth_labels, pred_labels)
        # 確認用
        # print(rand_index)
        rand_indices.append(rand_index)

    # 平均値と標準偏差の計算
    k_mean_rand_index = np.mean(rand_indices)
    k_std_rand_index = np.std(rand_indices)

    return rand_indices, k_mean_rand_index, k_std_rand_index
```

Display results and compute the mean and standard deviation.

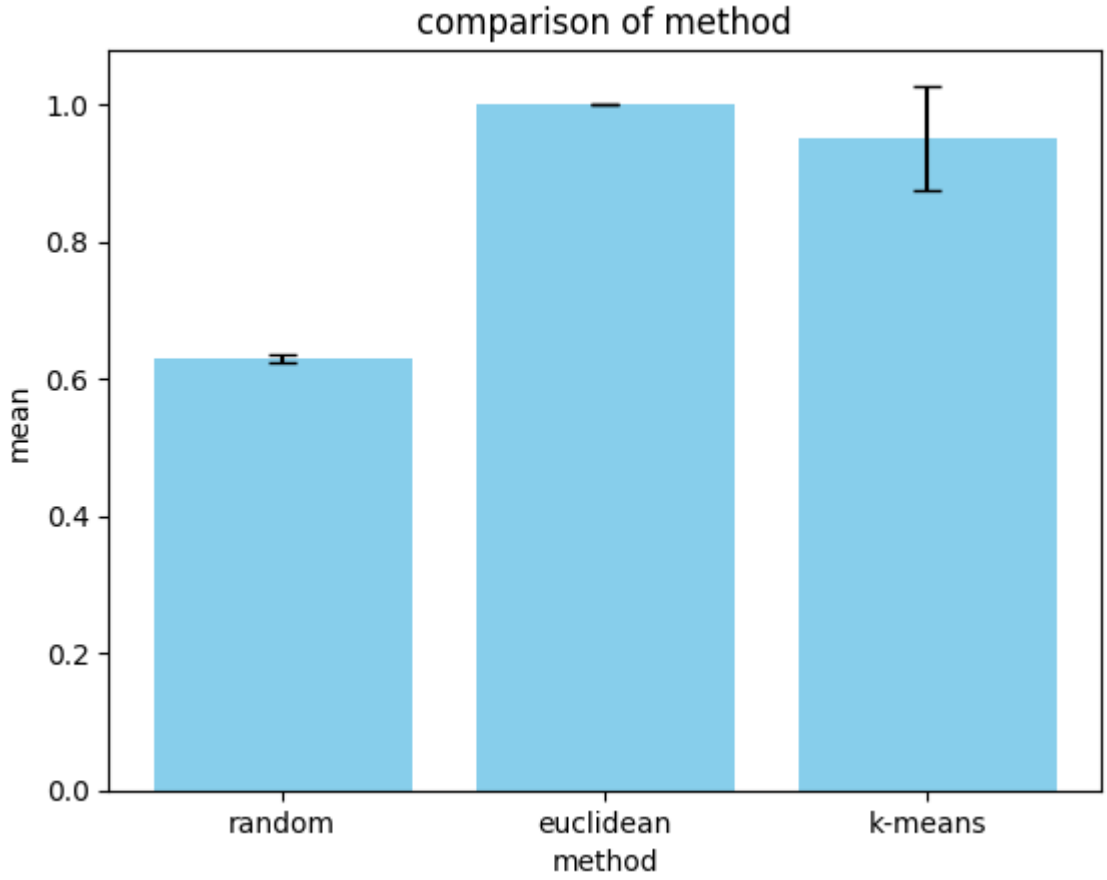
In [170…
```
rand_indices, k_mean_rand_index, k_std_rand_index = calculate_k_means_ind

print("mean:", k_mean_rand_index)
print("standard deviation:", k_std_rand_index)
```

```
mean: 0.951719191919192
standard deviation: 0.07575950701915456
```

Performance Analysis

Display the performance of the 3 clustering algorithms on the synthetic dataset using the bar function from matplotlib.pyplot.

In [171…
```
methods = ["random", "euclidean", "k-means"]
means = [random_mean_rand_index, euclidean_mean_rand_index, k_mean_rand_i
stds = [random_std_rand_index, euclidean_std_rand_index, k_std_rand_index

# 平均値のバープロット
plt.bar(methods, means, yerr=stds, capsize=5, color="skyblue")
plt.xlabel("method")
plt.ylabel("mean")
plt.title("comparison of method")
plt.show()
```



Load the iris dataset using the load_iris function from scikit-learn and perform the same performance analysis using this dataset.

In [172…
```
from sklearn.datasets import load_iris

# irisデータセットの読み込み
iris = load_iris()

features = iris.data  # 特徴量
truth_labels = iris.target  # 真値
targets = iris.target_names # データセットのターゲット名（クラス名）を取得

# 確認用
# print(len(features))
# print(len(truth_labels))
# print(len(targets))
```

In [173…
```
# random
rand_indices, random_mean_rand_index, random_std_rand_index = calculate_r

print("random mean:",random_mean_rand_index)
print("random standard deviation:",random_std_rand_index)
```

```
random mean: 0.5569637583892617
random standard deviation: 0.003664323107609955
```

In [174…
```
# euclid
num_clusters = 3 # クラスタ数

# ユークリッド距離を計算
euclidean_distances = pdist(features)
```

```
# 距離行列を用いて階層クラスタリングを行う
linkage_matrix = linkage(euclidean_distances, method="single")

clusters = assign_clusters_by_number(linkage_matrix, num_clusters)

rand_indices, euclidean_mean_rand_index, euclidean_std_rand_index = calcu


print("mean:", euclidean_mean_rand_index)
print("standard deviation:", euclidean_std_rand_index)
```

```
mean: 0.7766442953020136
standard deviation: 2.220446049250313e-16
```

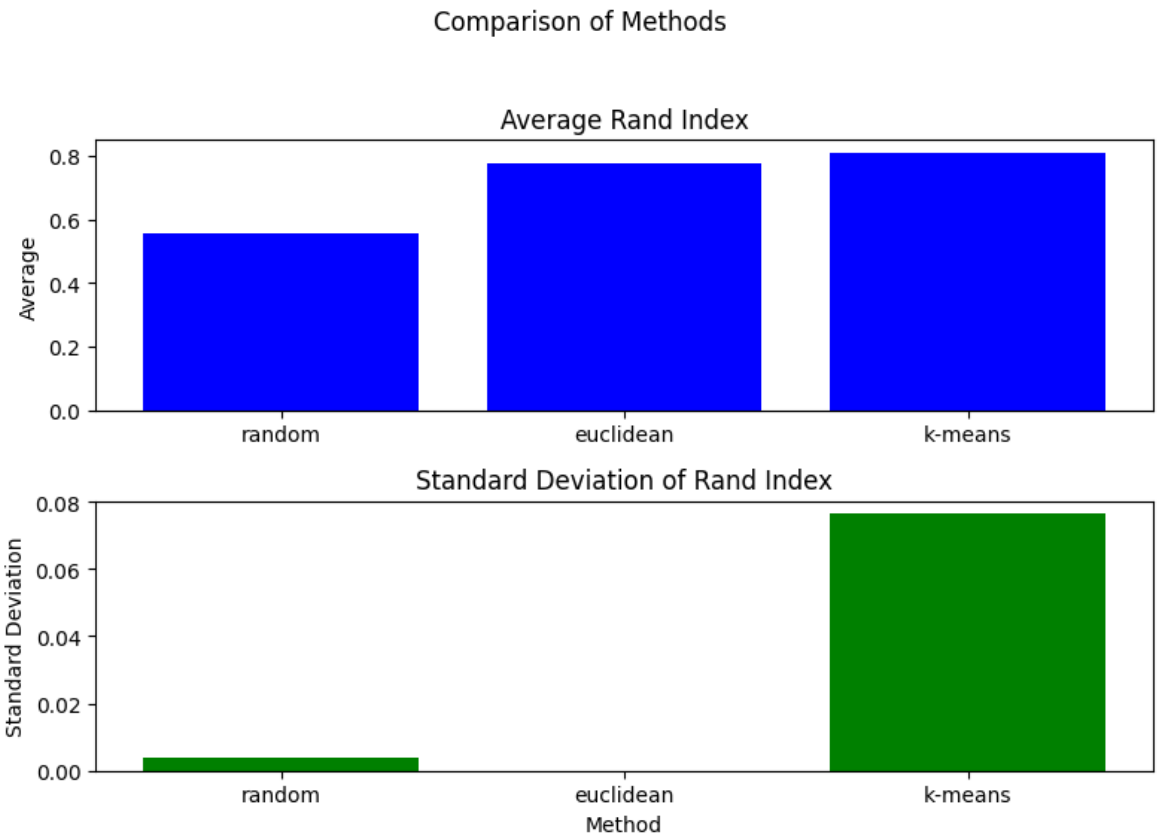
In [175…

```
# k-means
num_clusters = 3  # クラスタ数

rand_indices, k_mean_rand_index, k_std_rand_index = calculate_k_means_ind

print("k-means mean:",k_mean_rand_index)
print("k-means standard deviation:",k_std_rand_index)
```

```
k-means mean: 0.8091910514541385
k-means standard deviation: 0.07647374445048917
```

In [176…

```
import matplotlib.pyplot as plt

methods = ["random", "euclidean", "k-means"]
means = [random_mean_rand_index, euclidean_mean_rand_index, k_mean_rand_i
stds = [random_std_rand_index, euclidean_std_rand_index, k_std_rand_index

# 2つのサブプロットの作成
fig, axs = plt.subplots(2, 1, figsize=(8, 6))

# 平均値のバープロット
axs[0].bar(methods, means, color='blue')
axs[0].set_title("Average Rand Index")
axs[0].set_ylabel("Average")

# 標準偏差のバープロット
axs[1].bar(methods, stds, color='green')
axs[1].set_title("Standard Deviation of Rand Index")
axs[1].set_ylabel("Standard Deviation")

# 全体のタイトルとラベルの設定
plt.xlabel("Method")
plt.suptitle("Comparison of Methods")

# プロットの表示
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Load the Breast cancer wisconsin (diagnostic) dataset dataset using the load_breast_cancer function from scikit-learn and perform the same performance analysis using this dataset.

In [177…
```
# scikit-learnから乳がんデータセットをインポート
from sklearn.datasets import load_breast_cancer

# 乳がんデータセットをロード
breast_cancer_dataset = load_breast_cancer()

# データセットから特徴量（データ）を取得
features = breast_cancer_dataset.data

# データセットからターゲット（ラベル）を取得
truth_labels = breast_cancer_dataset.target

# データセットのターゲット名（クラス名）を取得
class_names = breast_cancer_dataset.target_names

# 確認用
# print(features)
# print(truth_labels)
# print(class_names)
```

In [178…
```
# random
rand_indices, random_mean_rand_index, random_std_rand_index = calculate_r

print("random mean:",random_mean_rand_index)
print("random standard deviation:",random_std_rand_index)
```

```
random mean: 0.499940840119805
random standard deviation: 0.0009936223422790749
```

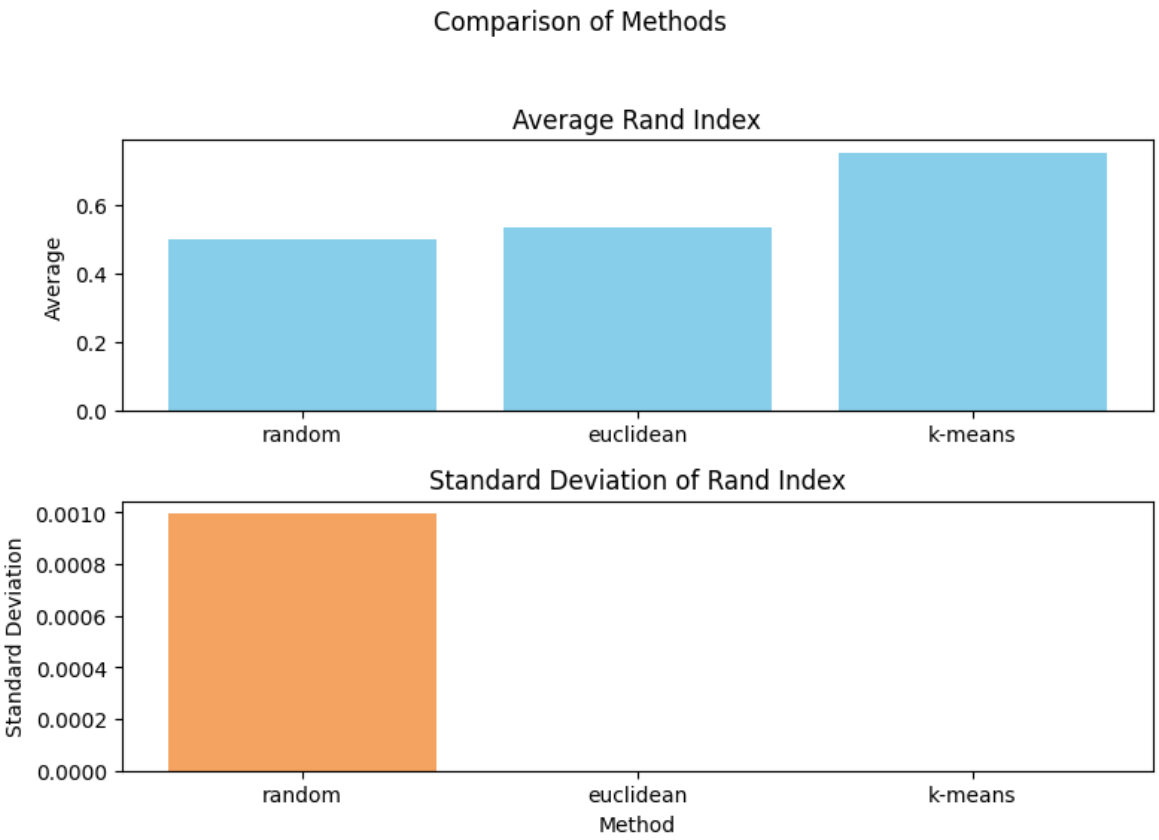
In [179…
```
# euclid
num_clusters = 2 # クラスタ数

# ユークリッド距離を計算
euclidean_distances = pdist(features)

# 距離行列を用いて階層クラスタリングを行う
linkage_matrix = linkage(euclidean_distances, method="single")

clusters = assign_clusters_by_number(linkage_matrix, num_clusters)

rand_indices, euclidean_mean_rand_index, euclidean_std_rand_index = calcu

print("mean:", euclidean_mean_rand_index)
print("standard deviation:", euclidean_std_rand_index)
```

```
mean: 0.5325503106512537
standard deviation: 0.0
```

In [180…
```
# k-means
num_clusters = 2  # クラスタ数

rand_indices, k_mean_rand_index, k_std_rand_index = calculate_k_means_ind

print("k-means mean:",k_mean_rand_index)
print("k-means standard deviation:",k_std_rand_index)
```

```
k-means mean: 0.7503774845912027
k-means standard deviation: 1.1102230246251565e-16
```

In [181…
```
methods = ["random", "euclidean", "k-means"]
means = [random_mean_rand_index, euclidean_mean_rand_index, k_mean_rand_i
stds = [random_std_rand_index, euclidean_std_rand_index, k_std_rand_index

# 2つのサブプロットの作成
fig, axs = plt.subplots(2, 1, figsize=(8, 6))

# 平均値のバープロット
axs[0].bar(methods, means, color='skyblue')
axs[0].set_title("Average Rand Index")
axs[0].set_ylabel("Average")

# 標準偏差のバープロット
axs[1].bar(methods, stds, color='sandybrown')
axs[1].set_title("Standard Deviation of Rand Index")
axs[1].set_ylabel("Standard Deviation")

# 全体のタイトルとラベルの設定
plt.xlabel("Method")
plt.suptitle("Comparison of Methods")

# プロットの表示
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



[Bonus] Determining the number of clusters

Implement the gap statistic method for determining the optimal number of clusters for the 3 datasets.

In [ ]:

Discuss the results.

answer here.

In [ ]: