```
!pip install scikit-learn scipy matplotlib
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: scikit-learn in
/home/greygosling/.local/lib/python3.12/site-packages (1.4.2)
Requirement already satisfied: scipy in
/home/greygosling/.local/lib/python3.12/site-packages (1.13.0)
Requirement already satisfied: matplotlib in
/home/greygosling/.local/lib/python3.12/site-packages (3.8.4)
Requirement already satisfied: numpy>=1.19.5 in
/home/greygosling/.local/lib/python3.12/site-packages (from scikit-
learn) (1.26.4)
Requirement already satisfied: joblib>=1.2.0 in
/home/greygosling/.local/lib/python3.12/site-packages (from scikit-
learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/greygosling/.local/lib/python3.12/site-packages (from scikit-
learn) (3.4.0)
Requirement already satisfied: contourpy>=1.0.1 in
/home/greygosling/.local/lib/python3.12/site-packages (from
matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/home/greygosling/.local/lib/python3.12/site-packages (from
matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/greygosling/.local/lib/python3.12/site-packages (from
matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/greygosling/.local/lib/python3.12/site-packages (from
matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/lib/python3.12/site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=8 in
/usr/lib64/python3.12/site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/greygosling/.local/lib/python3.12/site-packages (from
matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/lib/python3.12/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

Алгоритмы кластеризации

Задание на сегодня состоит из двух задач. Нужно реализовать алгоритмы k-means и иерархическую кластеризацию.

1. K-Means (1 балл)

Реализуйте метод сжатия изображений в формате PNG с помощью кластеризации пикселей.

Общая схема работы метода:

- С помощью алгоритма Ллойда построить по изображению набор из **K** базовых цветов. Базовый цвет – это центроид в пространстве RGB.
- Преобразовать исходное изображение в новое, в котором каждый пиксель заменен на ближайший к нему базовый цвет.

Необходимо, чтобы преобразованное изображение визуально не сильно отличалось от исходного

Lloyd's Algorithm

- Arbitrarily assign the ***K*** cluster centers
- **while** cluster centers keep changing:
  a. Compute the distance from each data point to centers, and assign point to the nearest center
  b. compute new centers for each cluster by taking the centroid of all the points in that cluster

Алгоритм минимизирует функцию ошибки (**loss function**) вида

$$\mathrm{D}(X, C) = \frac{1}{N} \sum_{i=1,\ldots,N} \min_{c \in C} dist^2(x_i, c)$$

Расстояние между точками евклидово

$$\mathrm{dist}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Для проверки критерия сходимости можно использовать функцию `np.allclose(x, y)`

```
np.allclose([[1,2,3], [1,2,3]], [[1,2,3], [1,2,2.95]], atol=0.01)
```

```
False
```

```
np.allclose([[1,2,3], [1,2,3]], [[1,2,3], [1,2,2.95]], atol=0.1)
```

```
True
```

Загрузка данных

```
# Загрузим картинку
img = plt.imread("flower.png")[:,:,:3]

# Теперь в переменной img находится трехмерный массив чисел.
print(img.shape)
```

```
(427, 640, 3)
```

```
_ = plt.imshow(img)
```

```
# Цвет каждой точки задается трехмерным вектором в пространстве RGB.
Это наши признаки.
print(img[400, 500])
```

```
[0.04313726 0.12156863 0.11372549]
```

```
# Преобразуем массив в двумерный. Теперь первая координата — номер
точки, вторая — ее признаки
X = img.reshape((-1,3))
print(X.shape)
```

```
(273280, 3)
```

Кластеризация

Примерная структура класса. Можно менять, если хотите

```
from scipy.spatial import distance
import numpy as np

class KMeans(object):
    def __init__(self, n_clusters, max_iter, seed=0, tol=0.001):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.centroids = None
        self.losses = []
        self.seed = seed
        self.tol = tol

    def distances(self, X):
        dist = np.array([distance.cdist(X, [centroid]) for centroid in
self.centroids])
        return dist.transpose(1, 0, 2).reshape(X.shape[0],
self.n_clusters)

    def loss(self, dist):
        min_dist = np.min(dist, axis=1)
        return np.sum(min_dist)

    def initialize(self, X):
        np.random.seed(self.seed)
        random_indices = np.random.choice(X.shape[0], self.n_clusters,
replace=False)
        self.centroids = X[random_indices]

    def fit_predict(self, X):
        self.initialize(X)
        labels = np.zeros(X.shape[0], dtype=int)

        for i in range(self.max_iter):
```

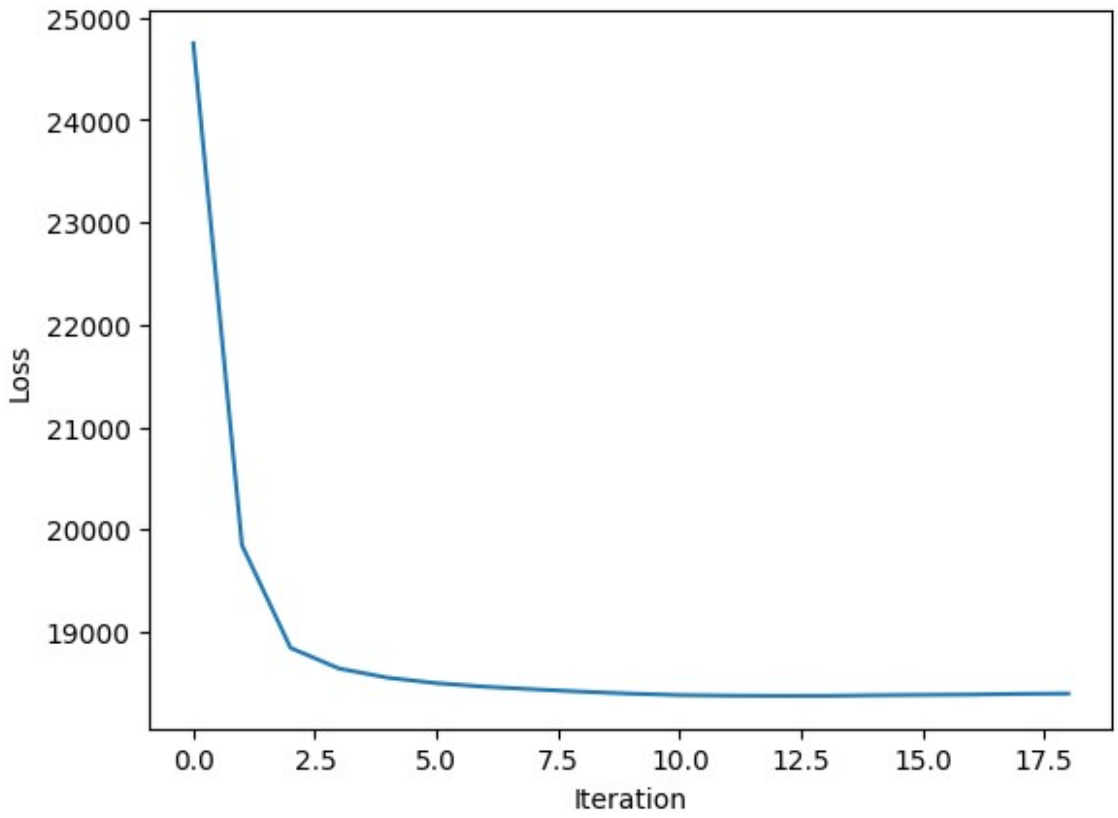
```
            dist = self.distances(X)
            labels = np.argmin(dist, axis=1)
            new_centroids = np.array([X[labels == j].mean(axis=0) for
j in range(self.n_clusters)])
            loss = self.loss(dist)

            self.losses.append(loss)
            if np.allclose(self.centroids, new_centroids,
atol=self.tol):
                break
            self.centroids = new_centroids

        return labels
```

Запустим кластеризацию на наших данных

```
cls = KMeans(n_clusters=8, max_iter=100, seed=0)
y_pred = cls.fit_predict(X)
losses = cls.losses
centroids = cls.centroids
```

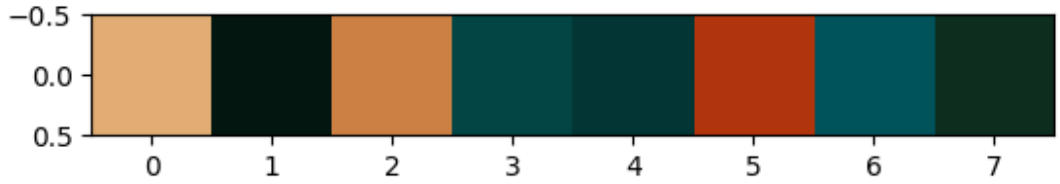
Посмотрим, как убывала функция ошибки

```
_ = plt.plot(losses)
_ = plt.xlabel('Iteration')
_ = plt.ylabel('Loss')
```



Наши центроиды – цвета в пространстве RGB. Можно их нарисовать

```
_ = plt.imshow([centroids])
```

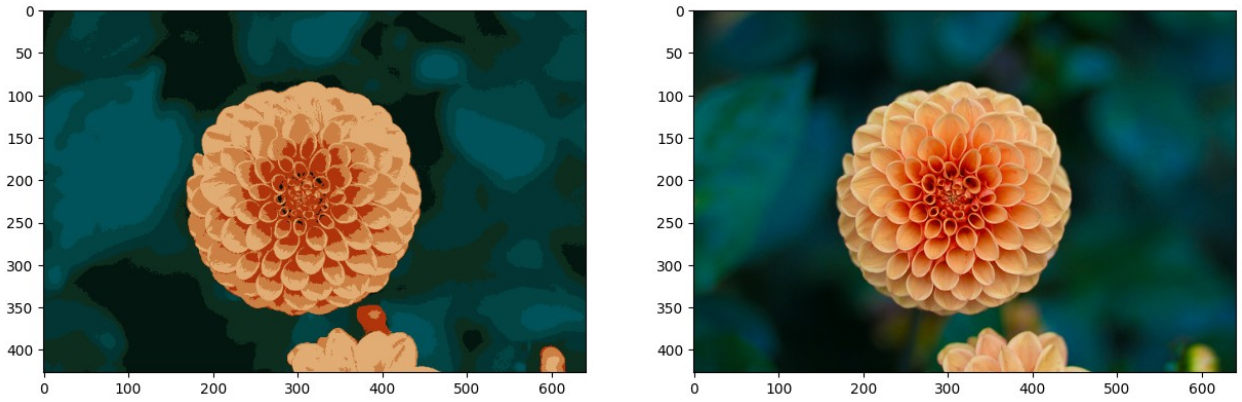


Теперь сделайте из `X` матрицу `Y`, в которой координаты каждой точки заменены на координаты центроида

```
Y = cls.centroids[y_pred]
```

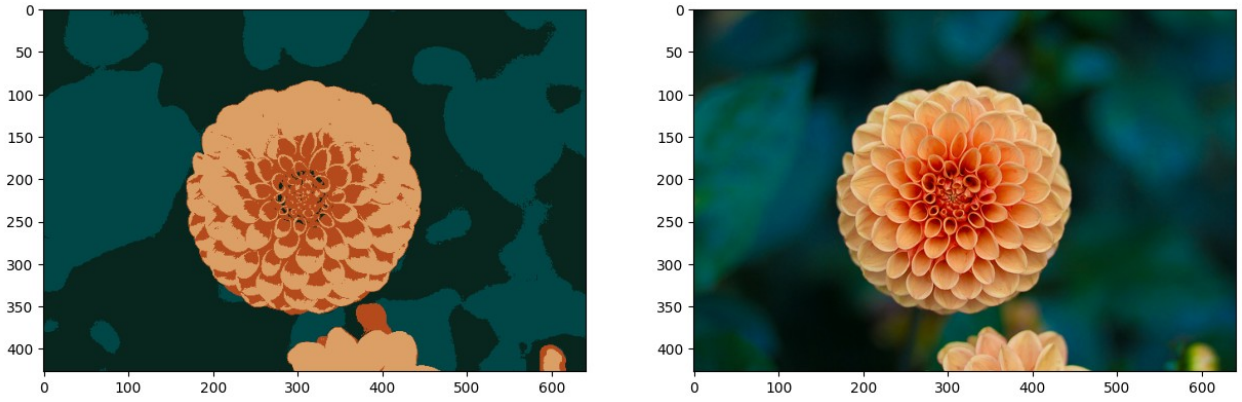
Посмотрим, что получилось

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
_ = ax1.imshow(np.array(Y).reshape(img.shape))
_ = ax2.imshow(img)
```

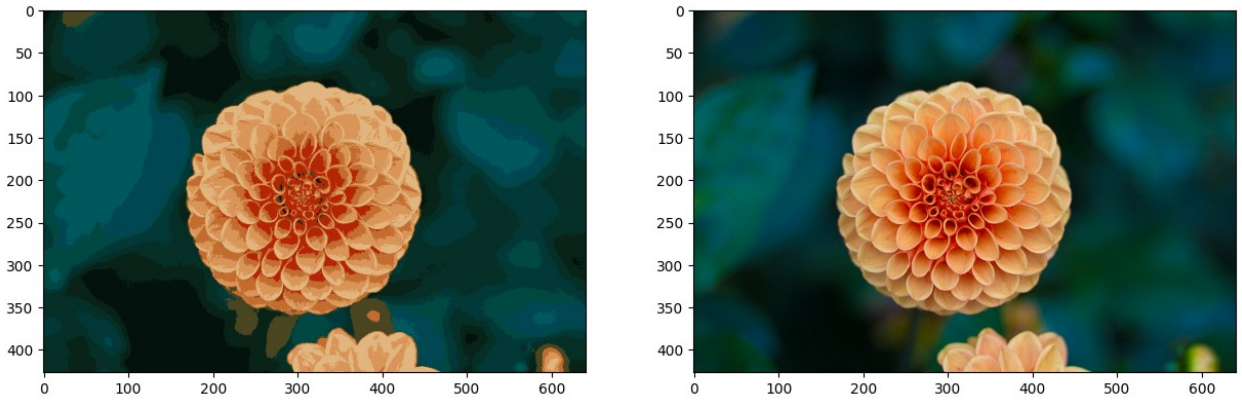


Поэкспериментируйте с разным числом кластеров

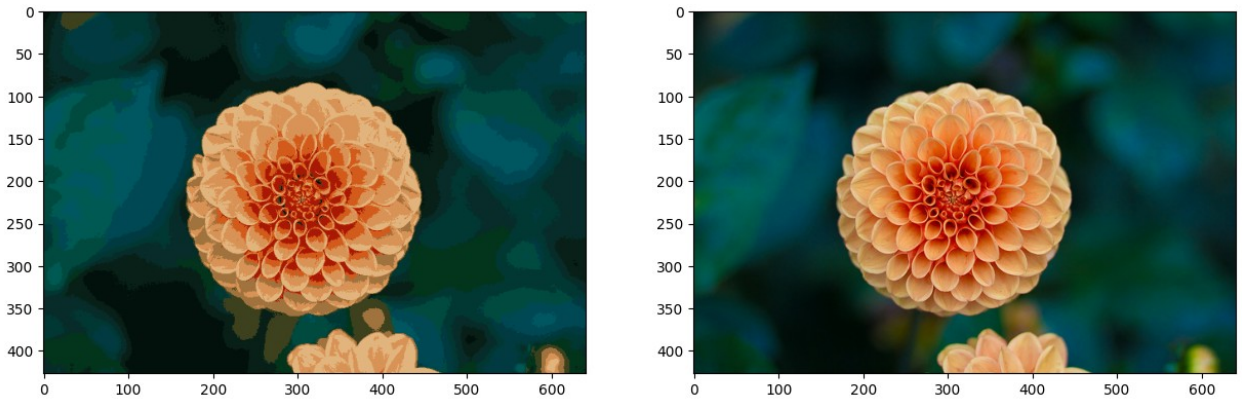
```
cls = KMeans(n_clusters=4, max_iter=100, seed=0)
y_pred = cls.fit_predict(X)
losses = cls.losses
centroids = cls.centroids

Y = centroids[y_pred, :]

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
_ = ax1.imshow(np.array(Y).reshape(img.shape))
_ = ax2.imshow(img)
```



```
cls = KMeans(n_clusters=12, max_iter=100, seed=0)
y_pred = cls.fit_predict(X)
losses = cls.losses
centroids = cls.centroids

Y = centroids[y_pred, :]

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
_ = ax1.imshow(np.array(Y).reshape(img.shape))
_ = ax2.imshow(img)
```



```
cls = KMeans(n_clusters=16, max_iter=100, seed=0)
y_pred = cls.fit_predict(X)
losses = cls.losses
centroids = cls.centroids

Y = centroids[y_pred, :]

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
_ = ax1.imshow(np.array(Y).reshape(img.shape))
_ = ax2.imshow(img)
```



2. Иерархическая кластеризация (1 балл)

Реализуйте алгоритм иерархической кластеризации

HIERARCHICALCLUSTERING($\mathbf{d}, n$)

```
1  Form n clusters, each with 1 element
2  Construct a graph T by assigning an isolated vertex to each cluster
3  while there is more than 1 cluster
4      Find the two closest clusters C1 and C2
5      Merge C1 and C2 into new cluster C with |C1| + |C2| elements
6      Compute distance from C to all other clusters
7      Add a new vertex C to T and connect to vertices C1 and C2
8      Remove rows and columns of d corresponding to C1 and C2
9      Add a row and column to d for the new cluster C
10 return T
```

Для вычисления расстояний между кластерами используйте среднее расстояние между входящими в них точками:

$$d_{avg}(C^*, C) = \frac{1}{|C^*||C|} \sum_{x \in C^*, y \in C} d(x, y)$$

2.1. Реализуйте функцию `distance_matrix`, вычисляющую матрицу попарных расстояний между точками.

В качестве метрики мы будем использовать евклидово расстояние,

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

```python
def distance_matrix(X):
    '''
    Compute matrix of pair-wise distances between samples
    :param X: array-like of shape (n_samples, n_features)
        Samples
    :return D: array-like of shape (n_samples, n_samples)
        Matrix of pair-wise distances between samples
    '''
    D = distance.cdist(X, X, 'euclidean')
    return D
```

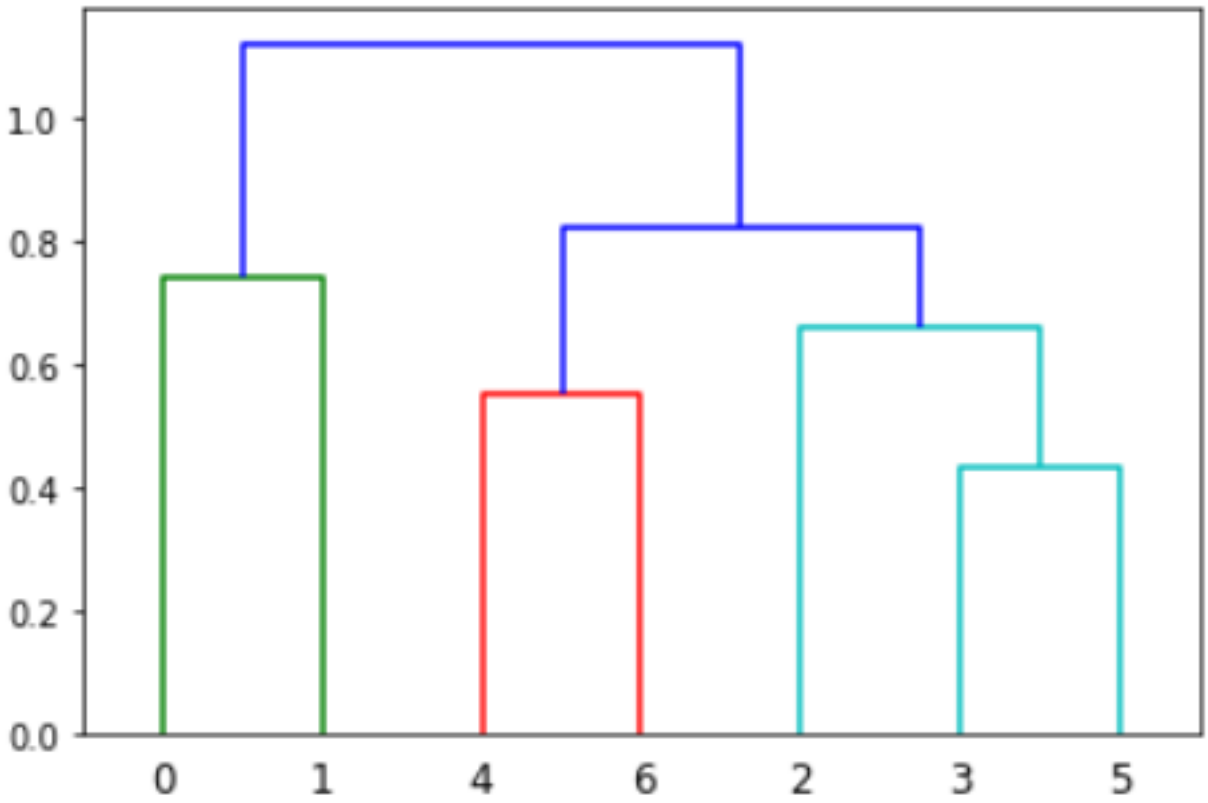
2.2. Реализуйте функцию `linkage`.

На вход подается матрица расстояний. На выходе – результат кластеризации в виде матрицы связей. Строки этой матрицы соответствуют операции объединения кластеров. Каждая строка имеет вид

`[C1, C2, dist(C1, C2), N]`

где `C1` и `C2` – номера объединяемых кластеров, `dist(C1, C2)` – расстояние между объединяемыми кластерами, `N` – число точек в новом кластере

Например, кластеризации



соответствует

```
Z = [[ 3.    5.    0.43  2.  ]
     [ 4.    6.    0.55  2.  ]
     [ 2.    7.    0.66  3.  ]
     [ 0.    1.    0.74  2.  ]
     [ 8.    9.    0.82  5.  ]
     [10.   11.    1.12  7.  ]]
```

```
def linkage(D):
    '''
    :param D: array-like of shape (n_samples, n_samples)
            Matrix of pair-wise distances between samples using
average linkage
```

```
    :return linkage: Linkage matrix
    '''
    n = D.shape[0]
    linkage_matrix = []
    clusters = {i: [i] for i in range(n)}
    active_clusters = list(clusters.keys())

    while len(active_clusters) > 1:
        min_dist = np.inf
        closest_pair = (None, None)

        # Перебираем все активные кластеры для поиска пары с
наименьшим средним расстоянием
        for i in range(len(active_clusters)):
            for j in range(i + 1, len(active_clusters)):
                a, b = active_clusters[i], active_clusters[j]
                # Используем среднее расстояние для вычисления
расстояния между кластерами
                if len(clusters[a]) > 0 and len(clusters[b]) > 0:
                    dist_ab = np.mean(D[np.ix_(clusters[a],
clusters[b])])
                    if dist_ab < min_dist:
                        min_dist = dist_ab
                        closest_pair = (a, b)

        ci, cj = closest_pair
        new_cluster = clusters[ci] + clusters[cj]
        new_index = max(clusters.keys()) + 1
        clusters[new_index] = new_cluster

        linkage_matrix.append([ci, cj, min_dist, len(new_cluster)])

        # Обновляем список активных кластеров
        active_clusters = [x for x in active_clusters if x != ci and x
!= cj] + [new_index]

    return np.array(linkage_matrix)
```

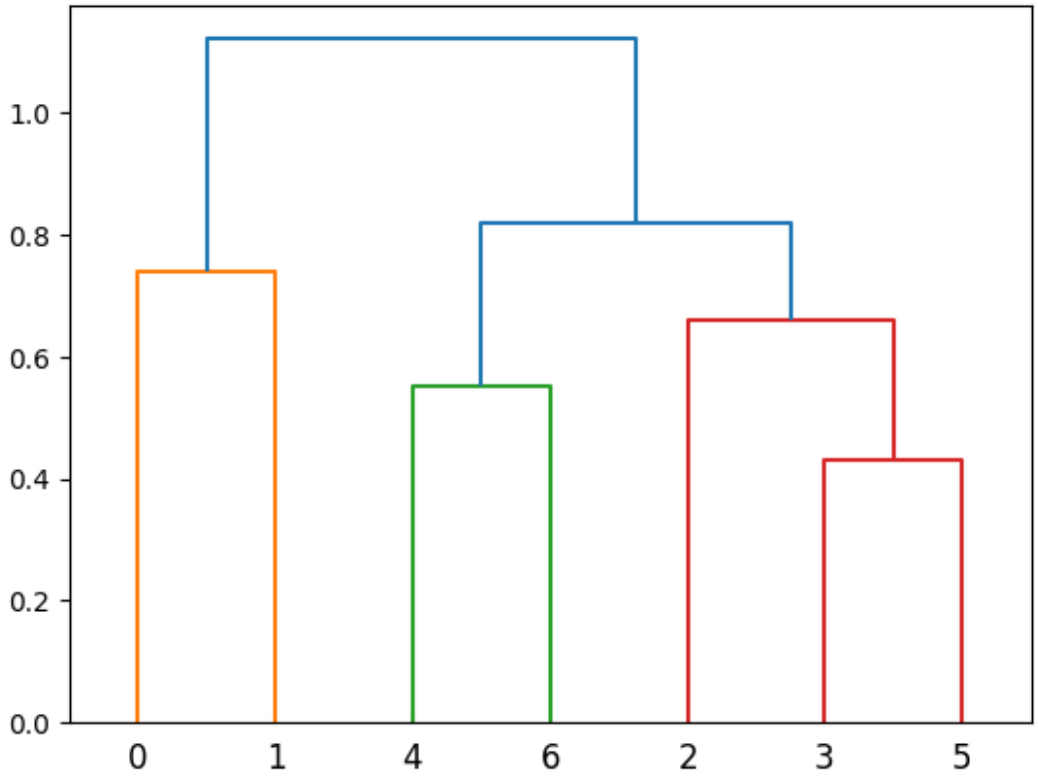
Проверка 1

```
# Нам дана матрица расстояний
D = np.array([[0.  , 0.74, 0.85, 0.54, 0.83, 0.92, 0.89],
              [0.74, 0.  , 1.59, 1.35, 1.2 , 1.48, 1.55],
              [0.85, 1.59, 0.  , 0.63, 1.13, 0.69, 0.73],
              [0.54, 1.35, 0.63, 0.  , 0.66, 0.43, 0.88],
              [0.83, 1.2 , 1.13, 0.66, 0.  , 0.72, 0.55],
              [0.92, 1.48, 0.69, 0.43, 0.72, 0.  , 0.8 ],
              [0.89, 1.55, 0.73, 0.88, 0.55, 0.8 , 0.  ]])

# кластеризуем
```

```
Z = linkage(D)

# и напечатаем что получилось
print(np.array(Z))
```

```
[[ 3.    5.    0.43  2.  ]
 [ 4.    6.    0.55  2.  ]
 [ 2.    7.    0.66  3.  ]
 [ 0.    1.    0.74  2.  ]
 [ 8.    9.    0.82  5.  ]
 [10.   11.    1.12  7.  ]]
```

```
# для построения дендрограммы воспользуемся функцией из библиотеки scipy
from scipy.cluster.hierarchy import dendrogram

# должна получиться картинка из начала этого раздела
_ = dendrogram(Z)
```



Проверка 2

```
D = np.array([
    [0.  , 0.43, 0.93, 0.85, 0.94, 0.7 , 0.95, 0.79, 0.89, 1.34, 0.8 , 0.64, 1.42, 1.37, 0.86, 1.2 , 0.49, 0.19, 1.48, 0.86],
    [0.43, 0.  , 0.66, 0.7 , 1.48, 0.58, 0.7 , 0.55, 0.68, 1.45, 1.31, 0.37, 1.24, 1.69, 0.91, 1.28, 1.04, 0.29, 0.94, 1.05],
```

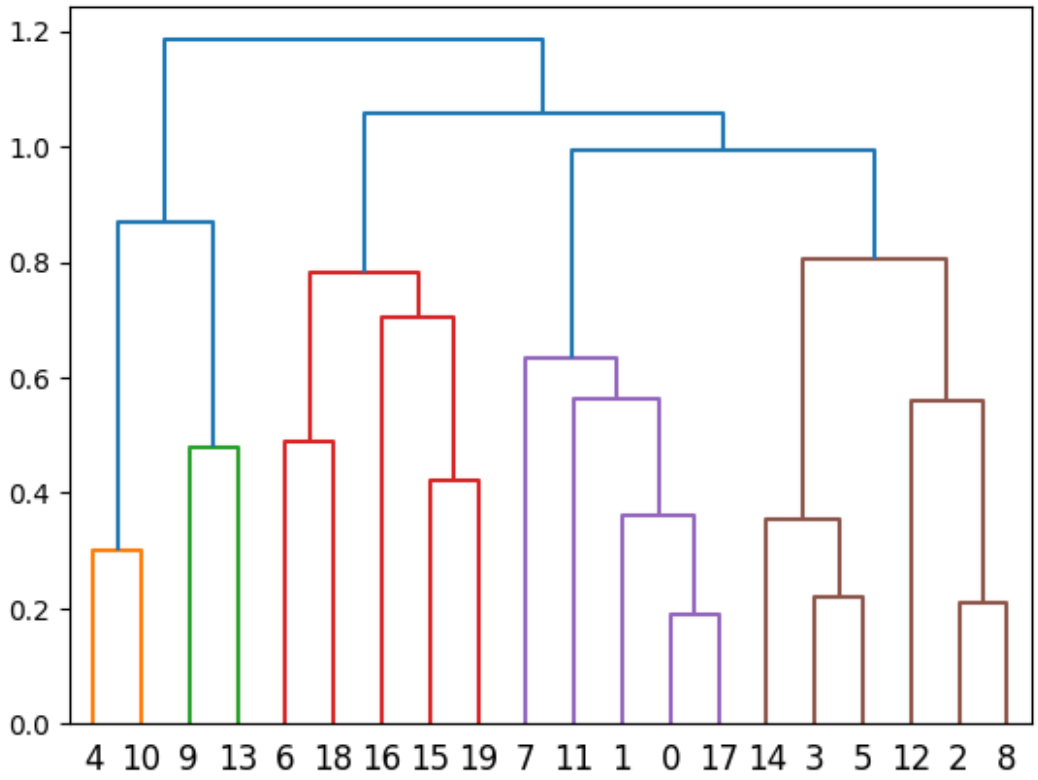
```
    [0.93, 0.66, 0.  , 1.01, 1.42, 0.8 , 0.76, 1.18, 0.21, 1.18, 1.35,
0.82, 0.68, 0.85, 0.75, 1.01, 0.93, 1.03, 0.6 , 0.91],
    [0.85, 0.7 , 1.01, 0.  , 1.02, 0.22, 0.89, 0.91, 0.79, 1.47, 1.04,
1.51, 0.71, 1.63, 0.34, 1.05, 1.24, 0.56, 1.09, 1.25],
    [0.94, 1.48, 1.42, 1.02, 0.  , 1.46, 1.48, 0.95, 1.44, 1.08, 0.3 ,
1.29, 1.28, 0.65, 1.  , 0.8 , 0.8 , 0.92, 1.48, 0.65],
    [0.7 , 0.58, 0.8 , 0.22, 1.46, 0.  , 0.77, 1.  , 0.65, 1.25, 1.27,
1.36, 0.81, 1.56, 0.37, 0.98, 1.17, 0.68, 0.88, 1.23],
    [0.95, 0.7 , 0.76, 0.89, 1.48, 0.77, 0.  , 1.41, 1.08, 1.61, 1.65,
0.89, 0.76, 1.32, 0.64, 0.7 , 0.59, 1.07, 0.49, 0.91],
    [0.79, 0.55, 1.18, 0.91, 0.95, 1.  , 1.41, 0.  , 1.05, 0.73, 1.08,
0.7 , 1.49, 1.17, 1.  , 1.34, 1.45, 0.49, 1.06, 1.3 ],
    [0.89, 0.68, 0.21, 0.79, 1.44, 0.65, 1.08, 1.05, 0.  , 0.96, 1.09,
0.94, 0.44, 1.06, 0.9 , 1.47, 1.2 , 0.79, 1.04, 1.39],
    [1.34, 1.45, 1.18, 1.47, 1.08, 1.25, 1.61, 0.73, 0.96, 0.  , 0.96,
1.1 , 1.05, 0.48, 1.36, 1.26, 1.38, 1.38, 1.03, 1.38],
    [0.8 , 1.31, 1.35, 1.04, 0.3 , 1.27, 1.65, 1.08, 1.09, 0.96, 0.  ,
1.08, 1.09, 0.79, 1.4 , 1.03, 1.02, 0.78, 1.79, 0.86],
    [0.64, 0.37, 0.82, 1.51, 1.29, 1.36, 0.89, 0.7 , 0.94, 1.1 , 1.08,
0.  , 1.42, 1.2 , 1.61, 1.3 , 0.86, 0.68, 1.04, 0.83],
    [1.42, 1.24, 0.68, 0.71, 1.28, 0.81, 0.76, 1.49, 0.44, 1.05, 1.09,
1.42, 0.  , 0.99, 0.84, 1.2 , 1.21, 1.22, 0.97, 1.58],
    [1.37, 1.69, 0.85, 1.63, 0.65, 1.56, 1.32, 1.17, 1.06, 0.48, 0.79,
1.2 , 0.99, 0.  , 1.13, 0.61, 1.  , 1.6 , 0.81, 0.83],
    [0.86, 0.91, 0.75, 0.34, 1.  , 0.37, 0.64, 1.  , 0.9 , 1.36, 1.4 ,
1.61, 0.84, 1.13, 0.  , 0.68, 0.89, 0.86, 0.65, 1.04],
    [1.2 , 1.28, 1.01, 1.05, 0.8 , 0.98, 0.7 , 1.34, 1.47, 1.26, 1.03,
1.3 , 1.2 , 0.61, 0.68, 0.  , 0.92, 1.47, 0.47, 0.42],
    [0.49, 1.04, 0.93, 1.24, 0.8 , 1.17, 0.59, 1.45, 1.2 , 1.38, 1.02,
0.86, 1.21, 1.  , 0.89, 0.92, 0.  , 0.9 , 1.18, 0.49],
    [0.19, 0.29, 1.03, 0.56, 0.92, 0.68, 1.07, 0.49, 0.79, 1.38, 0.78,
0.68, 1.22, 1.6 , 0.86, 1.47, 0.9 , 0.  , 1.56, 1.21],
    [1.48, 0.94, 0.6 , 1.09, 1.48, 0.88, 0.49, 1.06, 1.04, 1.03, 1.79,
1.04, 0.97, 0.81, 0.65, 0.47, 1.18, 1.56, 0.  , 0.84],
    [0.86, 1.05, 0.91, 1.25, 0.65, 1.23, 0.91, 1.3 , 1.39, 1.38, 0.86,
0.83, 1.58, 0.83, 1.04, 0.42, 0.49, 1.21, 0.84, 0.  ]])
```

Выведите Linkage матрицу и постройте дендрограмму

```
Z = linkage(D)
print(np.array(Z))
_ = dendrogram(Z)
```

```
[[ 0.         17.          0.19        2.        ]
 [ 2.          8.          0.21        2.        ]
 [ 3.          5.          0.22        2.        ]
 [ 4.         10.          0.3         2.        ]
 [14.         22.          0.355       3.        ]
 [ 1.         20.          0.36        3.        ]
```

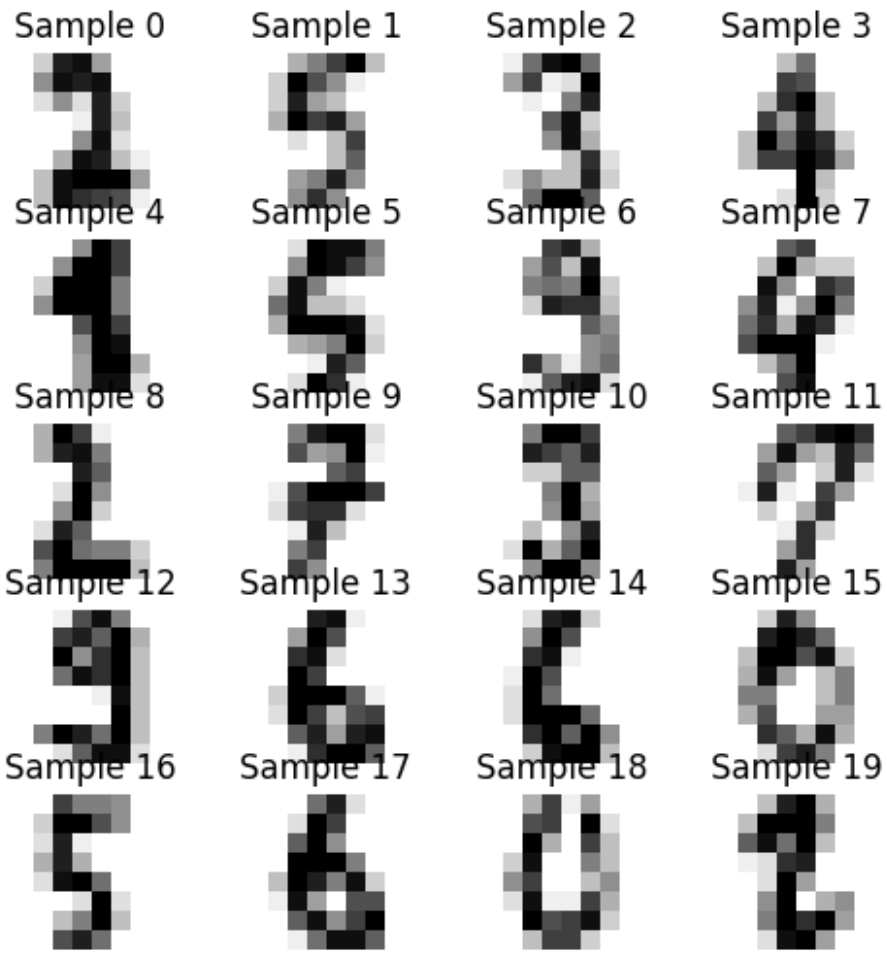
```
[15.         19.         0.42         2.        ]
[ 9.         13.         0.48         2.        ]
[ 6.         18.         0.49         2.        ]
[12.         21.         0.56         3.        ]
[11.         25.         0.56333333   4.        ]
[ 7.         30.         0.6325       5.        ]
[16.         26.         0.705        3.        ]
[28.         32.         0.78166667   5.        ]
[24.         29.         0.80666667   6.        ]
[23.         27.         0.87         4.        ]
[31.         34.         0.995       11.        ]
[33.         36.         1.05890909  16.        ]
[35.         37.         1.184375    20.        ]]
```



2.3. Digits dataset

```
from sklearn import datasets

# загрузим датасет Digits. Он состоит из рукописных изображений цифр
digits = datasets.load_digits().images

# выберем 20 случайных изображений
digits = np.random.permutation(digits)[:20]

# вот они
_, axes = plt.subplots(nrows=5, ncols=4, figsize=(6, 6))
```

```
for i, (ax, image) in enumerate(zip(axes.flatten(), digits)):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title(f'Sample {i}')
```




```
# каждое изображение — матрица размера (8, 8). Давайте вытянем ее в вектор признаков
XX = digits.reshape((digits.shape[0], -1))
```

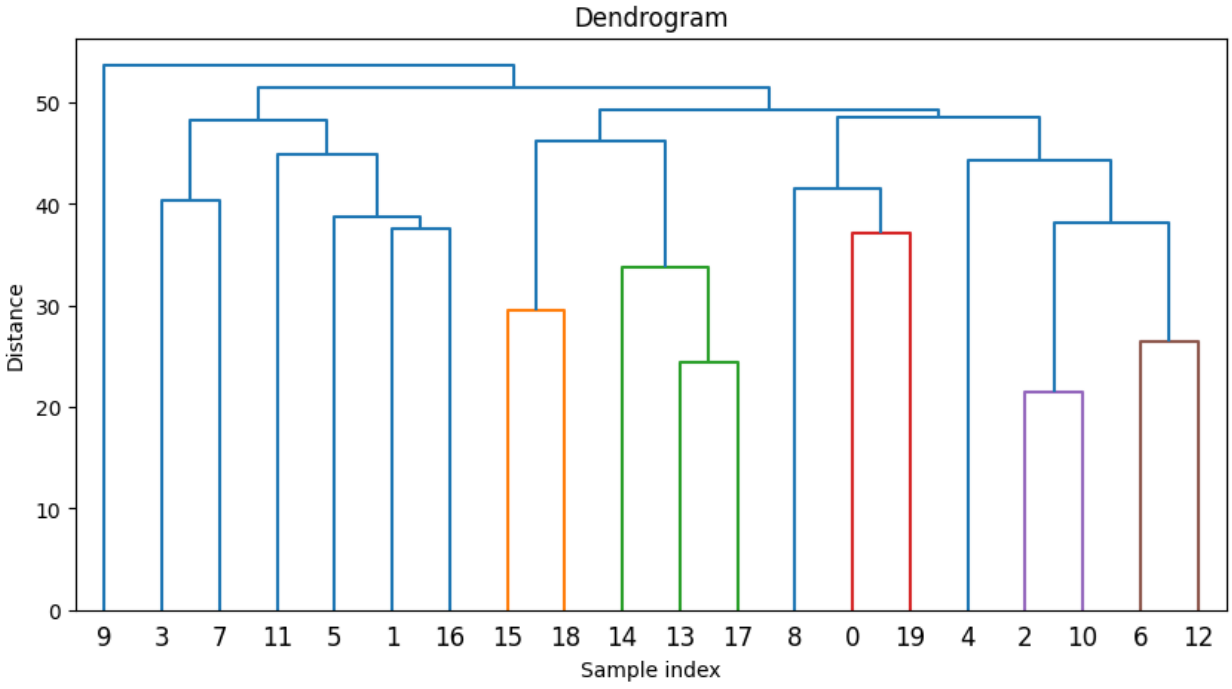
Кластеризуйте вектора и постройте дендрограмму. Какие выводы можно из нее сделать?

```
from scipy.spatial import distance
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram

def distance_matrix(X):
    return distance.cdist(X, X, 'euclidean')
```

```
def linkage(D):
    '''
    :param D: array-like of shape (n_samples, n_samples)
            Matrix of pair-wise distances between samples using
average linkage
    :return linkage: Linkage matrix
    '''
    n = D.shape[0]
    linkage_matrix = []
    clusters = {i: [i] for i in range(n)}
    active_clusters = list(clusters.keys())

    while len(active_clusters) > 1:
        min_dist = np.inf
        closest_pair = (None, None)

        # Перебираем все активные кластеры для поиска пары с
наименьшим средним расстоянием
        for i in range(len(active_clusters)):
            for j in range(i + 1, len(active_clusters)):
                a, b = active_clusters[i], active_clusters[j]
                # Используем среднее расстояние для вычисления
расстояния между кластерами
                if len(clusters[a]) > 0 and len(clusters[b]) > 0:
                    dist_ab = np.mean(D[np.ix_(clusters[a],
clusters[b])])
                    if dist_ab < min_dist:
                        min_dist = dist_ab
                        closest_pair = (a, b)

        ci, cj = closest_pair
        new_cluster = clusters[ci] + clusters[cj]
        new_index = max(clusters.keys()) + 1
        clusters[new_index] = new_cluster

        linkage_matrix.append([ci, cj, min_dist, len(new_cluster)])

        # Обновляем список активных кластеров
        active_clusters = [x for x in active_clusters if x != ci and x
!= cj] + [new_index]

    return np.array(linkage_matrix)

def plot_dendrogram(Z):
    plt.figure(figsize=(10, 5))
    dendrogram(Z)
    plt.title('Dendrogram')
    plt.xlabel('Sample index')
    plt.ylabel('Distance')
    plt.show()
```

```
D = distance_matrix(XX)
Z = linkage(D)
print(np.array(Z))
plot_dendrogram(Z)
```

```
[[ 2.         10.         21.42428529  2.        ]
 [13.         17.         24.35159132  2.        ]
 [ 6.         12.         26.45751311  2.        ]
 [15.         18.         29.52964612  2.        ]
 [14.         21.         33.79003842  3.        ]
 [ 0.         19.         37.06750599  2.        ]
 [ 1.         16.         37.60319135  2.        ]
 [20.         22.         38.09265979  4.        ]
 [ 5.         26.         38.68949464  3.        ]
 [ 3.          7.         40.31128874  2.        ]
 [ 8.         25.         41.46373718  3.        ]
 [ 4.         27.         44.2382525   5.        ]
 [11.         28.         44.82592442  4.        ]
 [23.         24.         46.14205291  5.        ]
 [29.         32.         48.24484832  6.        ]
 [30.         31.         48.53317836  8.        ]
 [33.         35.         49.26851636 13.        ]
 [34.         36.         51.4764269  19.        ]
 [ 9.         37.         53.67073963 20.        ]]
```



Выводы: дендрограмма показывает, как отдельные изображения группируются вместе на разных уровнях расстояния. Группы, которые формируются на более низких уровнях

расстояния, содержат более похожие изображения. Из дендрограммы можно сделать вывод о качестве кластеризации и о том, насколько легко или сложно различать разные цифры в данных. Если две цифры часто путаются, они, скорее всего, окажутся в одном кластере на более раннем этапе кластеризации.