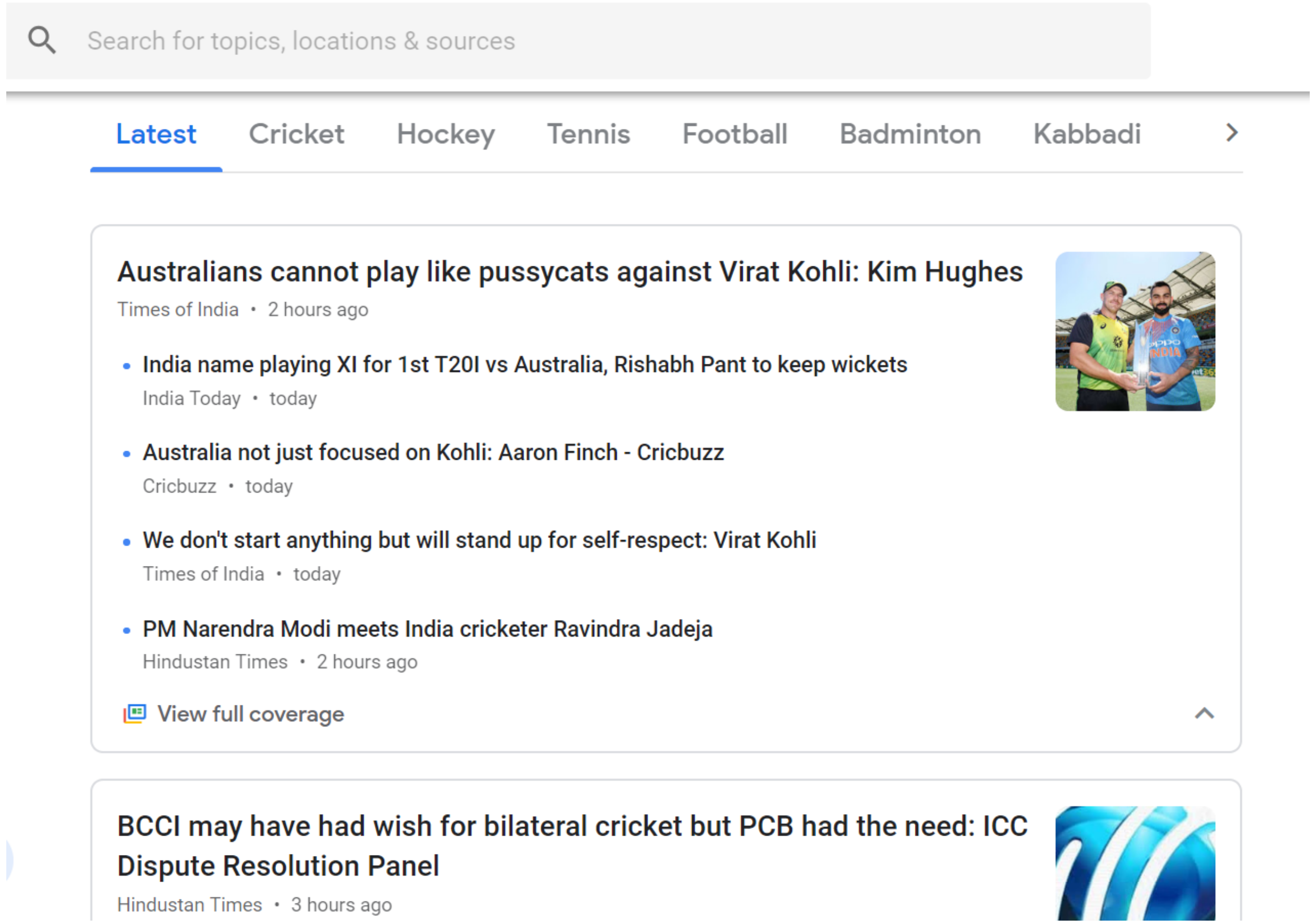
Unsupervised learning: basics

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Everyday example: Google news

- How does Google News classify articles?
- Unsupervised Learning Algorithm: Clustering
- Match frequent terms in articles to find similarity



Labeled and unlabeled data

Data with no labels

- Point 1: (1, 2)
- Point 2: (2, 2)
- Point 3: (3, 1)

Data with labels

- Point 1: (1, 2), Label: Danger Zone
- Point 2: (2, 2), Label: Normal Zone
- Point 3: (3, 1), Label: Normal Zone

What is unsupervised learning?

- A group of machine learning algorithms that find patterns in data
- Data for algorithms has not been labeled, classified or characterized
- The objective of the algorithm is to interpret any structure in the data
- Common unsupervised learning algorithms: clustering, neural networks, anomaly detection

What is clustering?

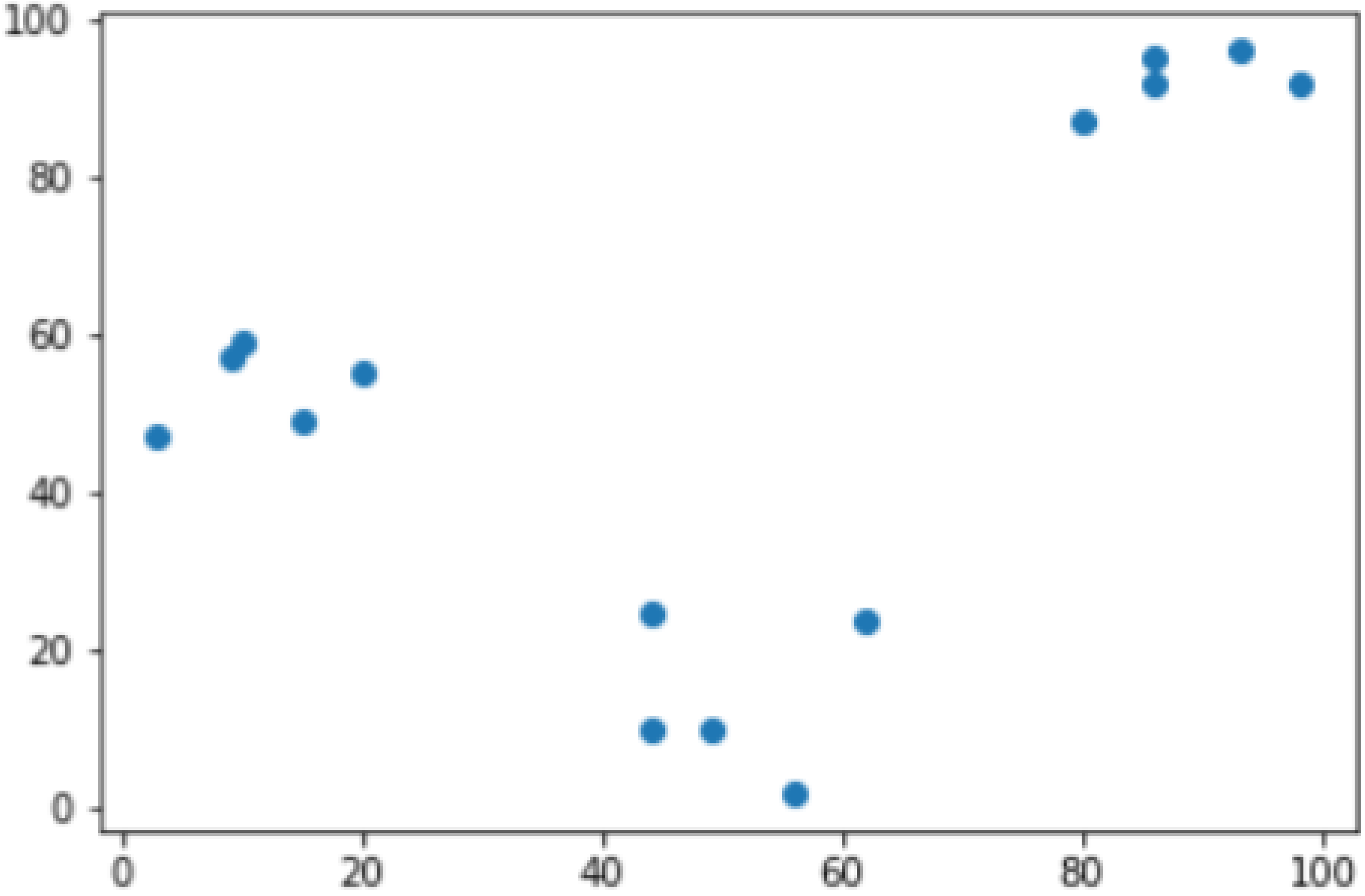
- The process of grouping items with similar characteristics
- Items in groups similar to each other than in other groups
- Example: distance between points on a 2D plane

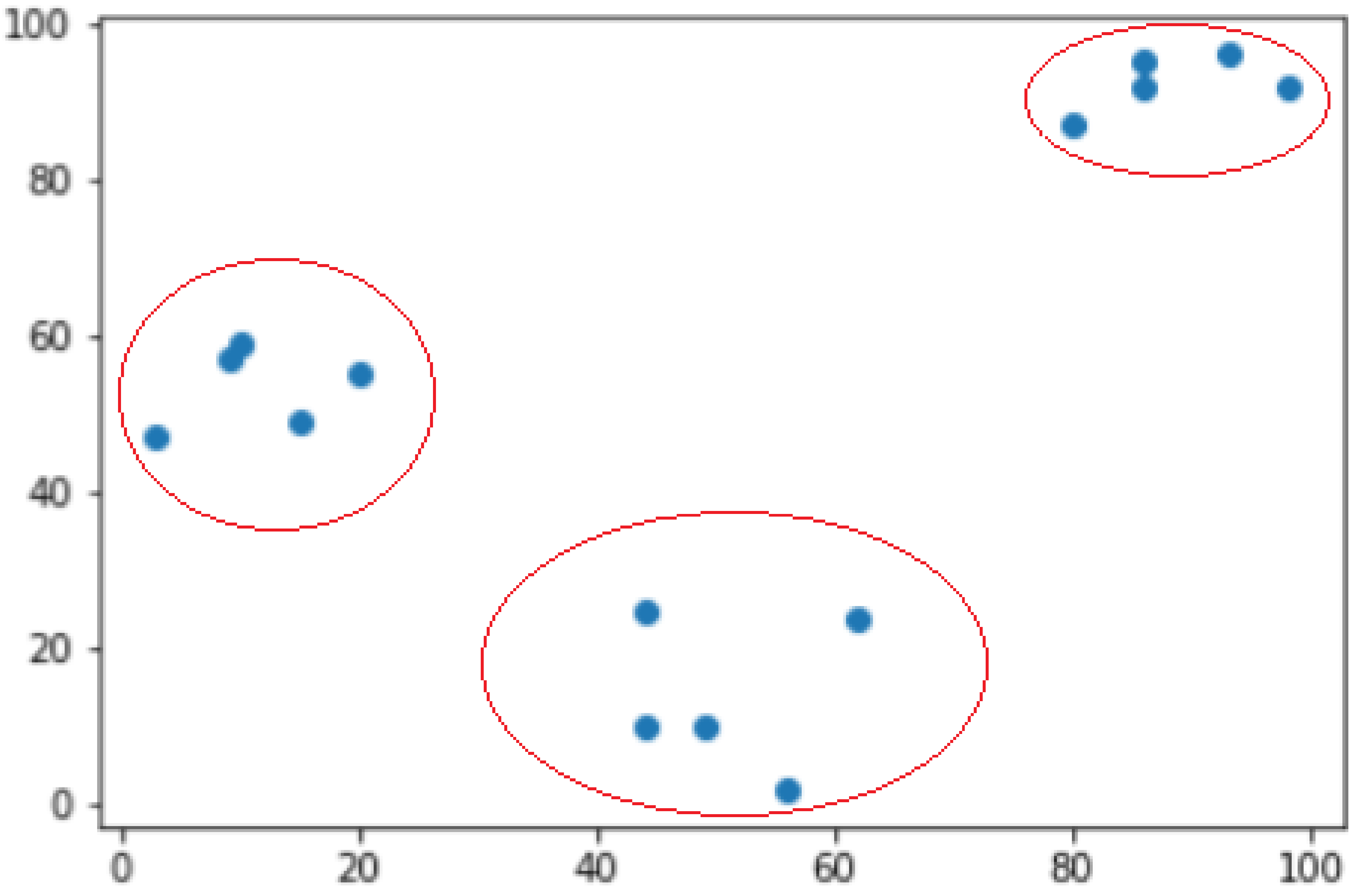
Plotting data for clustering - Pokemon sightings

```
from matplotlib import pyplot as plt
```

```
x_coordinates = [80, 93, 86, 98, 86, 9, 15, 3, 10, 20, 44, 56, 49, 62, 44]
y_coordinates = [87, 96, 95, 92, 92, 57, 49, 47, 59, 55, 25, 2, 10, 24, 10]
```

```
plt.scatter(x_coordinates, y_coordinates)
plt.show()
```


100
80
60
40
20
0
0
20
40
60
80
100


100
80
60
40
20
0
0
20
40
60
80
100

Up next - some practice

CLUSTERING METHODS WITH SCIPY

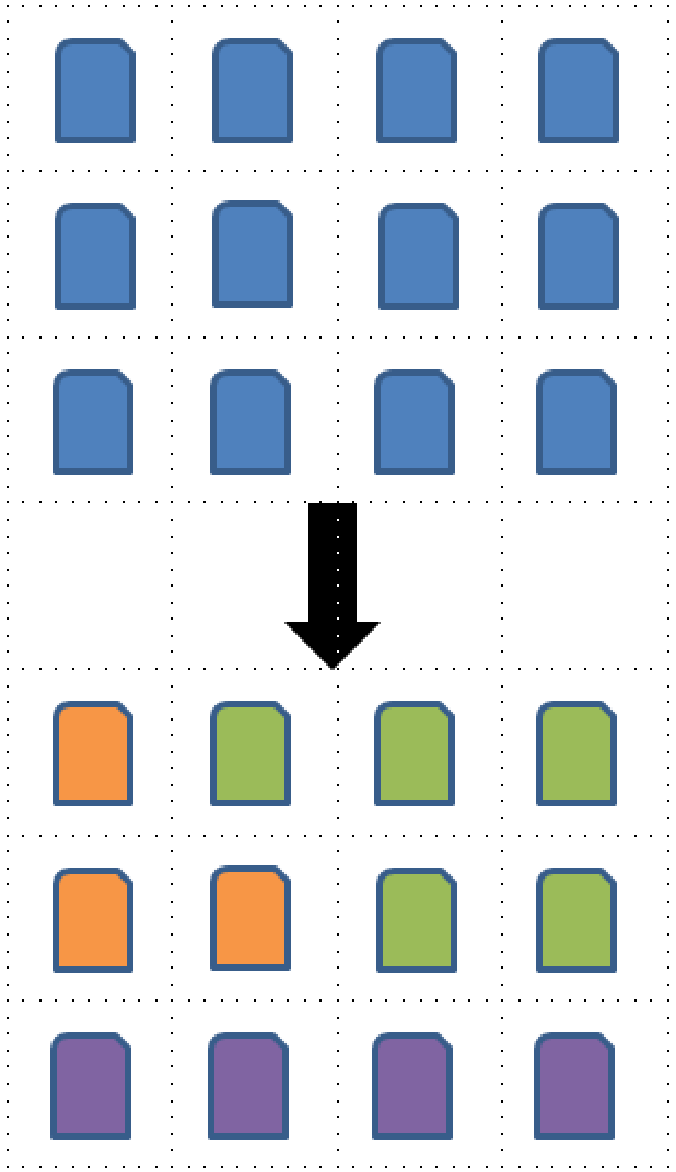
Basics of cluster analysis

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

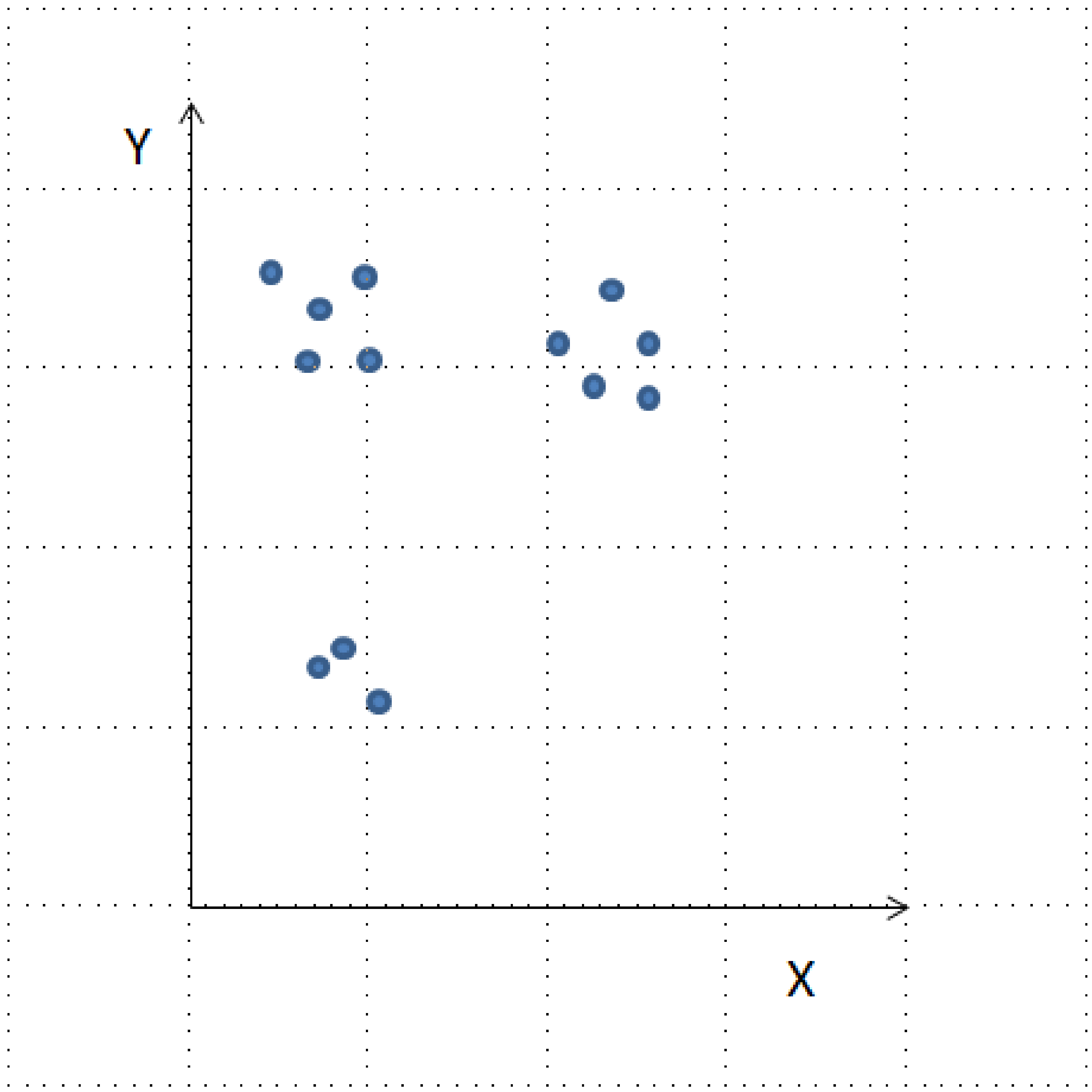
What is a cluster?

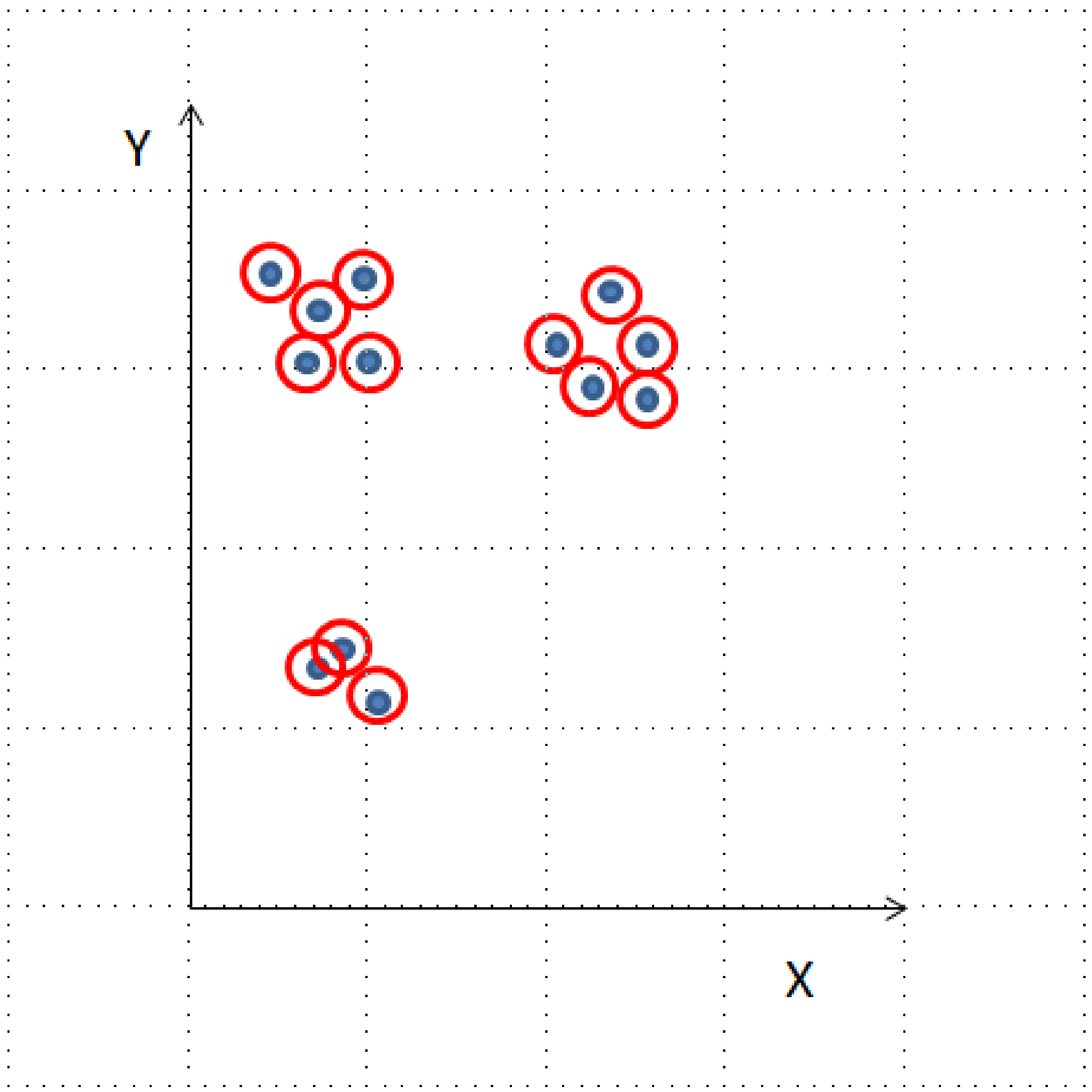
- A group of items with similar characteristics
- Google News: articles where similar words and word associations appear together
- Customer Segments

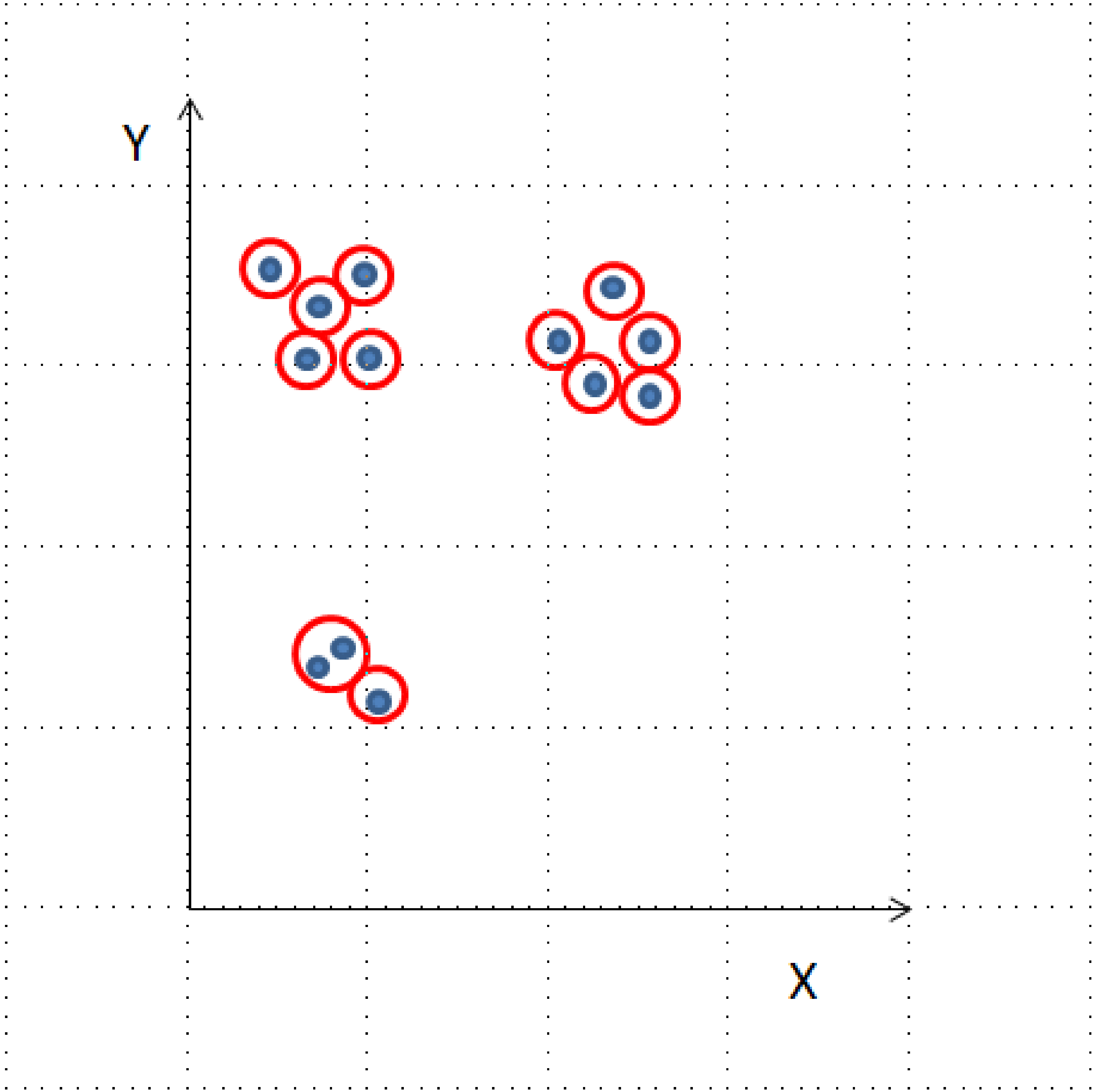


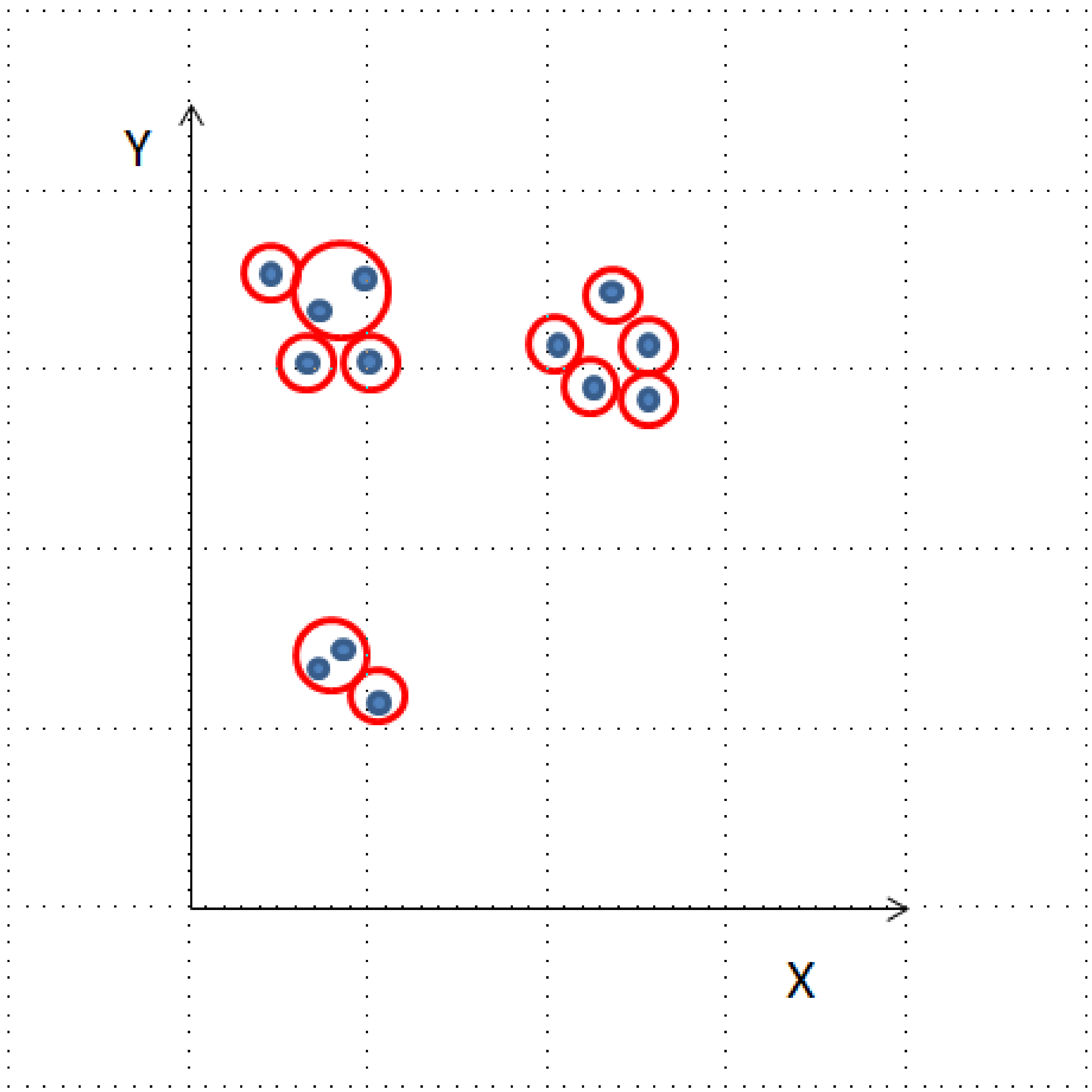
Clustering algorithms

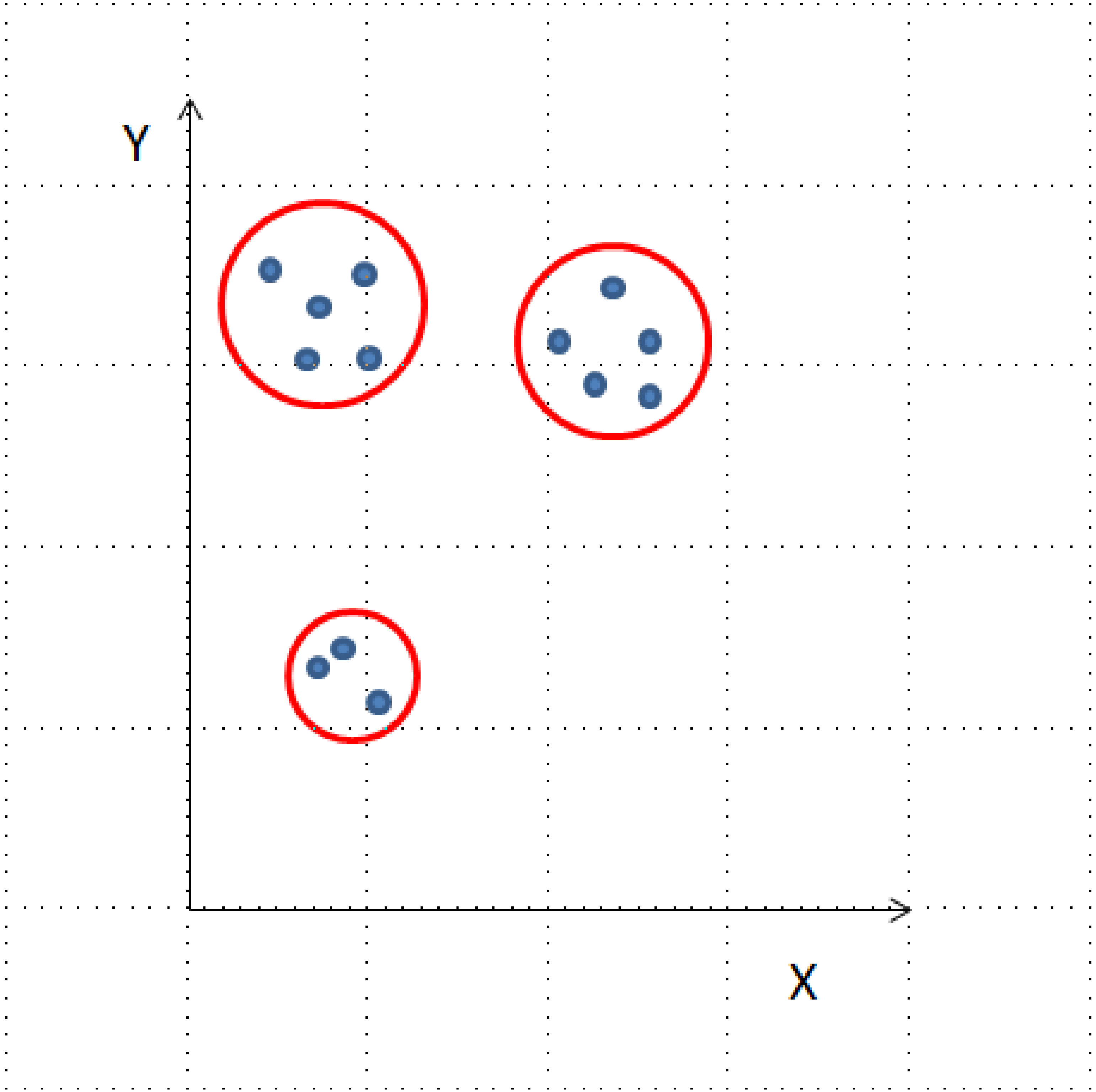
- Hierarchical clustering
- K means clustering
- Other clustering algorithms: DBSCAN, Gaussian Methods


Y
X


Y
X


Y
X


Y
X


Y
X

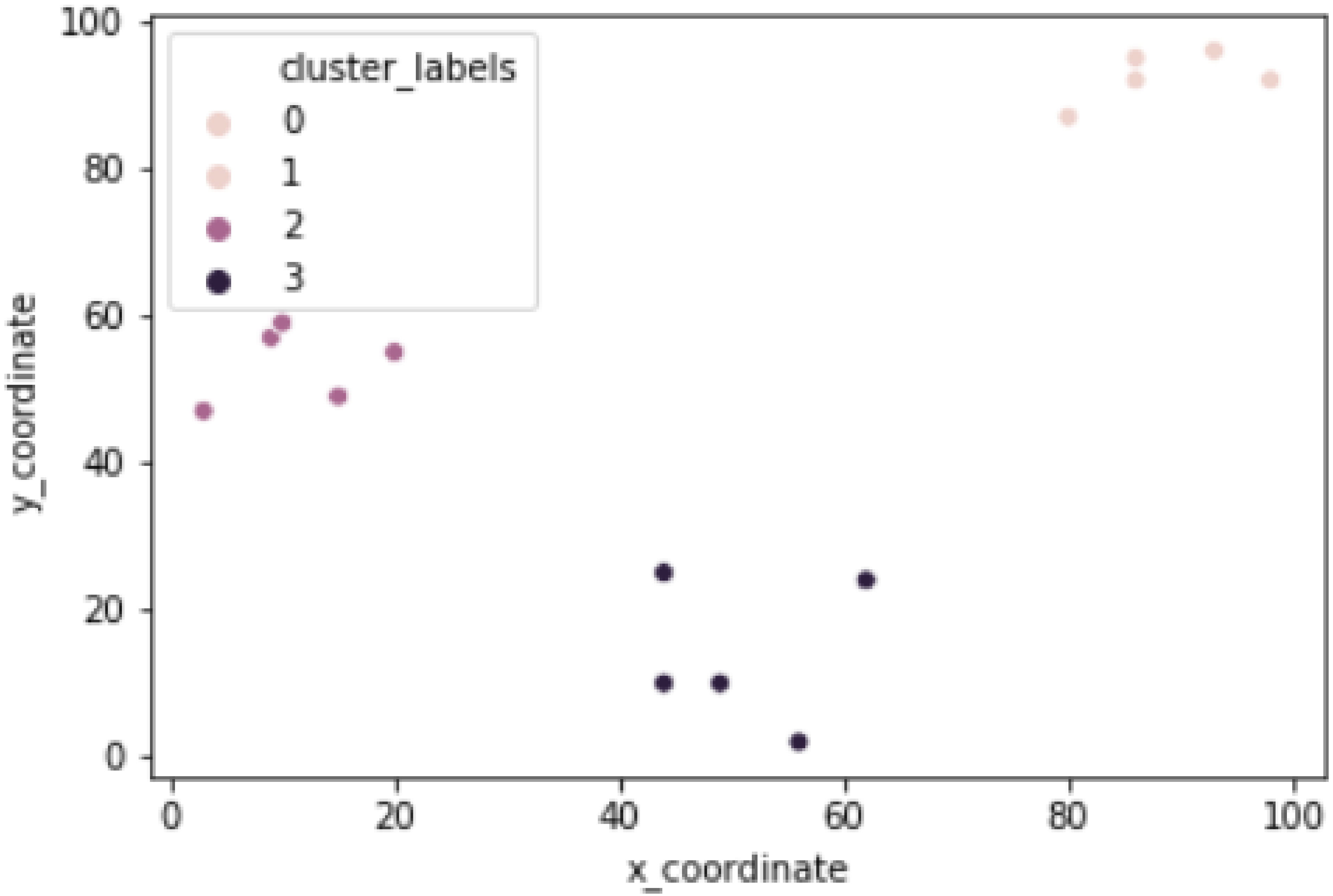
Hierarchical clustering in SciPy

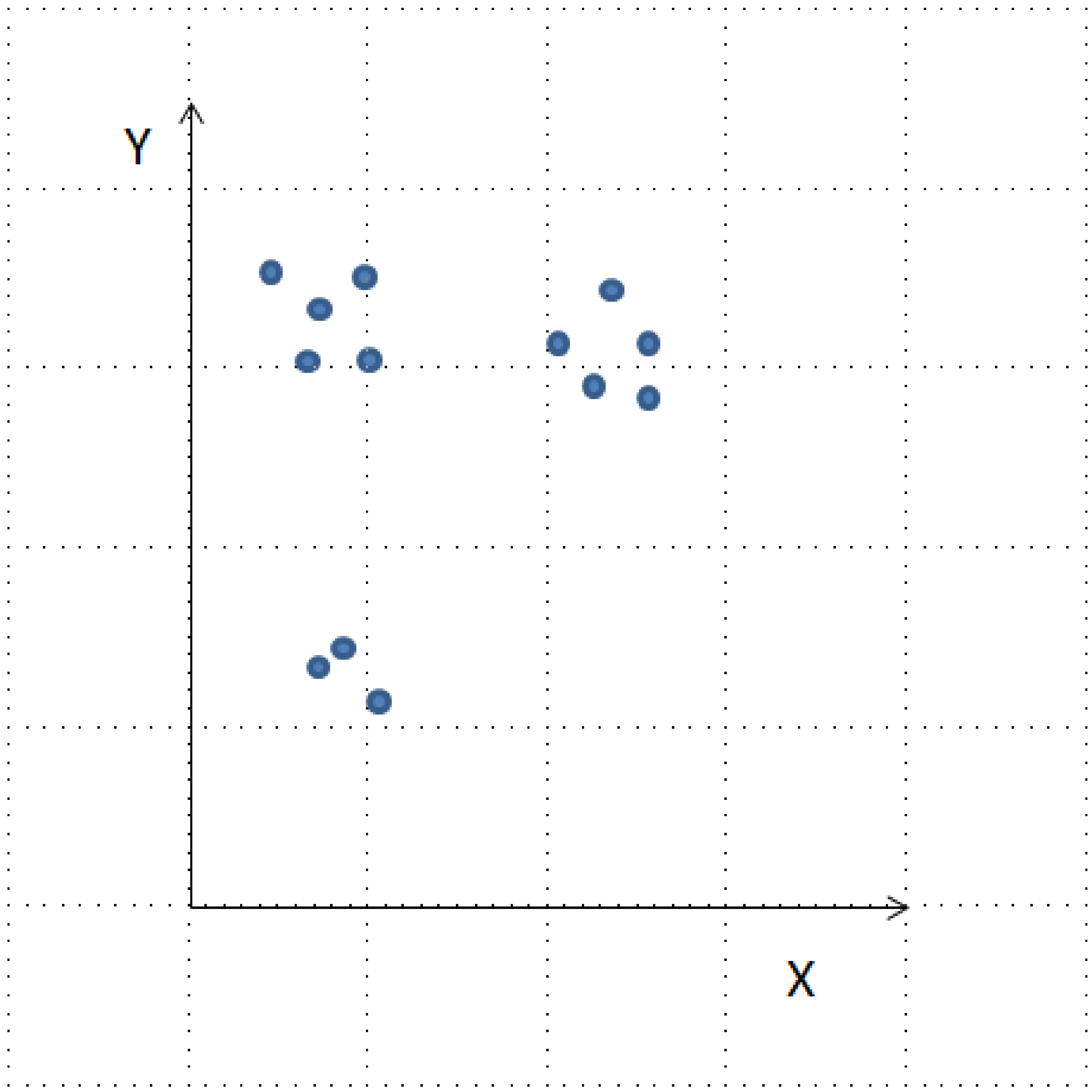
```
from scipy.cluster.hierarchy import linkage, fcluster
from matplotlib import pyplot as plt
import seaborn as sns, pandas as pd
```

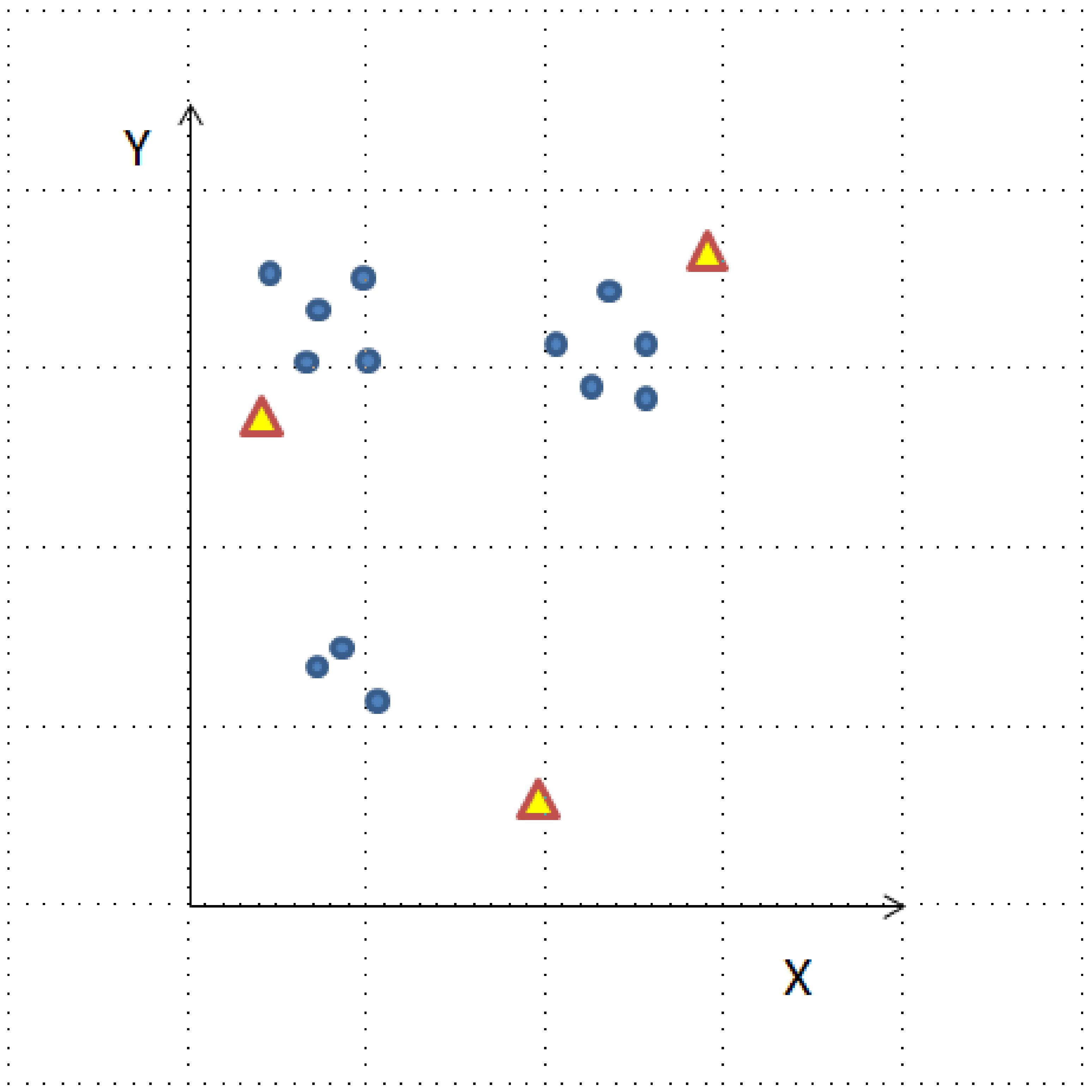
```
x_coordinates = [80.1, 93.1, 86.6, 98.5, 86.4, 9.5, 15.2, 3.4,
                 10.4, 20.3, 44.2, 56.8, 49.2, 62.5, 44.0]
y_coordinates = [87.2, 96.1, 95.6, 92.4, 92.4, 57.7, 49.4,
                 47.3, 59.1, 55.5, 25.6, 2.1, 10.9, 24.1, 10.3]

df = pd.DataFrame({'x_coordinate': x_coordinates,
                   'y_coordinate': y_coordinates})
```

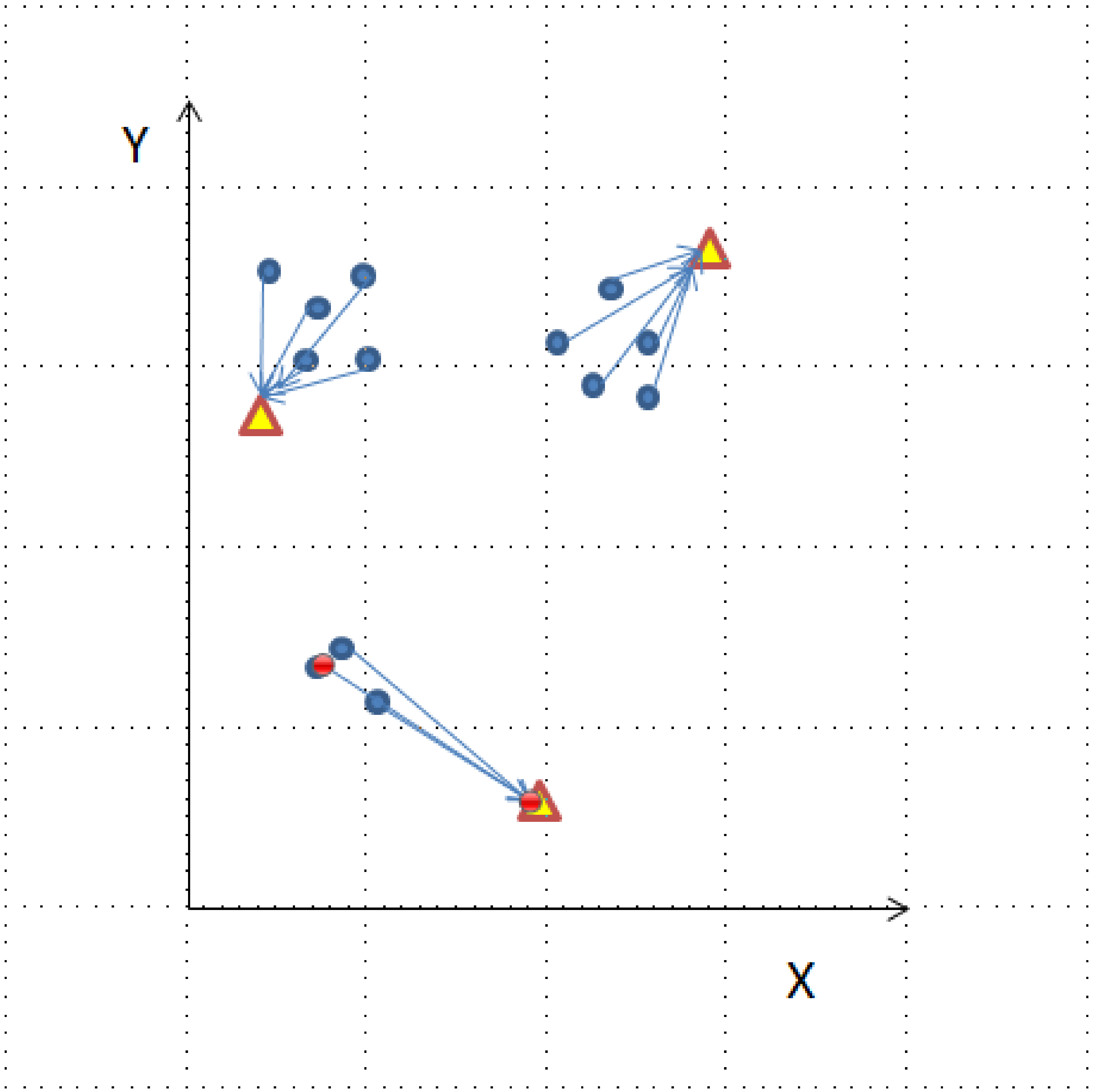
```
Z = linkage(df, 'ward')
df['cluster_labels'] = fcluster(Z, 3, criterion='maxclust')
```

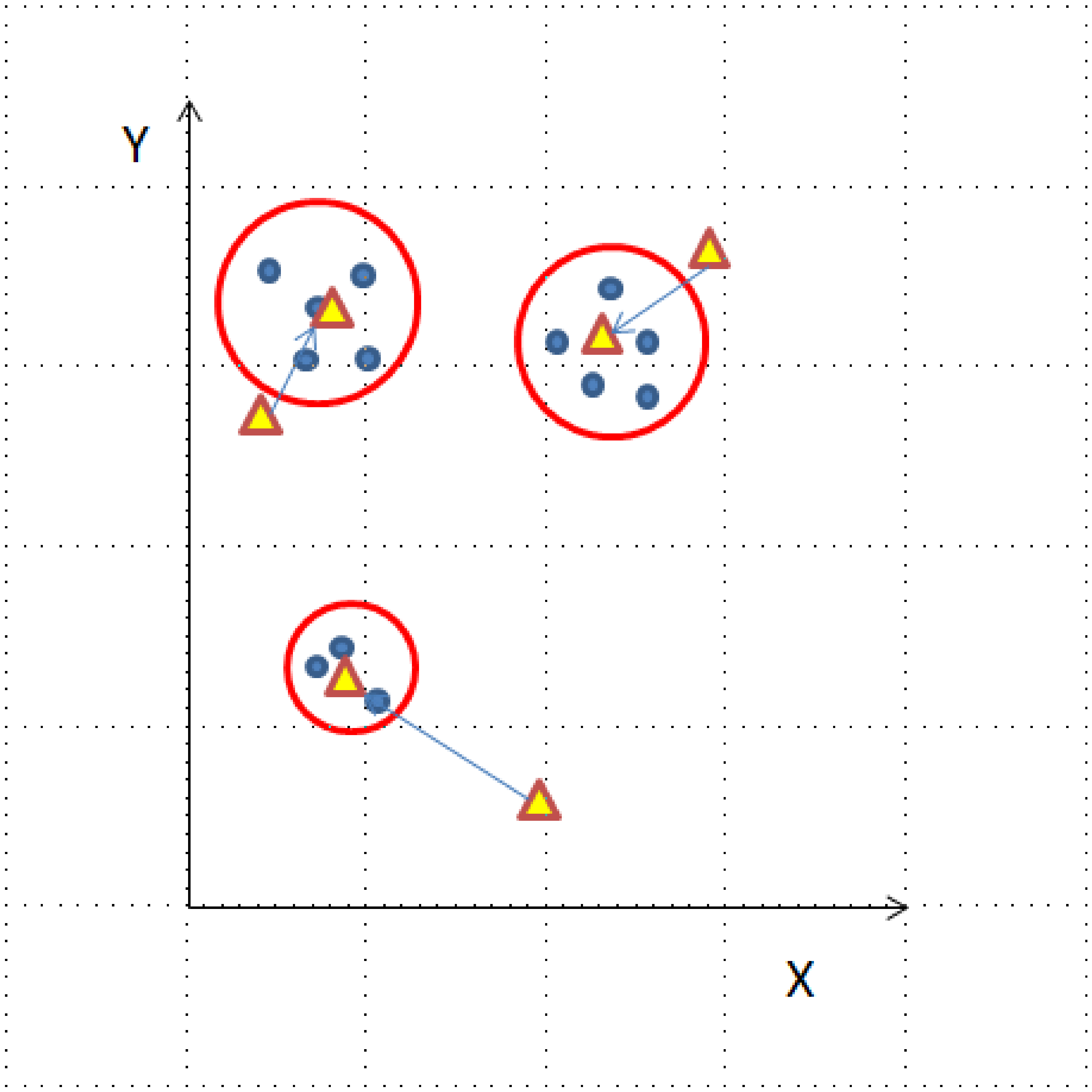
```
sns.scatterplot(x='x_coordinate', y='y_coordinate',
                hue='cluster_labels', data = df)
plt.show()
```


cluster_labels
0
1
2
3
y_coordinate
x_coordinate
100
80
60
40
20
0
0
20
40
60
80
100


Y
X


Y
X


Y
X


Y
X

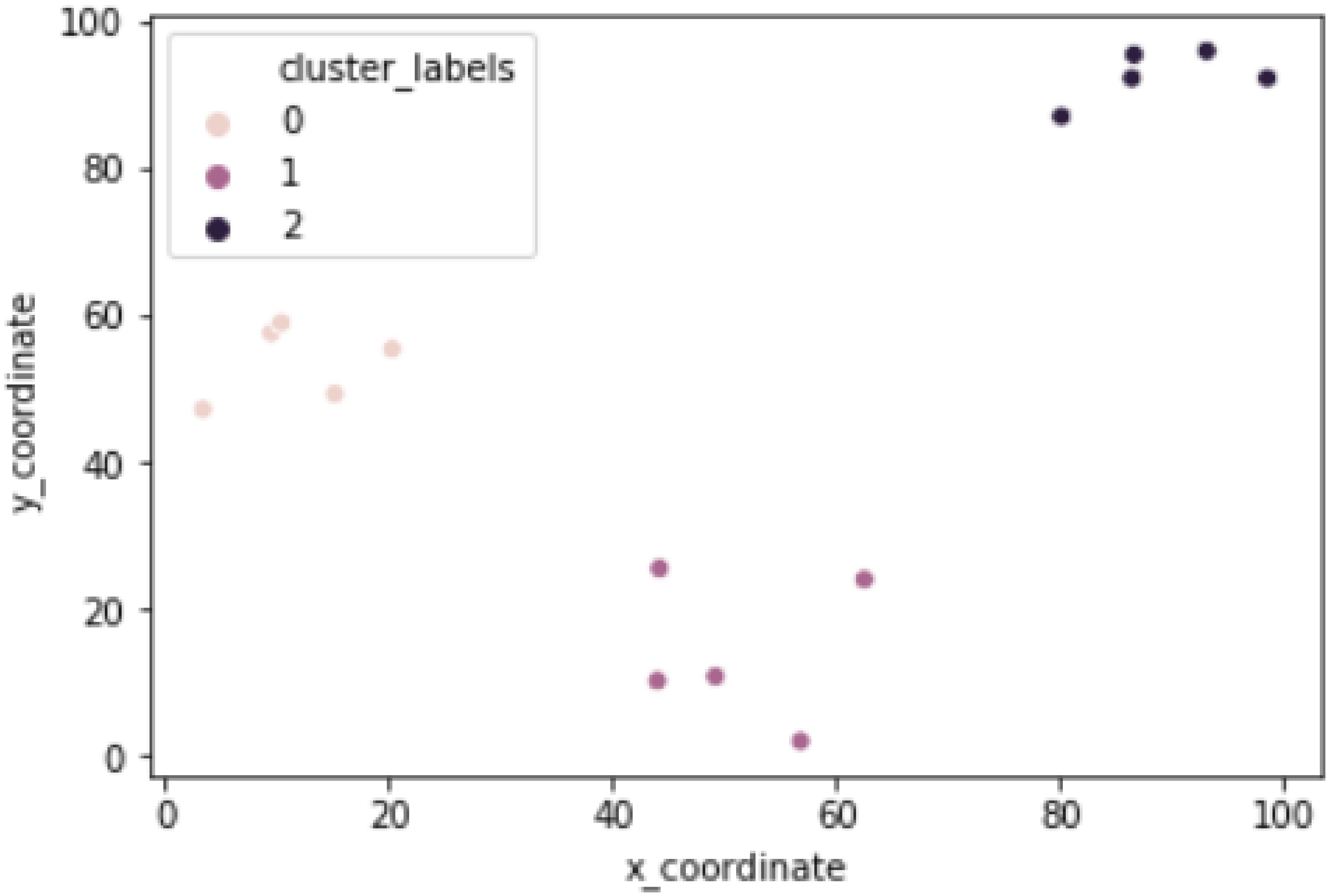
K-means clustering in SciPy

```
from scipy.cluster.vq import kmeans, vq
from matplotlib import pyplot as plt
import seaborn as sns, pandas as pd

import random
random.seed((1000,2000))
```

```
x_coordinates = [80.1, 93.1, 86.6, 98.5, 86.4, 9.5, 15.2, 3.4,
                 10.4, 20.3, 44.2, 56.8, 49.2, 62.5, 44.0]
y_coordinates = [87.2, 96.1, 95.6, 92.4, 92.4, 57.7, 49.4,
                 47.3, 59.1, 55.5, 25.6, 2.1, 10.9, 24.1, 10.3]

df = pd.DataFrame({'x_coordinate': x_coordinates, 'y_coordinate': y_coordinates})
```

```
centroids,_ = kmeans(df, 3)
df['cluster_labels'], _ = vq(df, centroids)
```

```
sns.scatterplot(x='x_coordinate', y='y_coordinate',
                hue='cluster_labels', data = df)
plt.show()
```


cluster_labels
0
1
2
y_coordinate
x_coordinate
0
20
40
60
80
100
0
20
40
60
80
100

Next up: hands-on exercises

CLUSTERING METHODS WITH SCIPY

Data preparation for cluster analysis

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Why do we need to prepare data for clustering?

- Variables have incomparable units (product dimensions in cm, price in $)
- Variables with same units have vastly different scales and variances (expenditures on cereals, travel)
- Data in raw form may lead to bias in clustering
- Clusters may be heavily dependent on one variable
- Solution: normalization of individual variables

Normalization of data

Normalization: process of rescaling data to a standard deviation of 1

> x_new = x / std_dev(x)

```
from scipy.cluster.vq import whiten
```

```
data = [5, 1, 3, 3, 2, 3, 3, 8, 1, 2, 2, 3, 5]
```

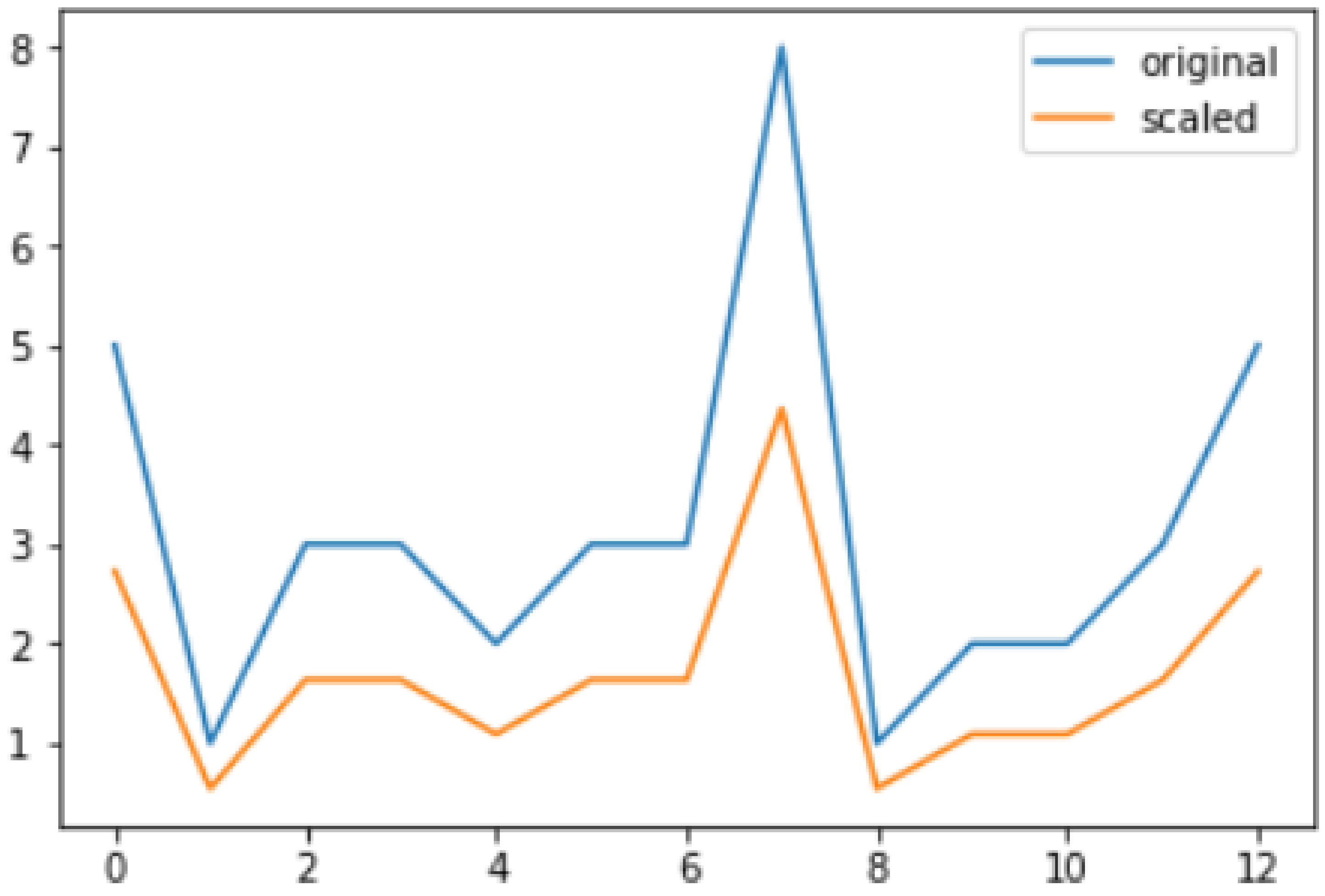
```
scaled_data = whiten(data)
print(scaled_data)
```

```
[2.73, 0.55, 1.64, 1.64, 1.09, 1.64, 1.64, 4.36, 0.55, 1.09, 1.09, 1.64, 2.73]
```

Illustration: normalization of data

```
# Import plotting library
from matplotlib import pyplot as plt

# Initialize original, scaled data
plt.plot(data,
         label="original")
plt.plot(scaled_data,
         label="scaled")
```

```
# Show legend and display plot
plt.legend()
plt.show()
```



Next up: some DIY exercises

CLUSTERING METHODS WITH SCIPY

Basics of hierarchical clustering

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Creating a distance matrix using linkage

```
scipy.cluster.hierarchy.linkage(observations,
                                method='single',
                                metric='euclidean',
                                optimal_ordering=False
)
```

- `method` : how to calculate the proximity of clusters
- `metric` : distance metric
- `optimal_ordering` : order data points

Which method should use?

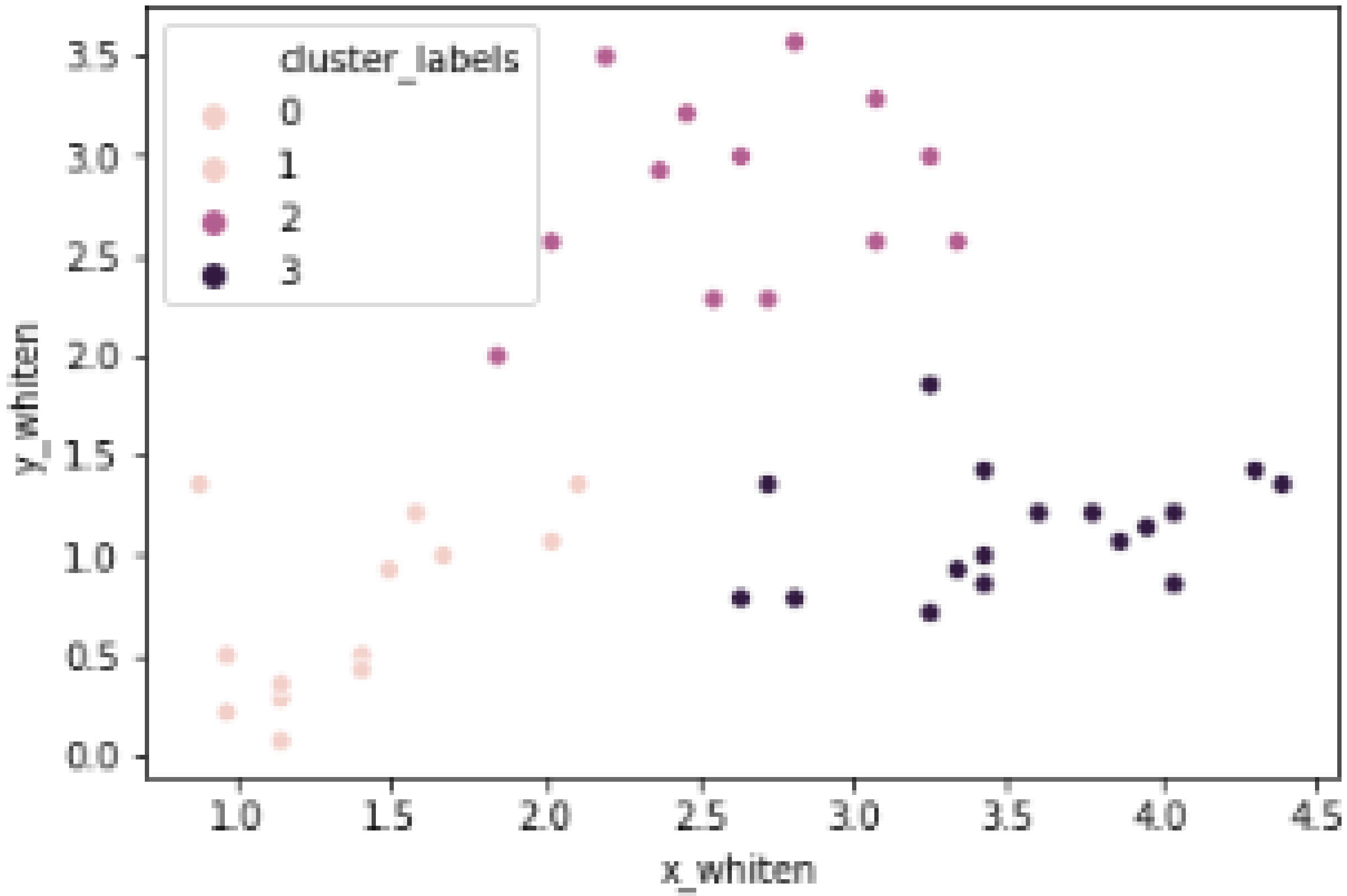
- single: based on two closest objects
- complete: based on two farthest objects
- average: based on the arithmetic mean of all objects
- centroid: based on the geometric mean of all objects
- median: based on the median of all objects
- ward: based on the sum of squares

Create cluster labels with fcluster

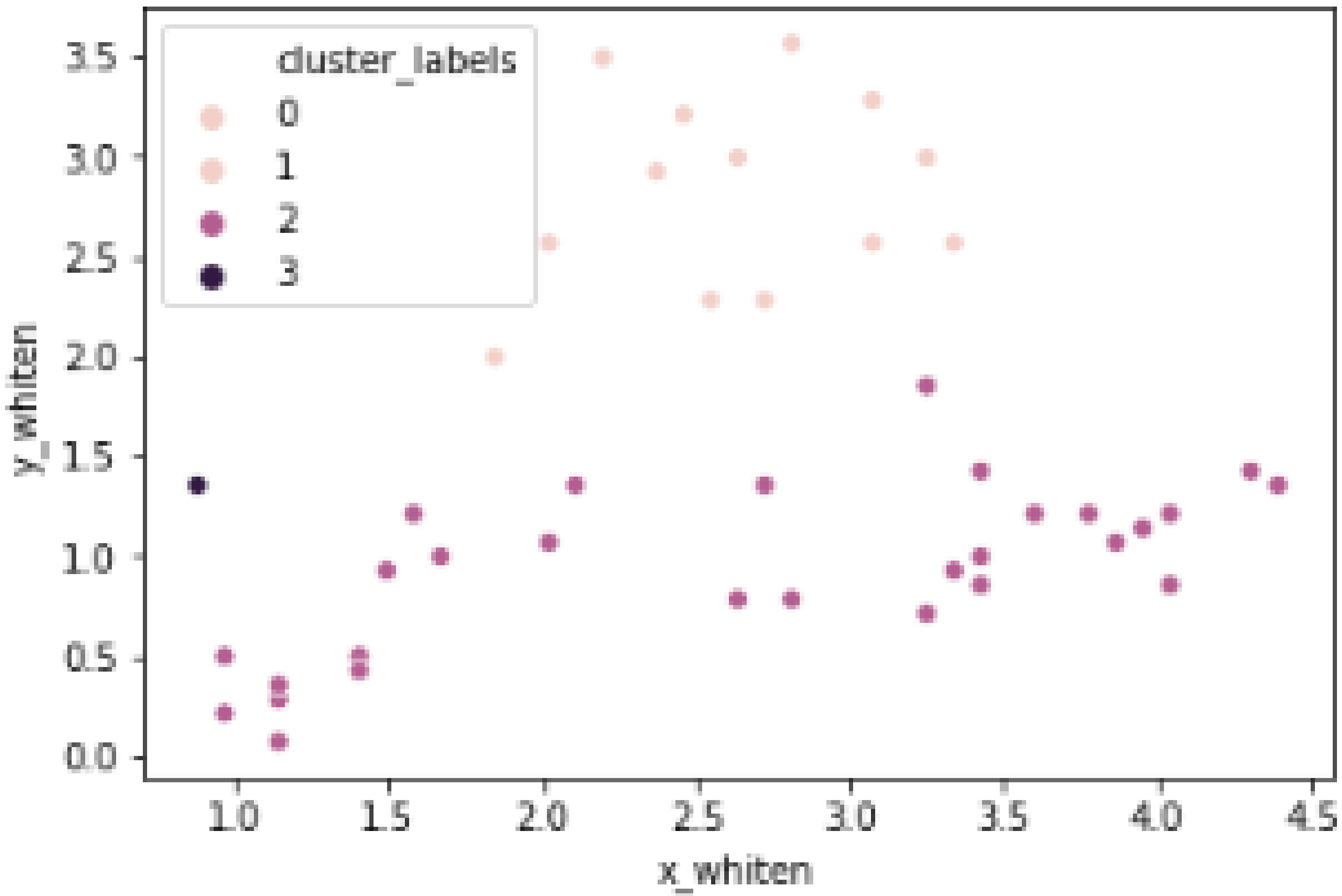
```
scipy.cluster.hierarchy.fcluster(distance_matrix,
                                 num_clusters,
                                 criterion
)
```

- `distance_matrix` : output of `linkage()` method
- `num_clusters` : number of clusters
- `criterion` : how to decide thresholds to form clusters

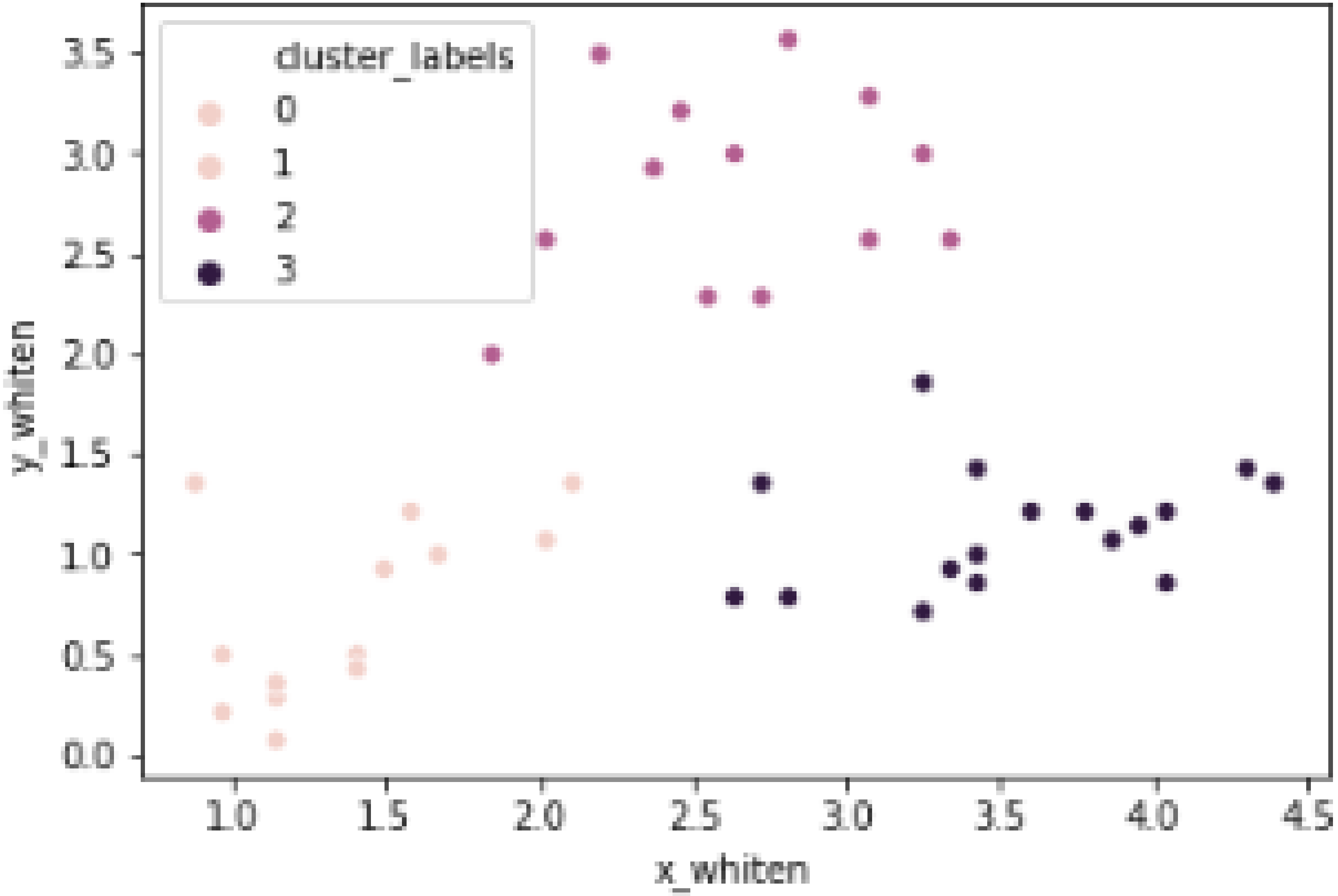
Hierarchical clustering with ward method



Hierarchical clustering with single method



Hierarchical clustering with complete method



Final thoughts on selecting a method

- No one right method for all
- Need to carefully understand the distribution of data

Let's try some exercises

CLUSTERING METHODS WITH SCIPY

Visualize clusters

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Why visualize clusters?

- Try to make sense of the clusters formed
- An additional step in validation of clusters
- Spot trends in data

An introduction to seaborn

- `seaborn` : a Python data visualization library based on `matplotlib`
- Has better, easily modifiable aesthetics than matplotlib!
- Contains functions that make data visualization tasks easy in the context of data analytics
- Use case for clustering: `hue` parameter for plots

Visualize clusters with matplotlib

```
from matplotlib import pyplot as plt
```

```
df = pd.DataFrame({'x': [2, 3, 5, 6, 2],
                   'y': [1, 1, 5, 5, 2],
                   'labels': ['A', 'A', 'B', 'B', 'A']})

colors = {'A':'red', 'B':'blue'}

df.plot.scatter(x='x',
                y='y',
                c=df['labels'].apply(lambda x: colors[x]))
plt.show()
```

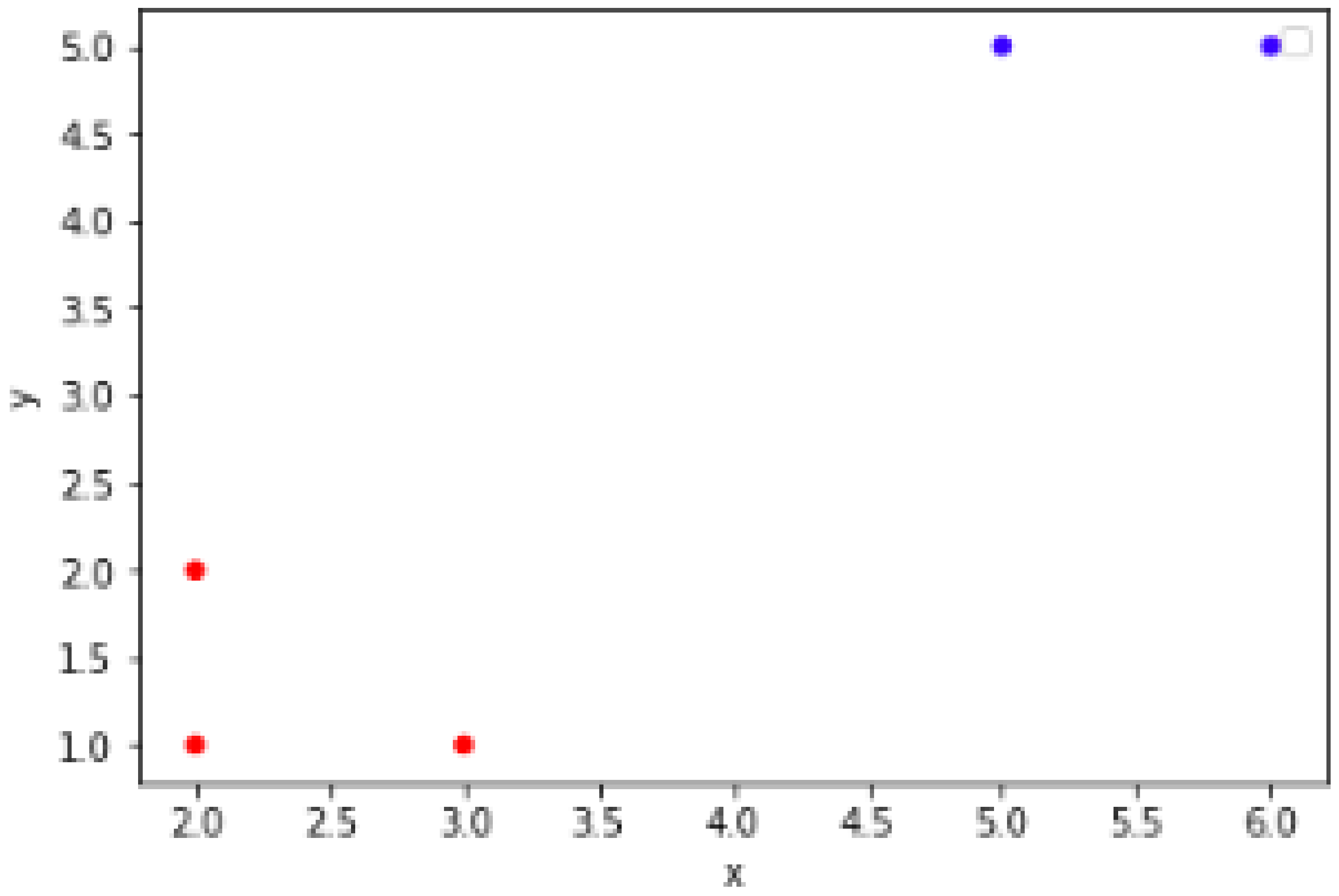
Visualize clusters with seaborn

```
from matplotlib import pyplot as plt
import seaborn as sns
```

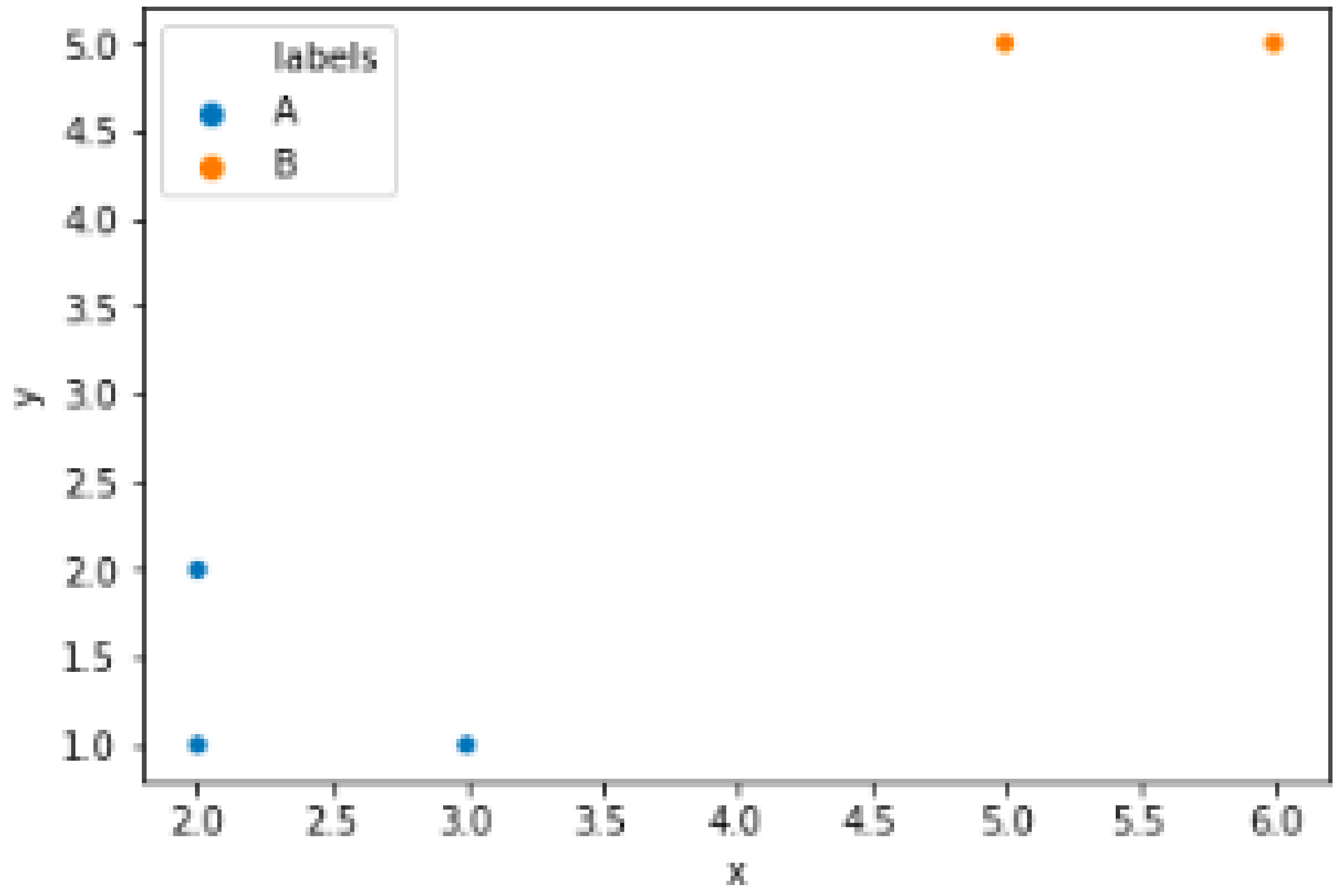
```
df = pd.DataFrame({'x': [2, 3, 5, 6, 2],
                   'y': [1, 1, 5, 5, 2],
                   'labels': ['A', 'A', 'B', 'B', 'A']})

sns.scatterplot(x='x',
                y='y',
                hue='labels',
                data=df)
plt.show()
```

Comparison of both methods of visualization

MATPLOTLIB PLOT



SEABORN PLOT



Next up: Try some visualizations

CLUSTERING METHODS WITH SCIPY

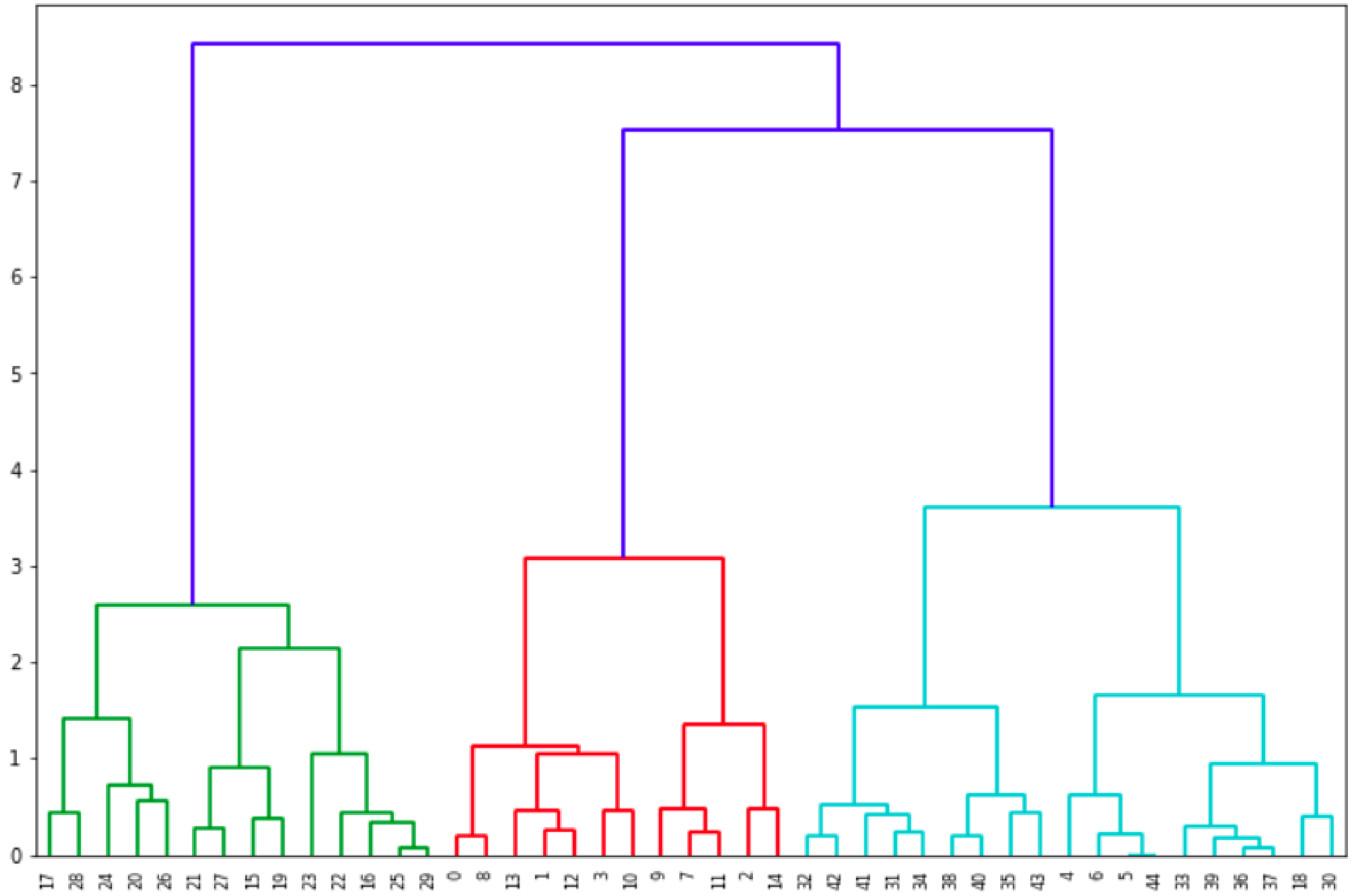
How many clusters?

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Introduction to dendrograms

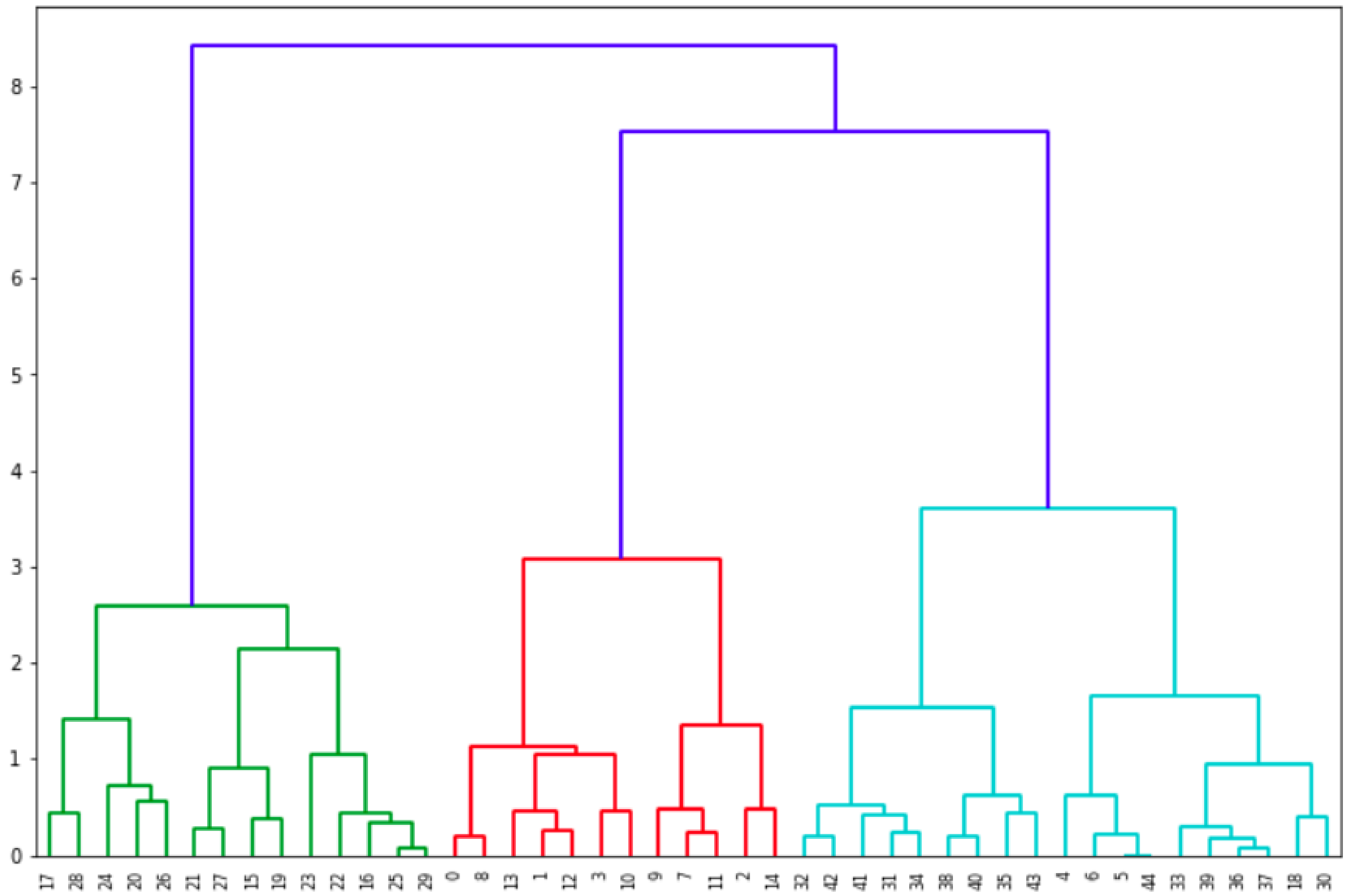
- Strategy till now - decide clusters on visual inspection
- Dendrograms help in showing progressions as clusters are merged
- A dendrogram is a branching diagram that demonstrates how each cluster is composed by branching out into its child nodes

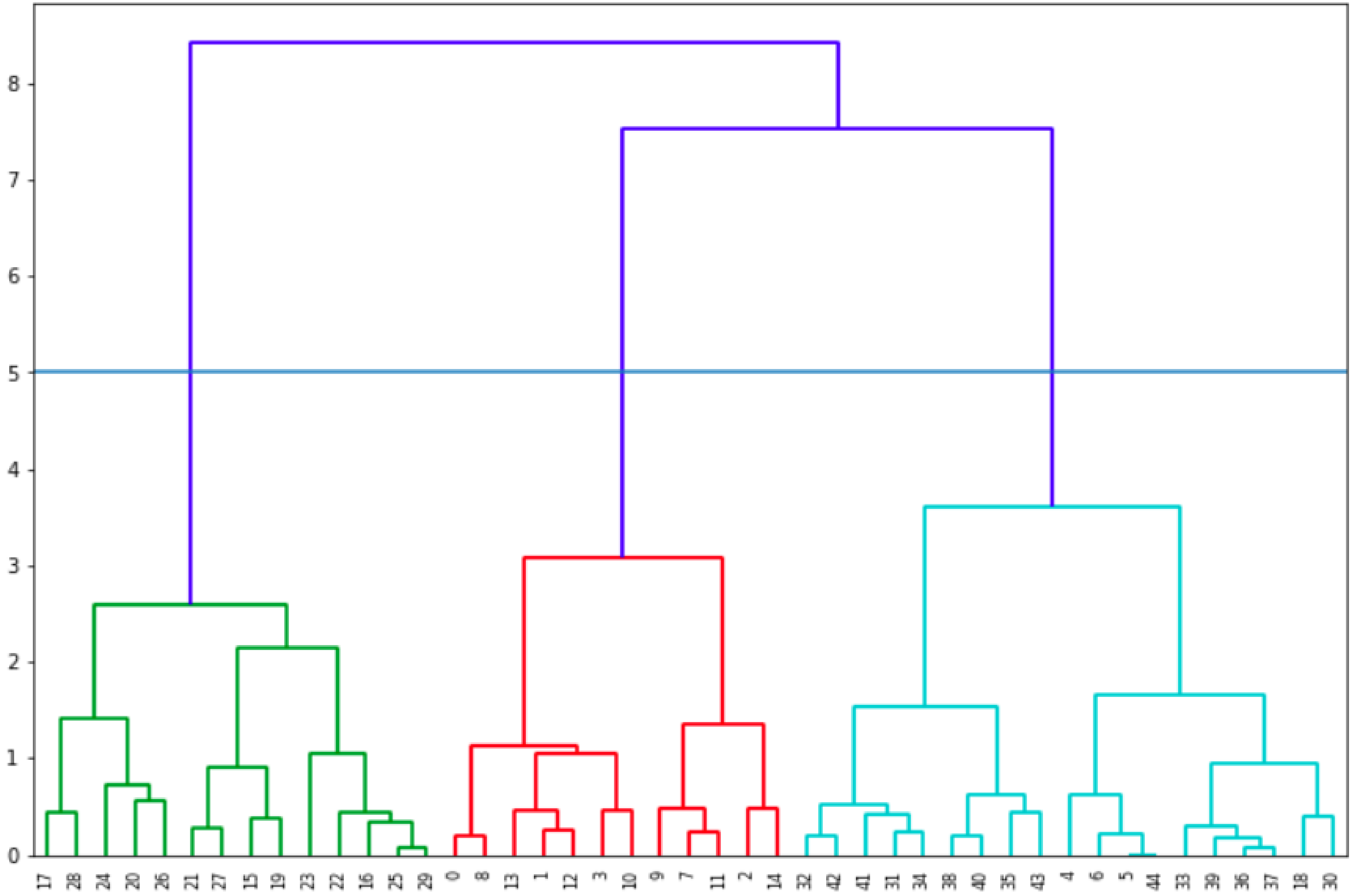


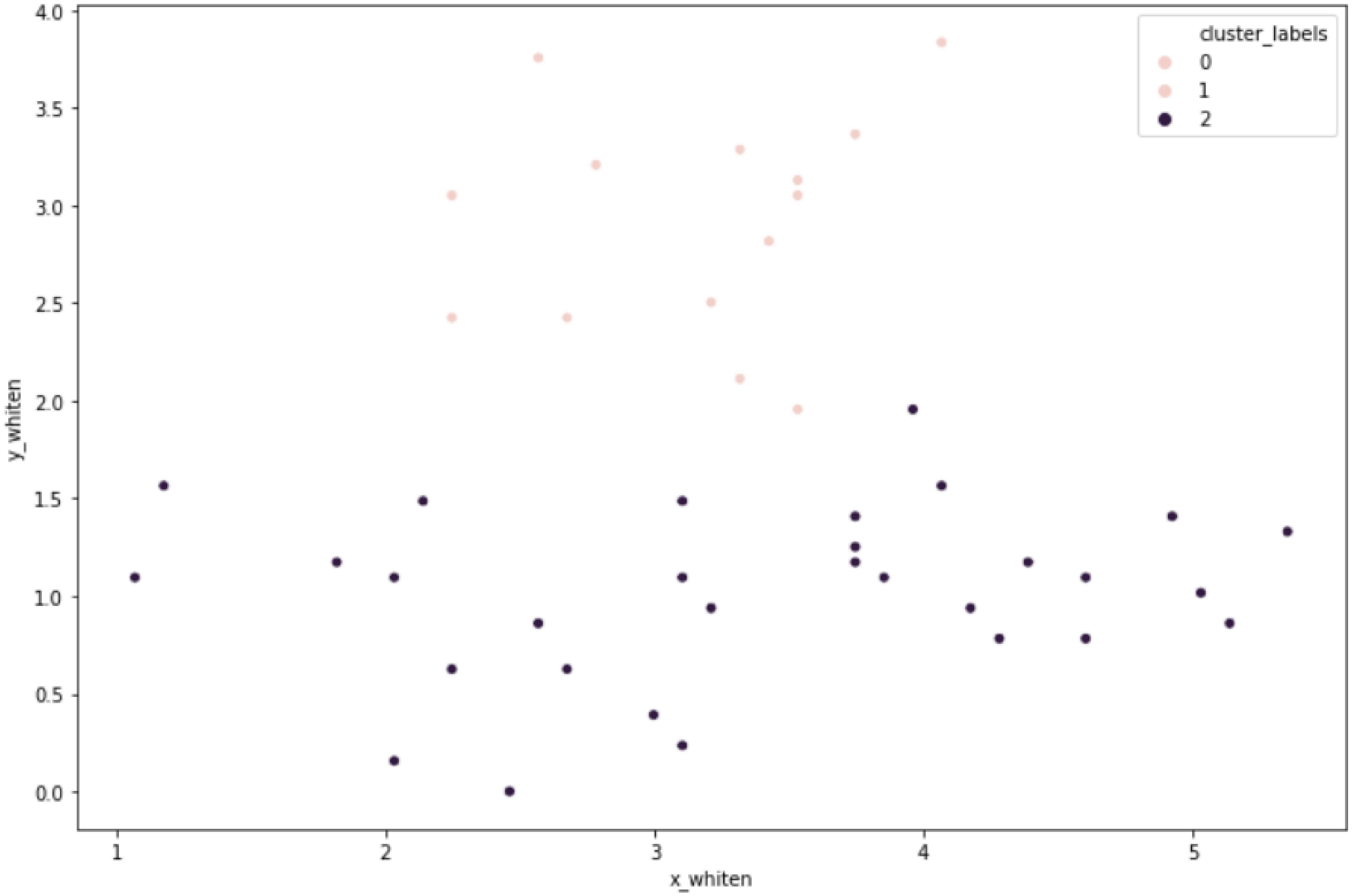
Create a dendrogram in SciPy

```
from scipy.cluster.hierarchy import dendrogram
```

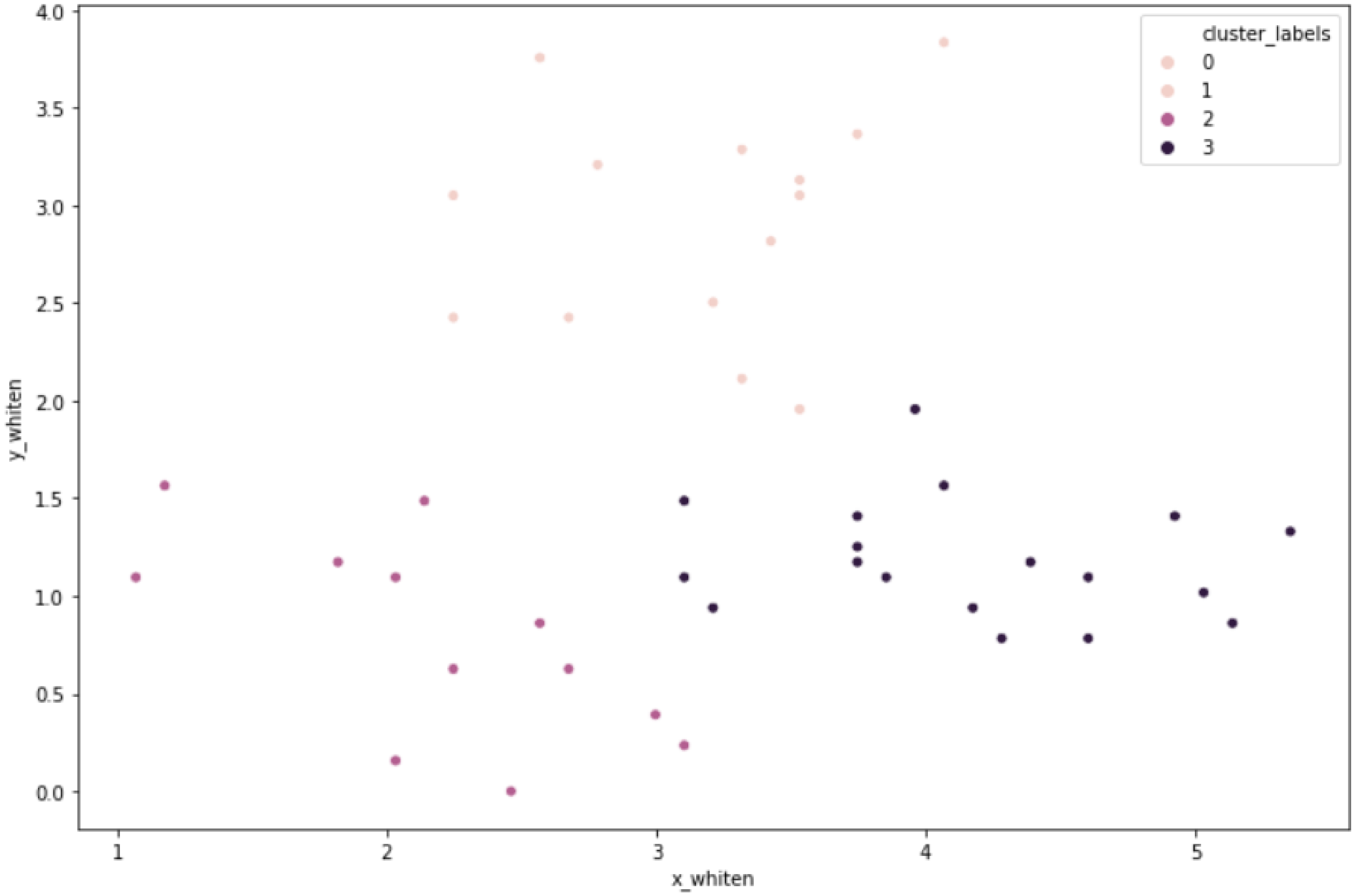
```
Z = linkage(df[['x_whiten', 'y_whiten']],
            method='ward',
            metric='euclidean')

dn = dendrogram(Z)
plt.show()
```

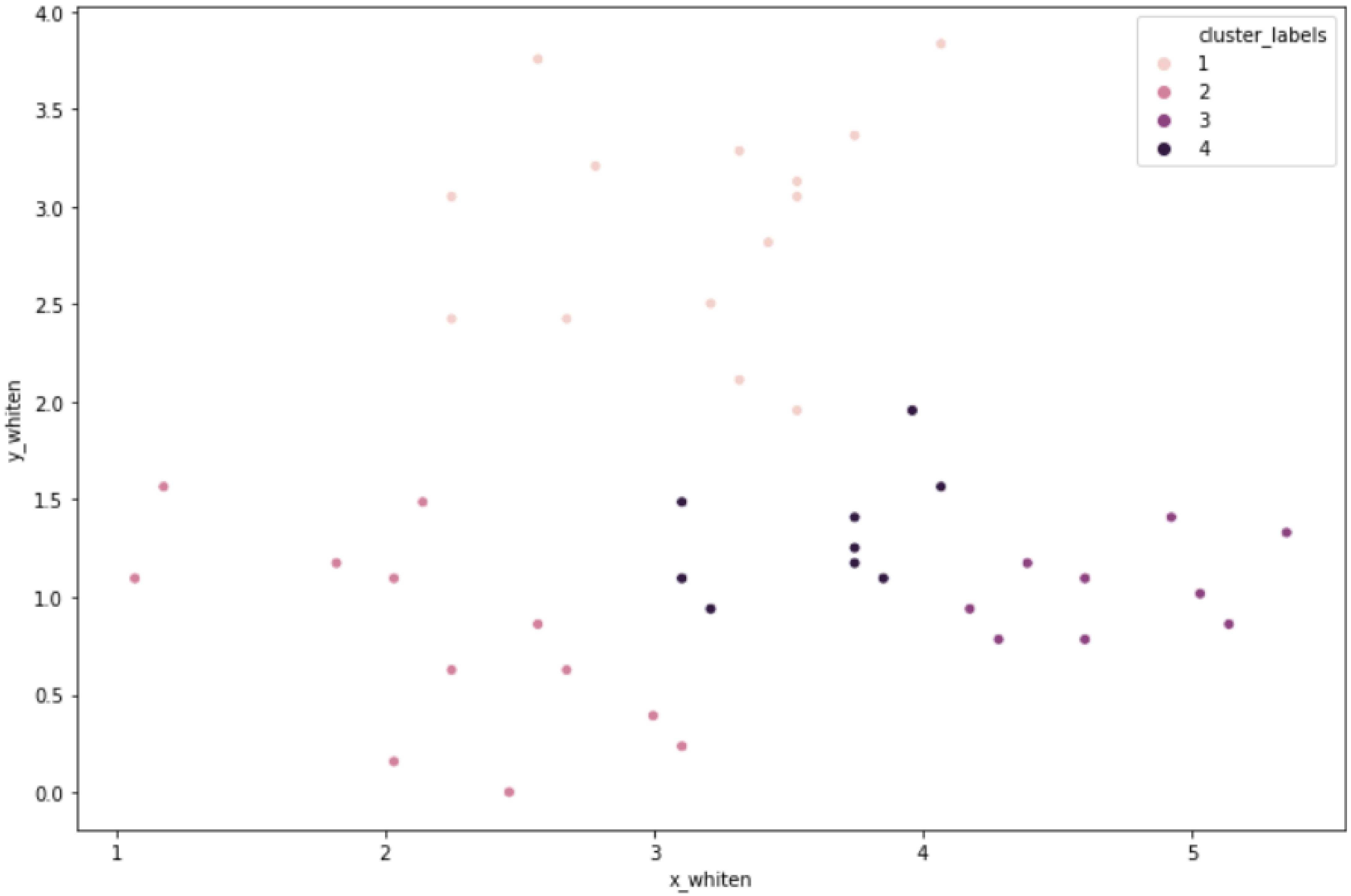

8
7
6
5
4
3
2
1
0
17 28 24 20 26 21 27 15 19 23 22 16 25 29 0 8 13 1 12 3 10 9 7 11 2 14 32 42 41 31 34 38 40 35 43 4 6 5 44 33 39 36 37 18 30


8
7
6
5
4
3
2
1
0
17 28 24 20 26 21 27 15 19 23 22 16 25 29 0 8 13 1 12 3 10 9 7 11 2 14 32 42 41 31 34 38 40 35 43 4 6 5 44 33 39 36 37 18 30



cluster_labels
0
1
2
y_whiten
x_whiten
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5


cluster_labels
0
1
2
3
y_whiten
x_whiten
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5


cluster_labels
1
2
3
4
y_whiten
x_whiten
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5

Next up - try some exercises

CLUSTERING METHODS WITH SCIPY

Limitations of hierarchical clustering

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Measuring speed in hierarchical clustering

- `timeit` module
- Measure the speed of `.linkage()` method
- Use randomly generated points
- Run various iterations to extrapolate

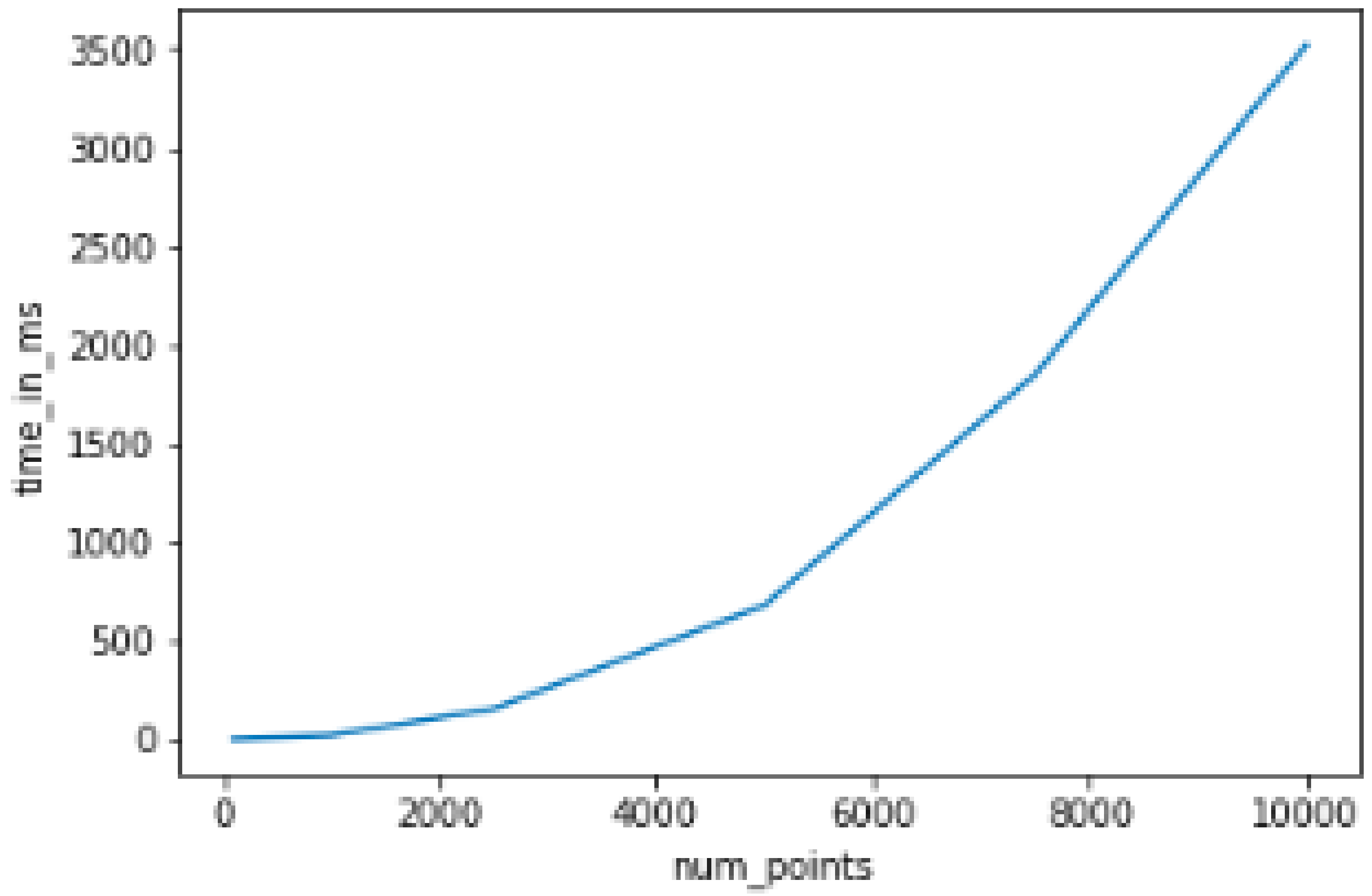
Use of timeit module

```
from scipy.cluster.hierarchy import linkage
import pandas as pd
import random, timeit

points = 100
df = pd.DataFrame({'x': random.sample(range(0, points), points),
                   'y': random.sample(range(0, points), points)})

%timeit linkage(df[['x', 'y']], method = 'ward', metric = 'euclidean')
```

```
1.02 ms ± 133 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Comparison of runtime of linkage method

- Increasing runtime with data points
- Quadratic increase of runtime
- Not feasible for large datasets



Next up - exercises

CLUSTERING METHODS WITH SCIPY

Basics of k-means clustering

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Why k-means clustering?

- A critical drawback of hierarchical clustering: runtime
- K means runs significantly faster on large datasets

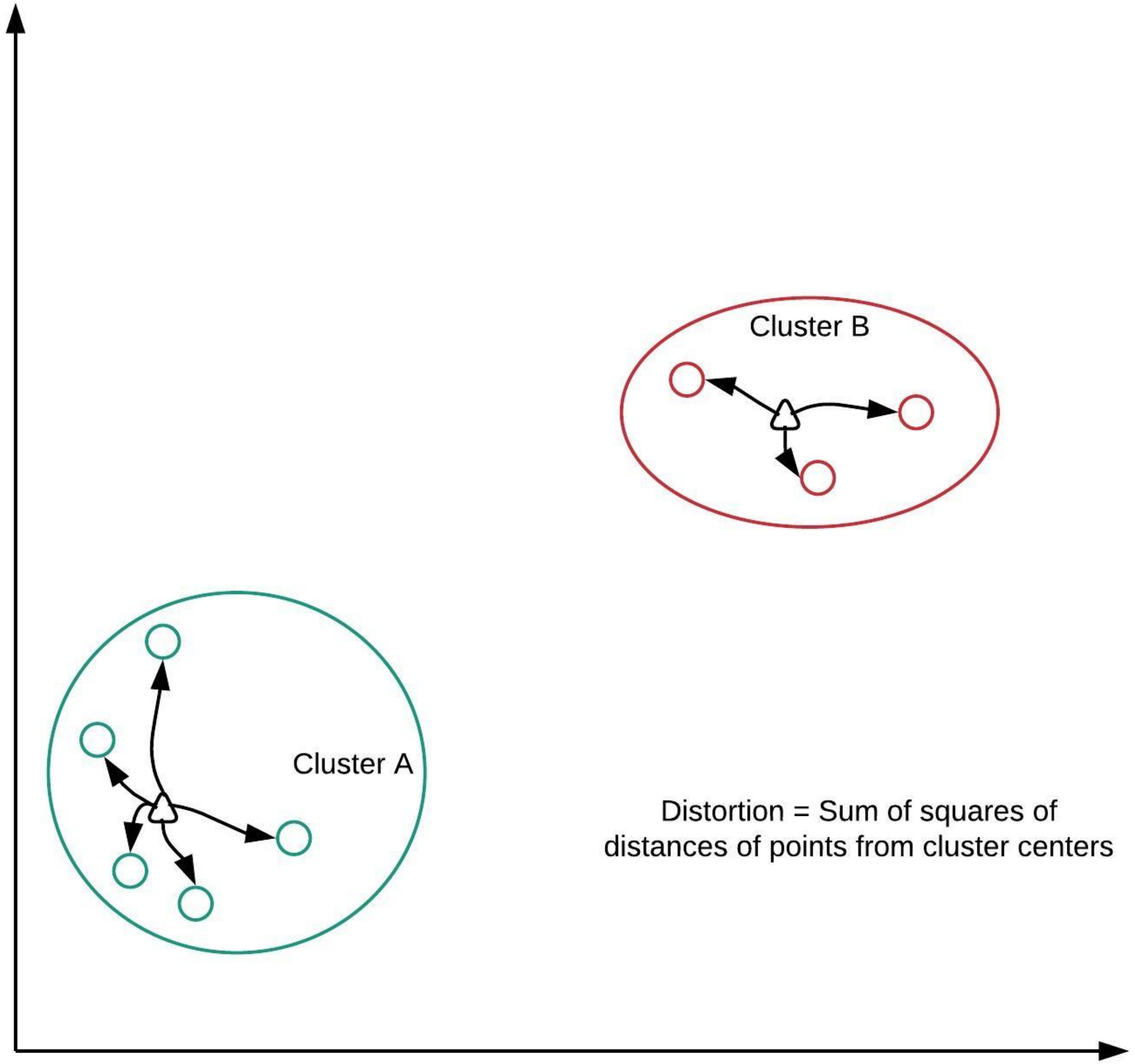
Step 1: Generate cluster centers

```
kmeans(obs, k_or_guess, iter, thresh, check_finite)
```

- `obs` : standardized observations
- `k_or_guess` : number of clusters
- `iter` : number of iterations (default: 20)
- `thres` : threshold (default: 1e-05)
- `check_finite` : whether to check if observations contain only finite numbers (default: True)

Returns two objects: cluster centers, distortion

How is distortion calculated?



Step 2: Generate cluster labels

```
vq(obs, code_book, check_finite=True)
```

- `obs` : standardized observations
- `code_book` : cluster centers
- `check_finite` : whether to check if observations contain only finite numbers (default: True)

Returns two objects: a list of cluster labels, a list of distortions

A note on distortions

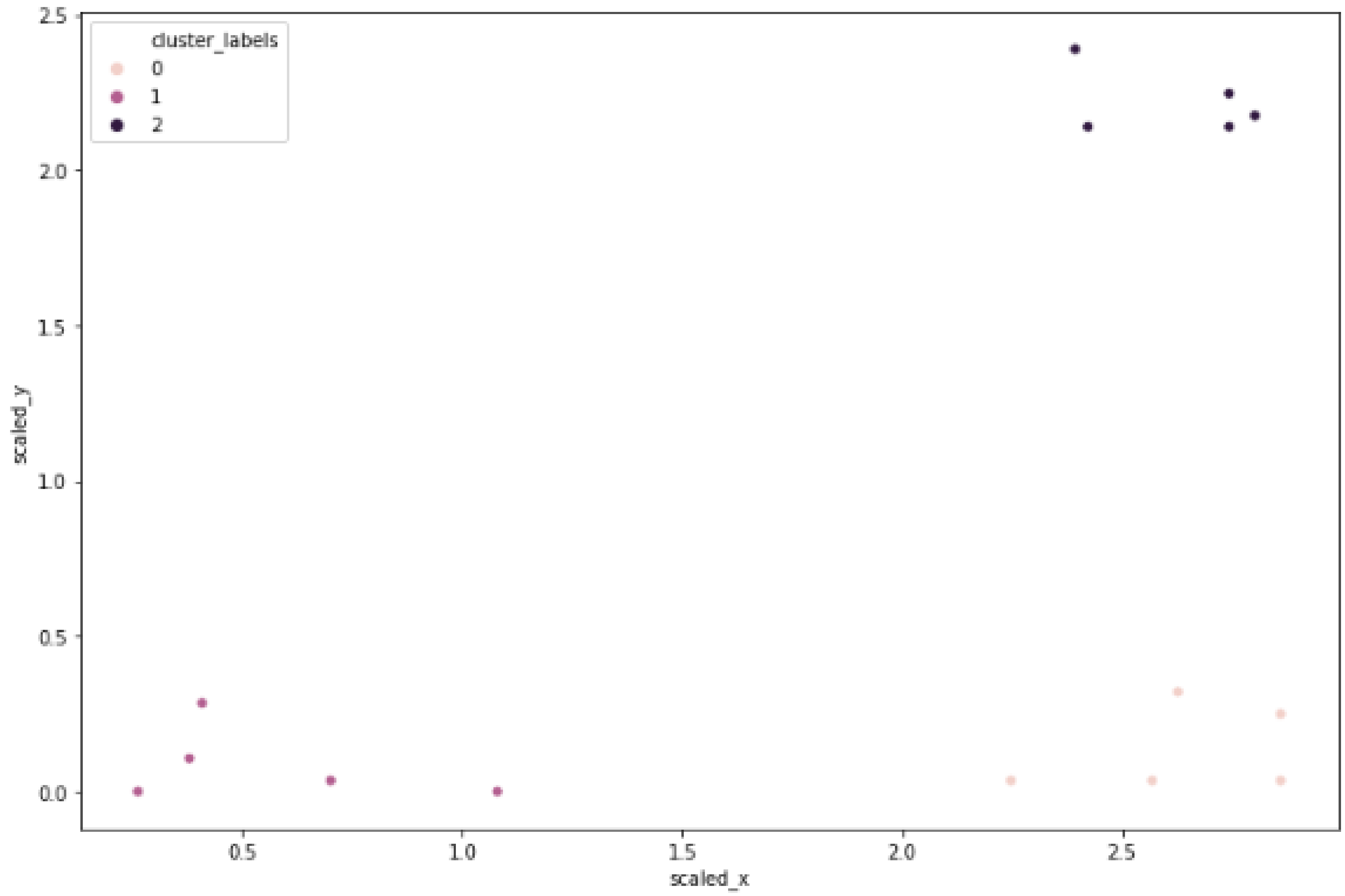
- `kmeans` returns a single value of distortions
- `vq` returns a list of distortions.

Running k-means

```
# Import kmeans and vq functions
from scipy.cluster.vq import kmeans, vq
```

```
# Generate cluster centers and labels
cluster_centers, _ = kmeans(df[['scaled_x', 'scaled_y']], 3)
df['cluster_labels'], _ = vq(df[['scaled_x', 'scaled_y']], cluster_centers)
```

```
# Plot clusters
sns.scatterplot(x='scaled_x', y='scaled_y', hue='cluster_labels', data=df)
plt.show()
```



cluster_labels
0
1
2
2.5
2.0
1.5
1.0
0.5
0.0
scaled_y
0.5
1.0
1.5
2.0
2.5
scaled_x

Next up: exercises!

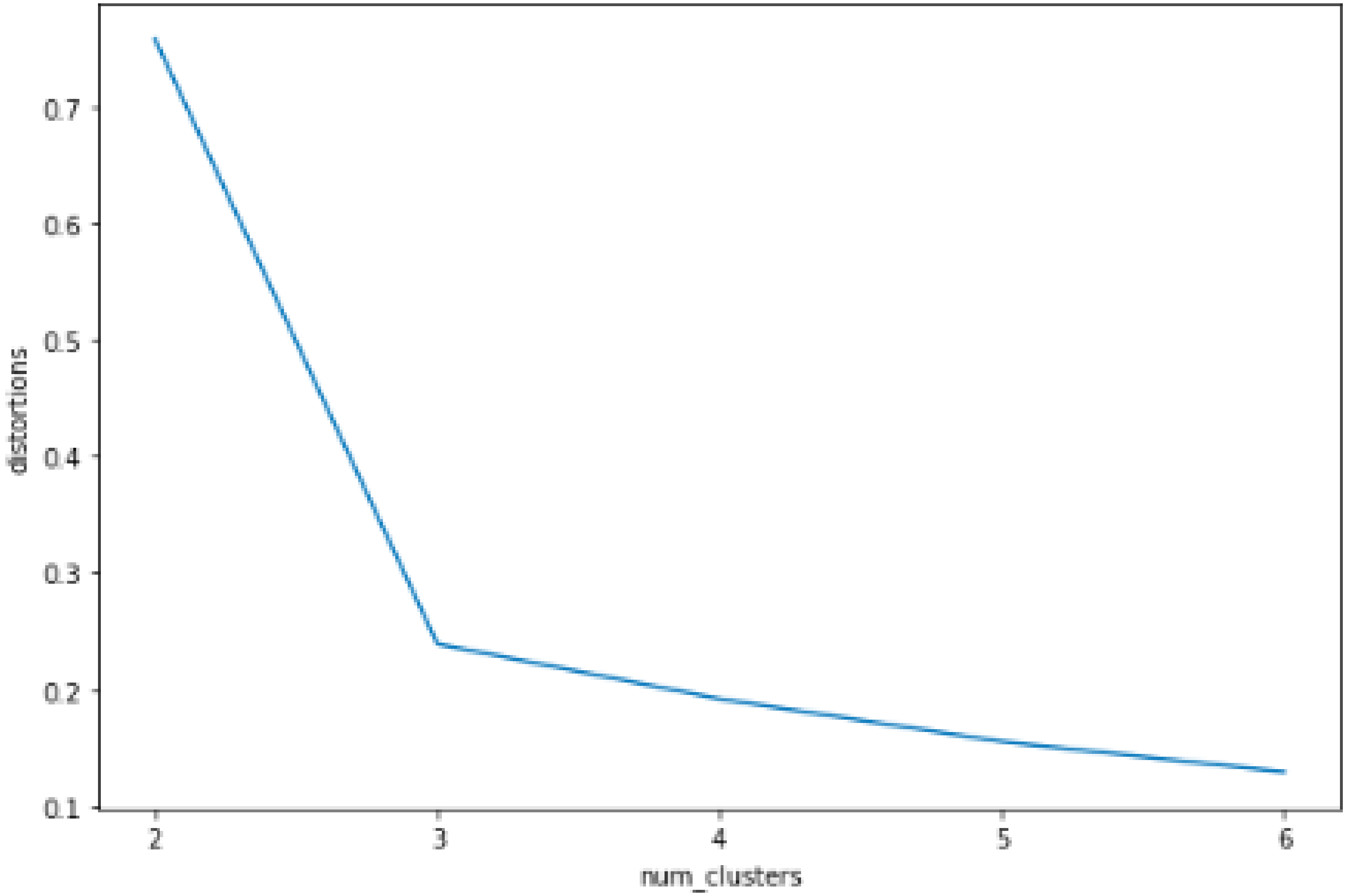
How many clusters?

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

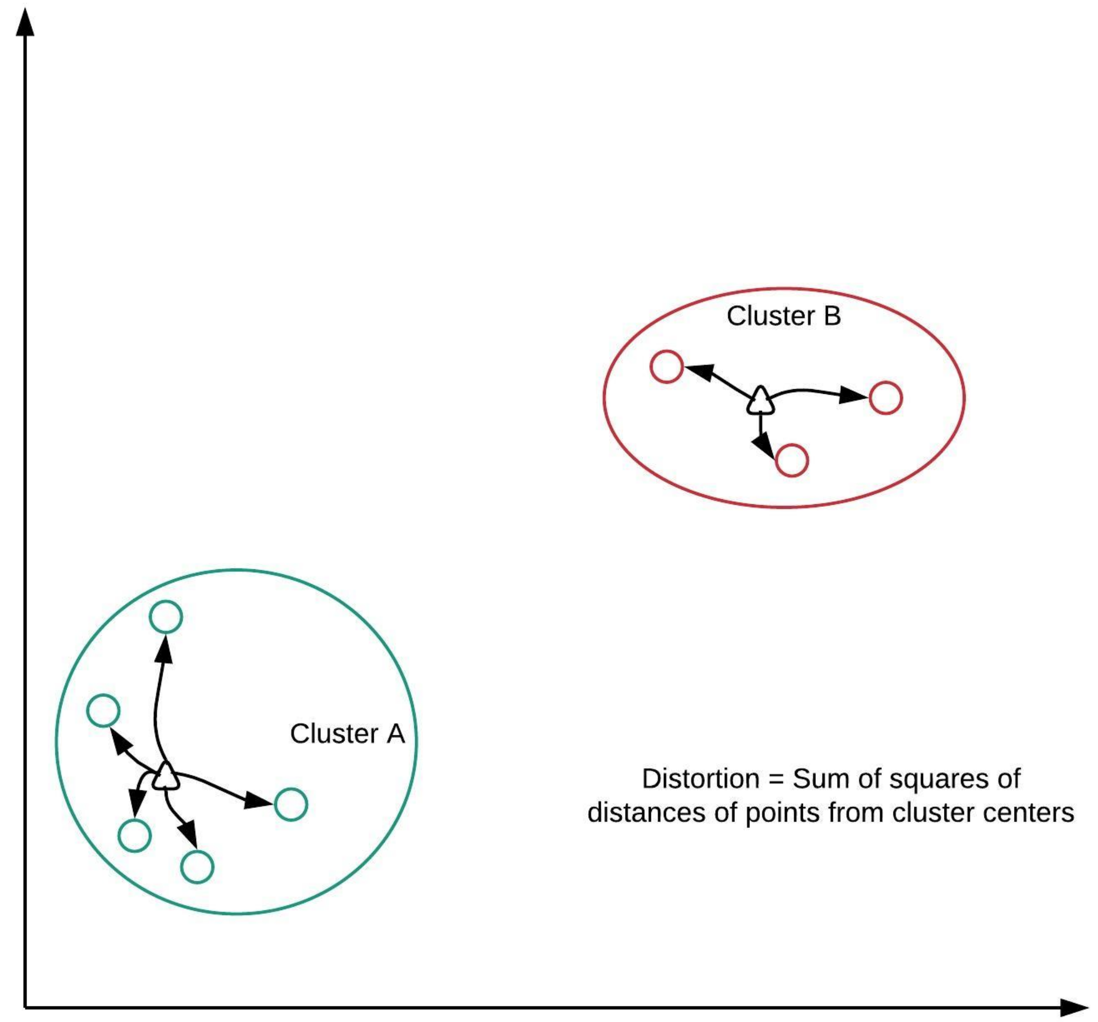
How to find the right k?

- No *absolute* method to find right number of clusters (k) in k-means clustering
- Elbow method



Distortions revisited

- Distortion: sum of squared distances of points from cluster centers
- Decreases with an increasing number of clusters
- Becomes zero when the number of clusters equals the number of points
- Elbow plot: line plot between cluster centers and distortion



Elbow method

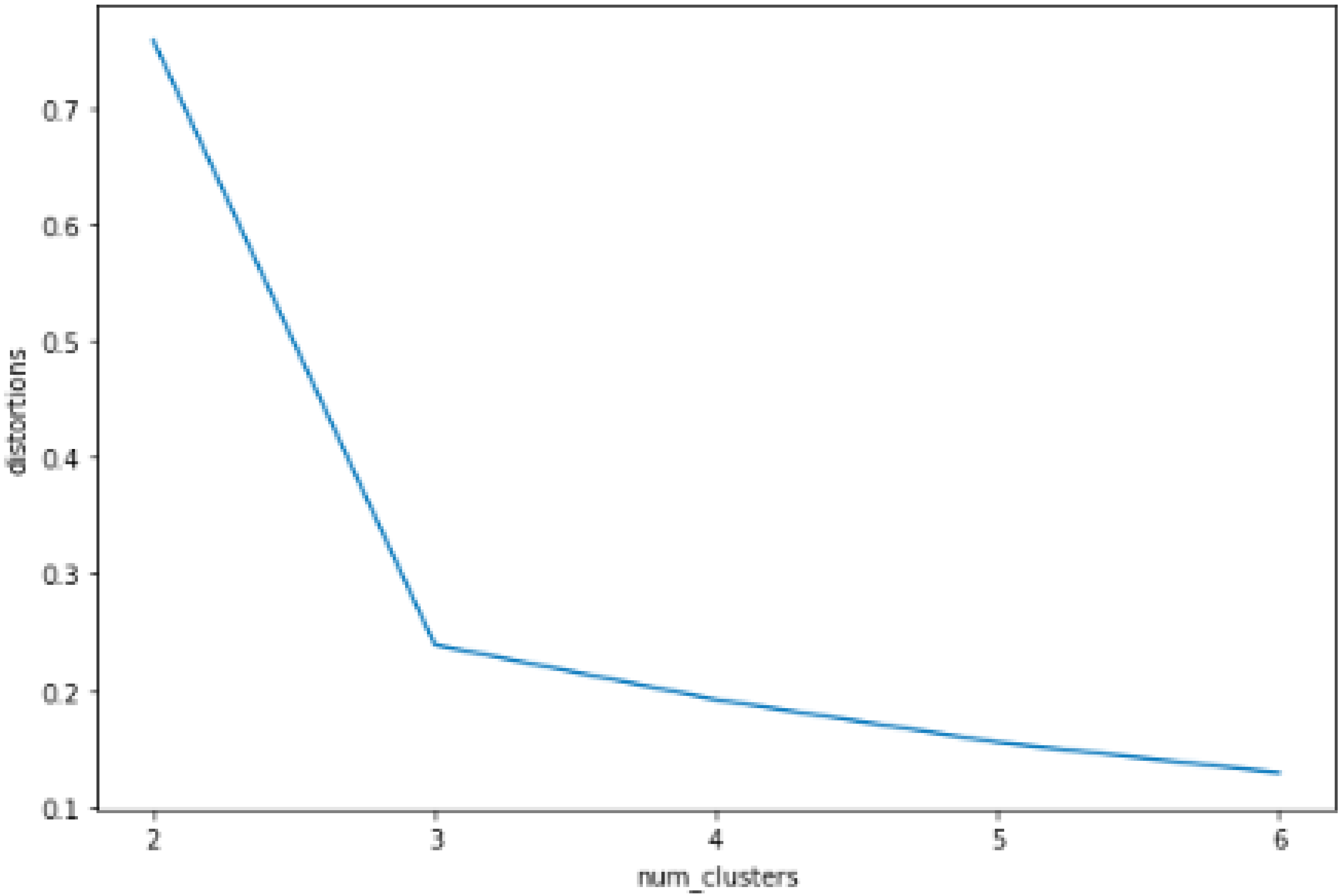
- Elbow plot: plot of the number of clusters and distortion
- Elbow plot helps indicate number of clusters present in data

Elbow method in Python

```
# Declaring variables for use
distortions = []

num_clusters = range(2, 7)
```

```
# Populating distortions for various clusters
for i in num_clusters:
    centroids, distortion = kmeans(df[['scaled_x', 'scaled_y']], i)
    distortions.append(distortion)
```

```
# Plotting elbow plot data
elbow_plot_data = pd.DataFrame({'num_clusters': num_clusters,
                                'distortions': distortions})

sns.lineplot(x='num_clusters', y='distortions',
             data = elbow_plot_data)
plt.show()
```


0.7
0.6
0.5
0.4
0.3
0.2
0.1
distortions
2
3
4
5
6
num_clusters

Final thoughts on using the elbow method

- Only gives an indication of optimal *k* (numbers of clusters)
- Does not always pinpoint how many *k* (numbers of clusters)
- Other methods: average silhouette and gap statistic

Next up: exercises

CLUSTERING METHODS WITH SCIPY

Limitations of k-means clustering

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Limitations of k-means clustering

- How to find the right *K* (number of clusters)?
- Impact of seeds
- Biased towards equal sized clusters

Impact of seeds

Initialize a random seed

```
from numpy import random
random.seed(12)
```

Seed: `np.array(1000, 2000)`

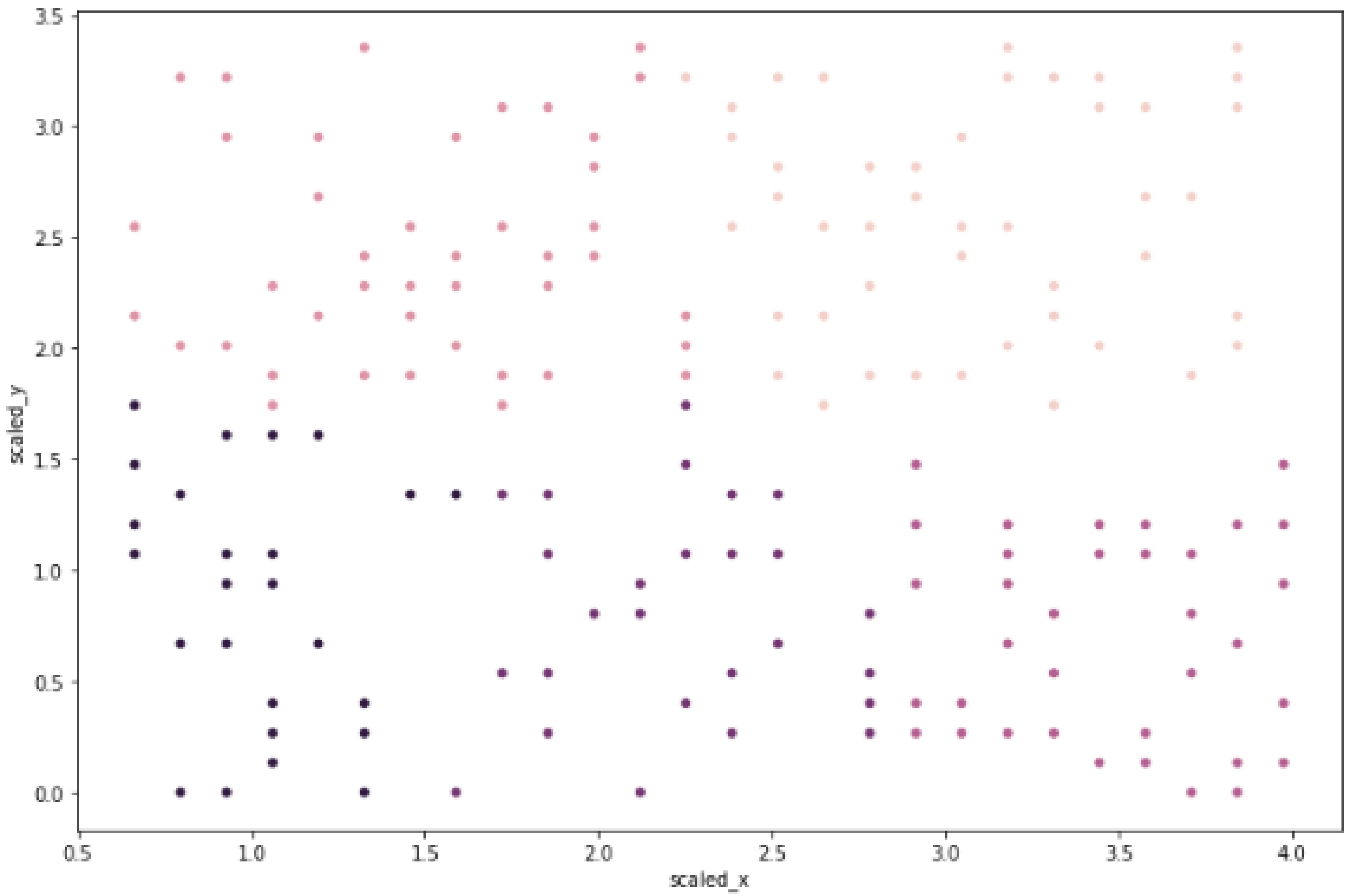
Cluster sizes: 29, 29, 43, 47, 52

Seed: `np.array(1,2,3)`

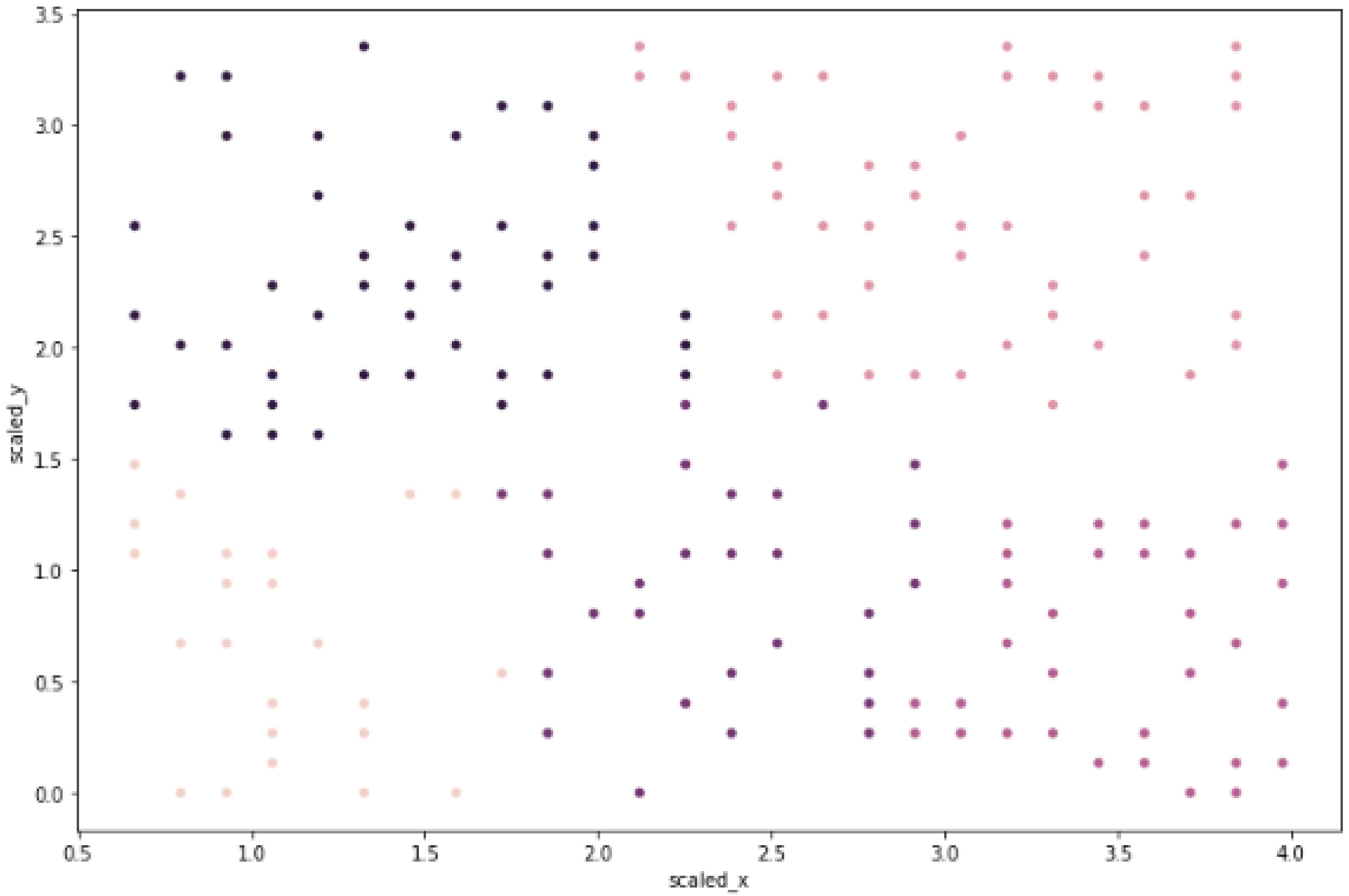
Cluster sizes: 26, 31, 40, 50, 53

Impact of seeds: plots

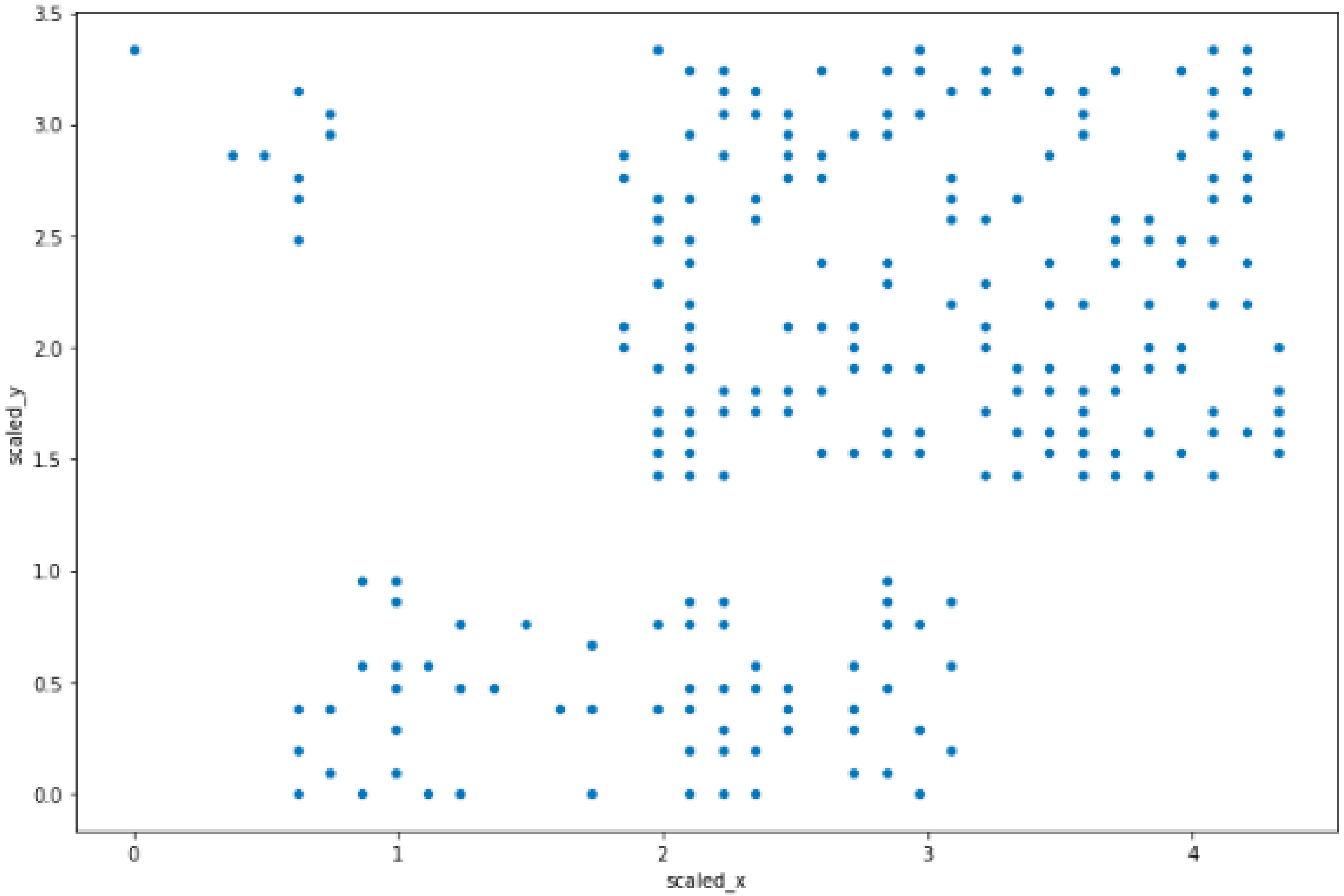
Seed: `np.array(1000, 2000)`



Seed: `np.array(1,2,3)`

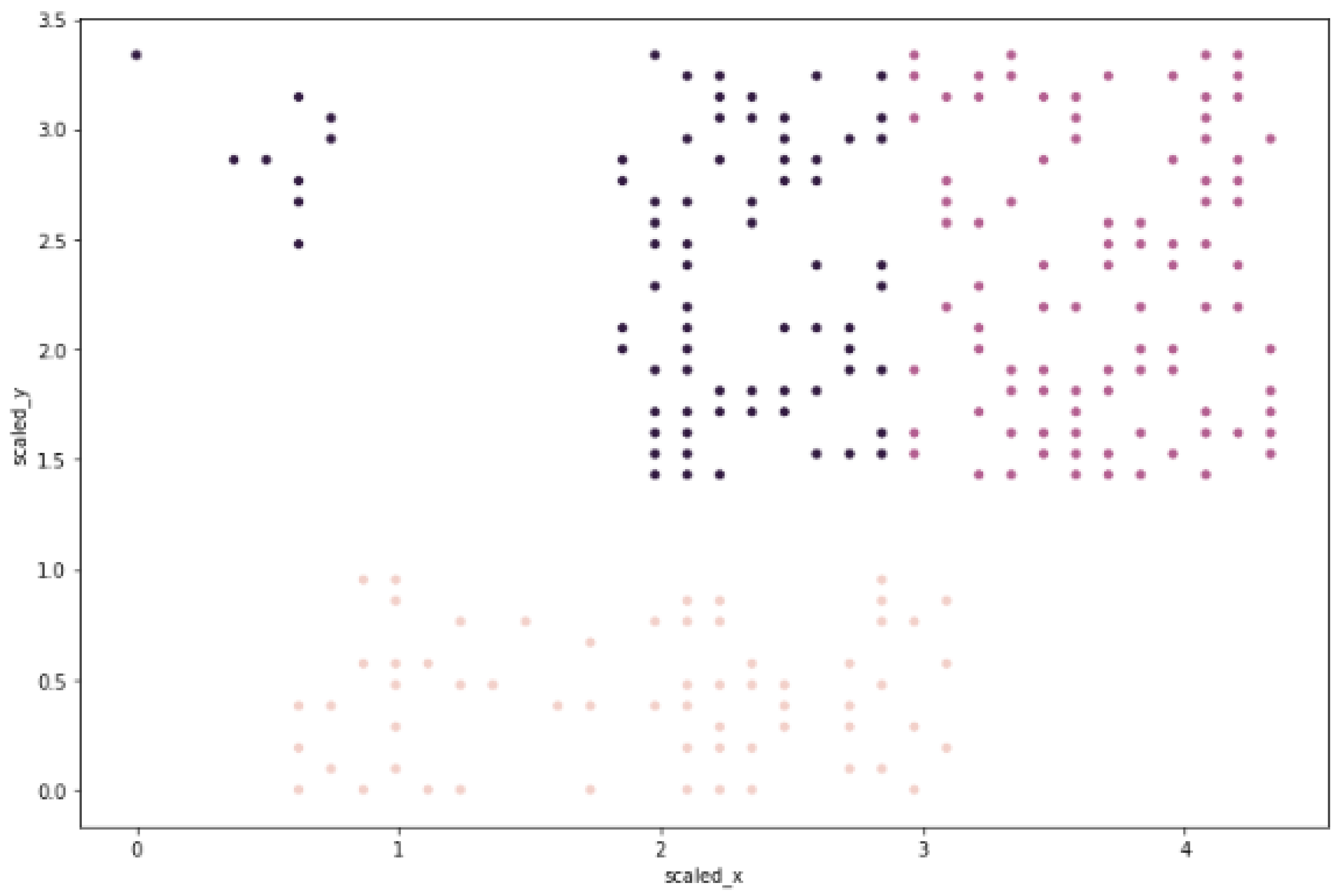


Uniform clusters in k means

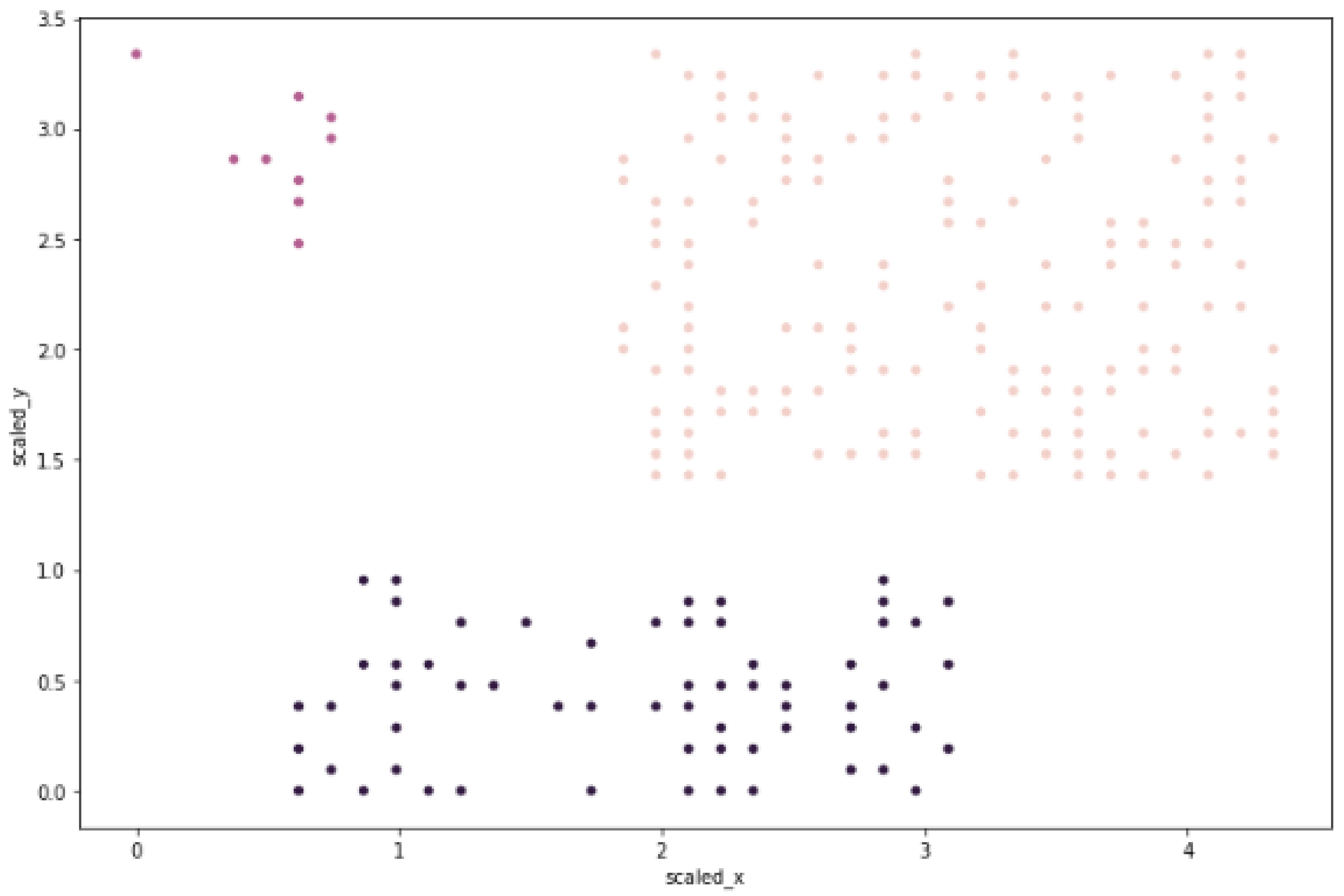


Uniform clusters in k-means: a comparison

K-means clustering with 3 clusters



Hierarchical clustering with 3 clusters



Final thoughts

- Each technique has its pros and cons
- Consider your data size and patterns before deciding on algorithm
- Clustering is exploratory phase of analysis

Next up: exercises

CLUSTERING METHODS WITH SCIPY

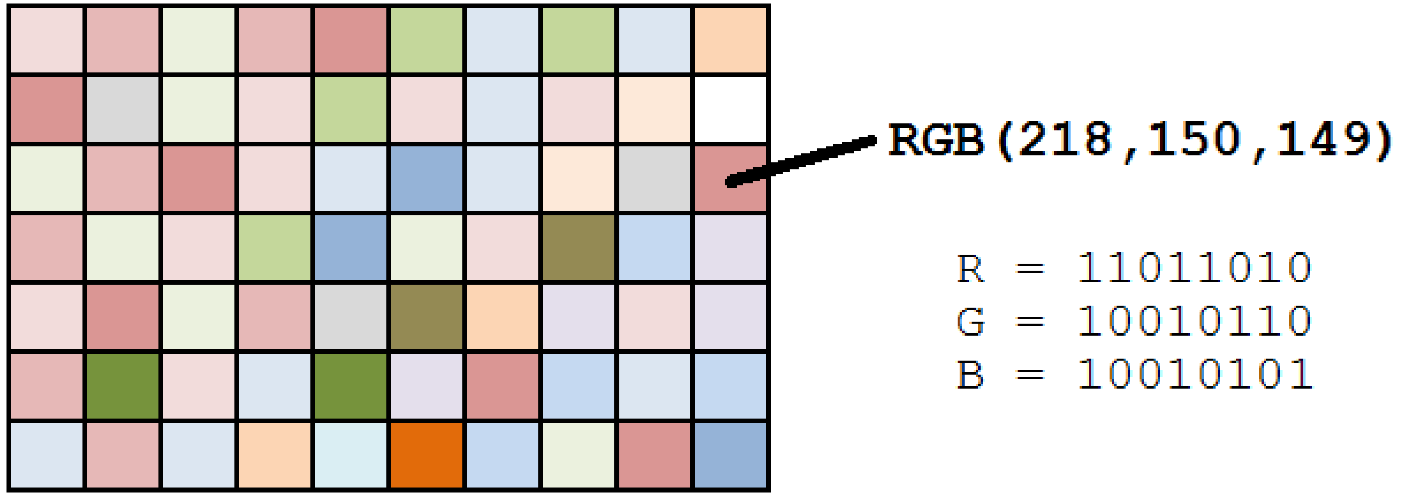
Dominant colors in images

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

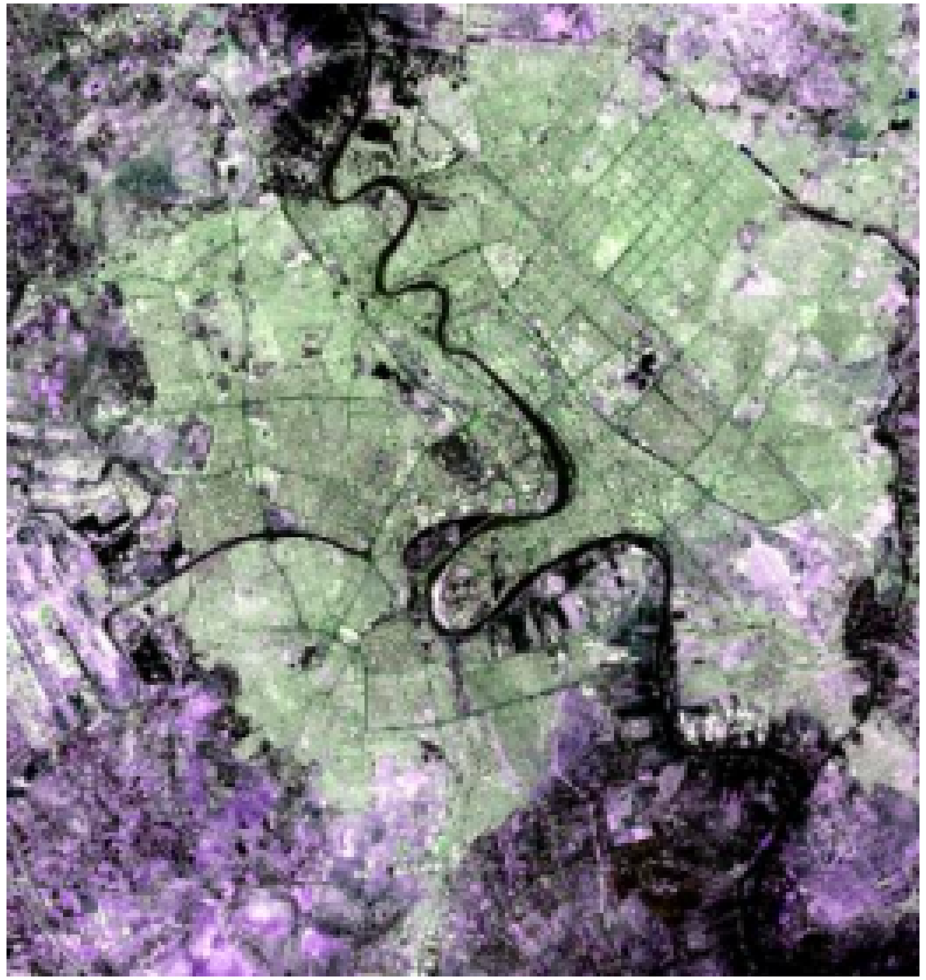
Dominant colors in images

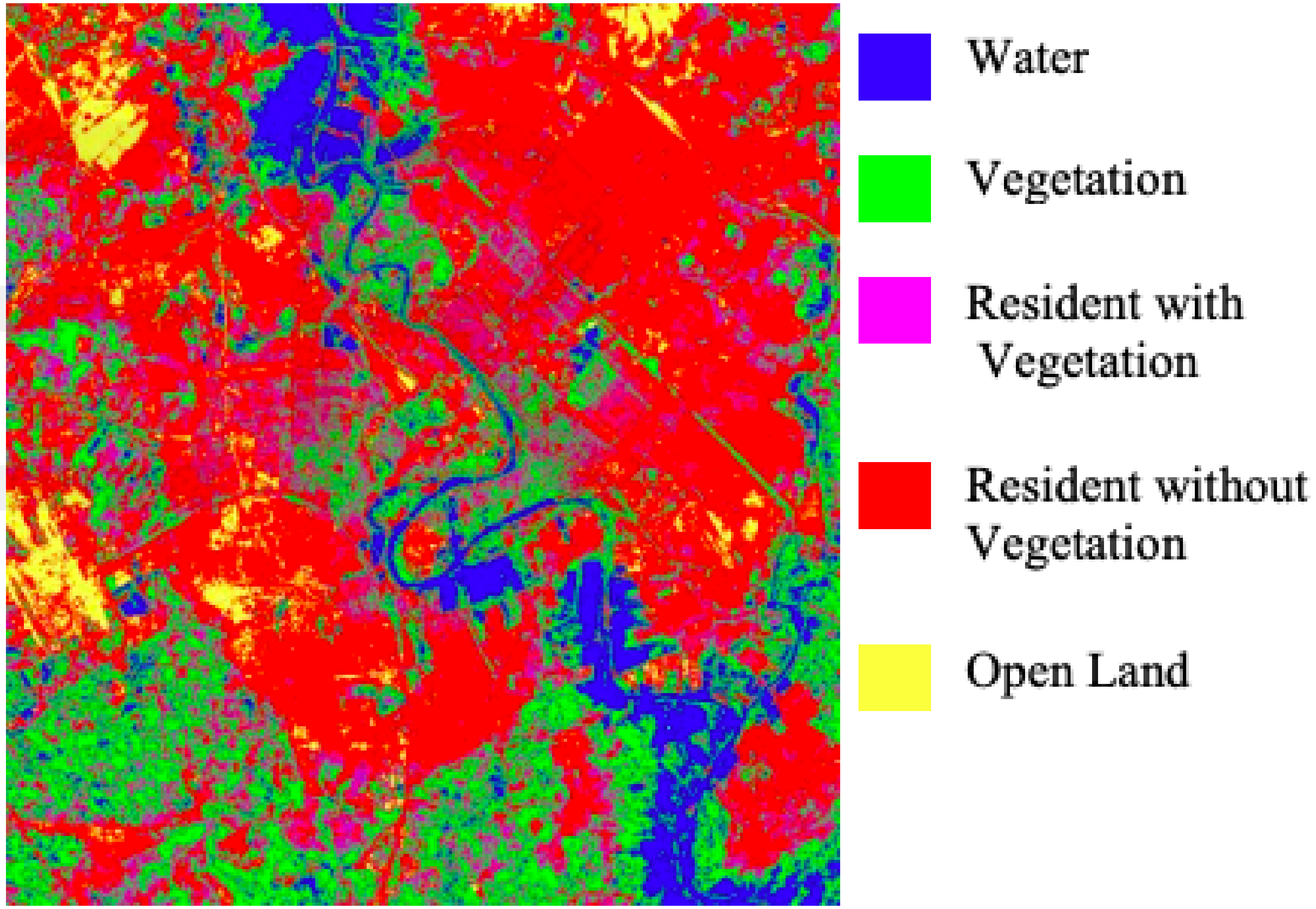
- All images consist of pixels
- Each pixel has three values: *Red*, *Green* and *Blue*
- Pixel color: combination of these RGB values
- Perform k-means on standardized RGB values to find cluster centers
- Uses: Identifying features in satellite images



Source

Feature identification in satellite images





Source

Tools to find dominant colors

- Convert image to pixels: `matplotlib.image.imread`
- Display colors of cluster centers: `matplotlib.pyplot.imshow`

Convert image to RGB matrix

```python
import matplotlib.image as img
image = img.imread('sea.jpg')
image.shape
```

```
(475, 764, 3)
```

```python
r = []
g = []
b = []

for row in image:
    for pixel in row:
        # A pixel contains RGB values
        temp_r, temp_g, temp_b = pixel
        r.append(temp_r)
        g.append(temp_g)
        b.append(temp_b)
```

Data frame with RGB values

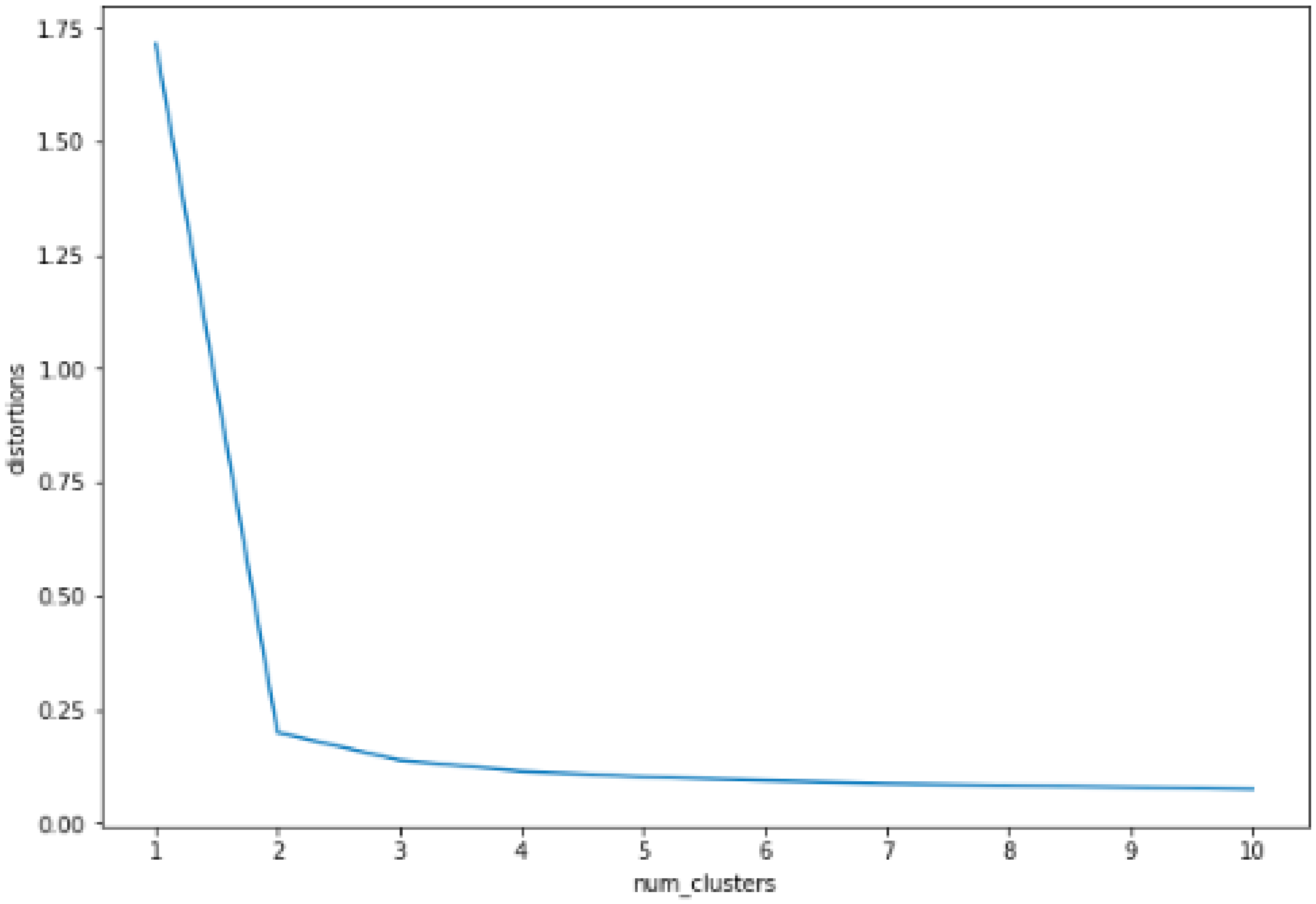
```
pixels = pd.DataFrame({'red': r,
                       'blue': b,
                       'green': g})

pixels.head()
```

red	blue	green
252	255	252
75	103	81
...	...	...

Create an elbow plot

```
distortions = []
num_clusters = range(1, 11)

# Create a list of distortions from the kmeans method
for i in num_clusters:
    cluster_centers, _ = kmeans(pixels[['scaled_red', 'scaled_blue',
                                        'scaled_green']], i)
    distortions.append(distortion)

# Create a data frame with two lists - number of clusters and distortions
elbow_plot = pd.DataFrame({'num_clusters': num_clusters,
                           'distortions': distortions})

# Creat a line plot of num_clusters and distortions
sns.lineplot(x='num_clusters', y='distortions', data = elbow_plot)
plt.xticks(num_clusters)
plt.show()
```

Elbow plot



1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
distortions
1
2
3
4
5
6
7
8
9
10
num_clusters

Find dominant colors

```
cluster_centers, _ = kmeans(pixels[['scaled_red', 'scaled_blue',
                                    'scaled_green']], 2)
```

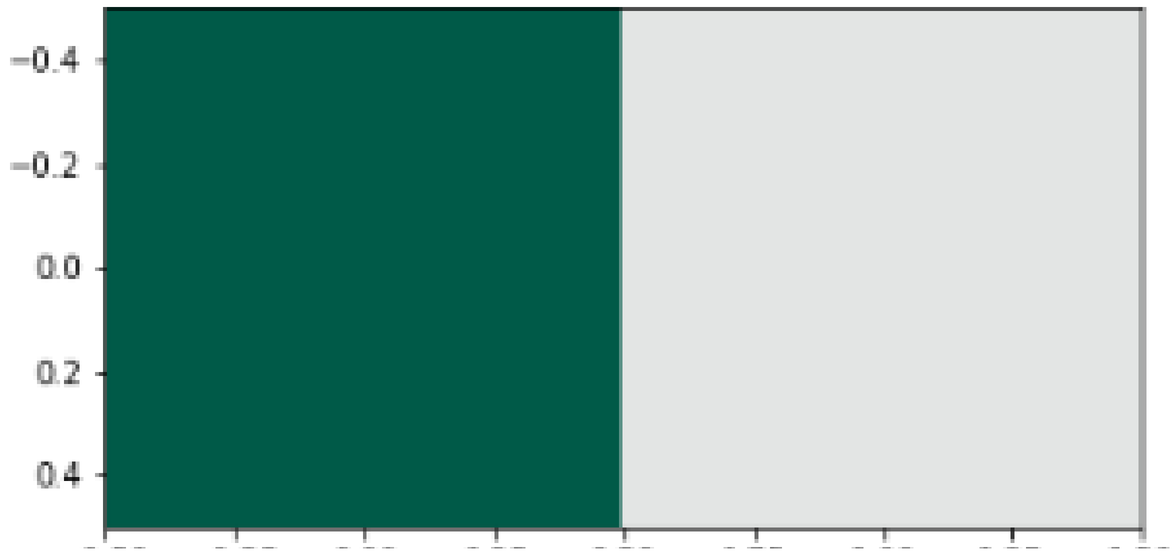
```
colors = []

# Find Standard Deviations
r_std, g_std, b_std = pixels[['red', 'blue', 'green']].std()

# Scale actual RGB values in range of 0-1
for cluster_center in cluster_centers:
    scaled_r, scaled_g, scaled_b = cluster_center
    colors.append((
        scaled_r * r_std/255,
        scaled_g * g_std/255,
        scaled_b * b_std/255
    ))
```

Display dominant colors

```
#Dimensions: 2 x 3 (N X 3 matrix)
print(colors)
```

```
[(0.08192923122023911, 0.34205845943857993, 0.2824002984155429),
 (0.893281510956742, 0.899818770315129, 0.8979114272960784)]
```

```
#Dimensions: 1 x 2 x 3 (1 X N x 3 matrix)
plt.imshow([colors])
plt.show()
```



Next up: exercises

Document clustering

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Document clustering: concepts

1. Clean data before processing
2. Determine the importance of the terms in a document (in TF-IDF matrix)
3. Cluster the TF-IDF matrix
4. Find top terms, documents in each cluster

Clean and tokenize data

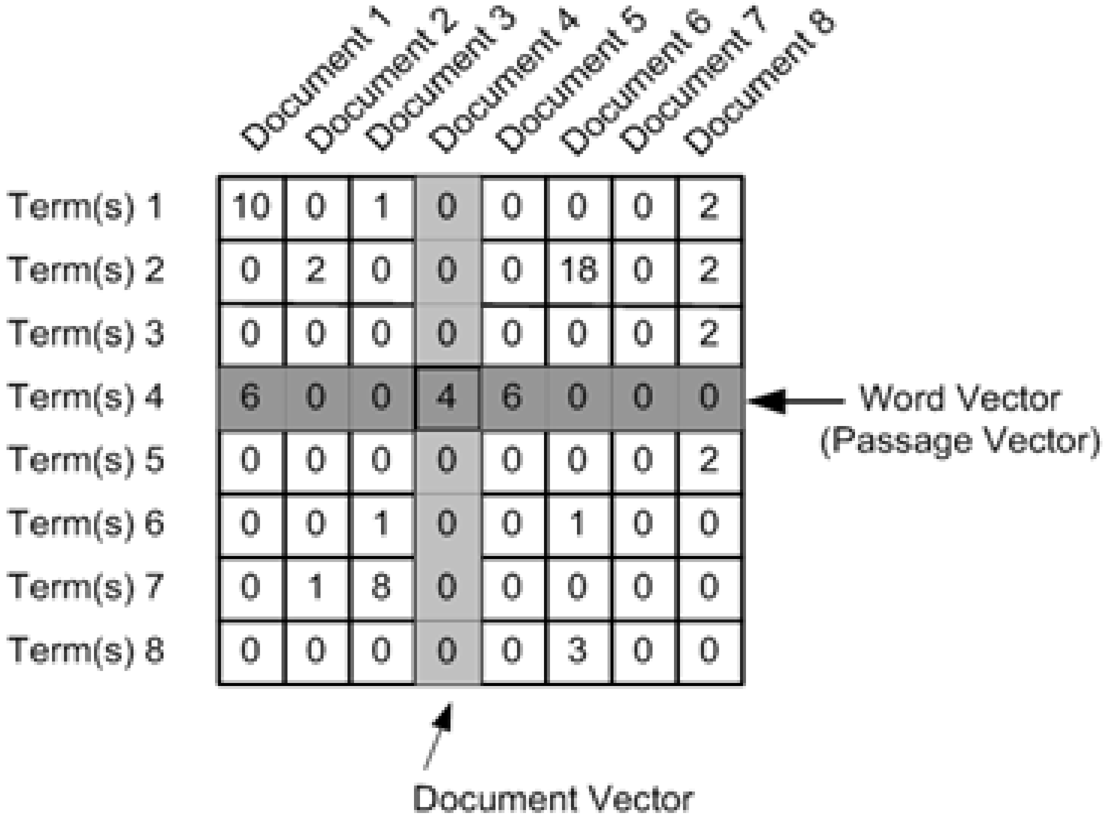
- Convert text into smaller parts called tokens, clean data for processing

```
from nltk.tokenize import word_tokenize
import re

def remove_noise(text, stop_words = []):
    tokens = word_tokenize(text)
    cleaned_tokens = []
    for token in tokens:
        token = re.sub('[^A-Za-z0-9]+', '', token)
        if len(token) > 1 and token.lower() not in stop_words:
            # Get lowercase
            cleaned_tokens.append(token.lower())
    return cleaned_tokens
remove_noise("It is lovely weather we are having.
              I hope the weather continues.")
```

```
['lovely', 'weather', 'hope', 'weather', 'continues']
```

Document term matrix and sparse matrices

- Document term matrix formed
- Most elements in matrix are zeros



Source

- Sparse matrix is created

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Source

TF-IDF (Term Frequency - Inverse Document Frequency)

- A weighted measure: evaluate how important a word is to a document in a collection

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=50,
                                   min_df=0.2, tokenizer=remove_noise)
tfidf_matrix = tfidf_vectorizer.fit_transform(data)
```

Clustering with sparse matrix

- `kmeans()` in SciPy does not support sparse matrices
- Use `.todense()` to convert to a matrix

```
cluster_centers, distortion = kmeans(tfidf_matrix.todense(), num_clusters)
```

Top terms per cluster

- Cluster centers: lists with a size equal to the number of terms
- Each value in the cluster center is its importance
- Create a dictionary and print top terms

```
terms = tfidf_vectorizer.get_feature_names()

for i in range(num_clusters):
    center_terms = dict(zip(terms, list(cluster_centers[i])))
    sorted_terms = sorted(center_terms, key=center_terms.get, reverse=True)
    print(sorted_terms[:3])
```

```
['room', 'hotel', 'staff']

['bad', 'location', 'breakfast']
```

More considerations

- Work with hyperlinks, emoticons etc.
- Normalize words (run, ran, running -> run)
- `.todense()` may not work with large datasets

Next up: exercises!

CLUSTERING METHODS WITH SCIPY

Clustering with multiple features

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

Basic checks

```
# Cluster centers
print(fifa.groupby('cluster_labels')[['scaled_heading_accuracy',
    'scaled_volleys', 'scaled_finishing']].mean())
```

cluster_labels	scaled_heading_accuracy	scaled_volleys	scaled_finishing
0	3.21	2.83	2.76
1	0.71	0.64	0.58

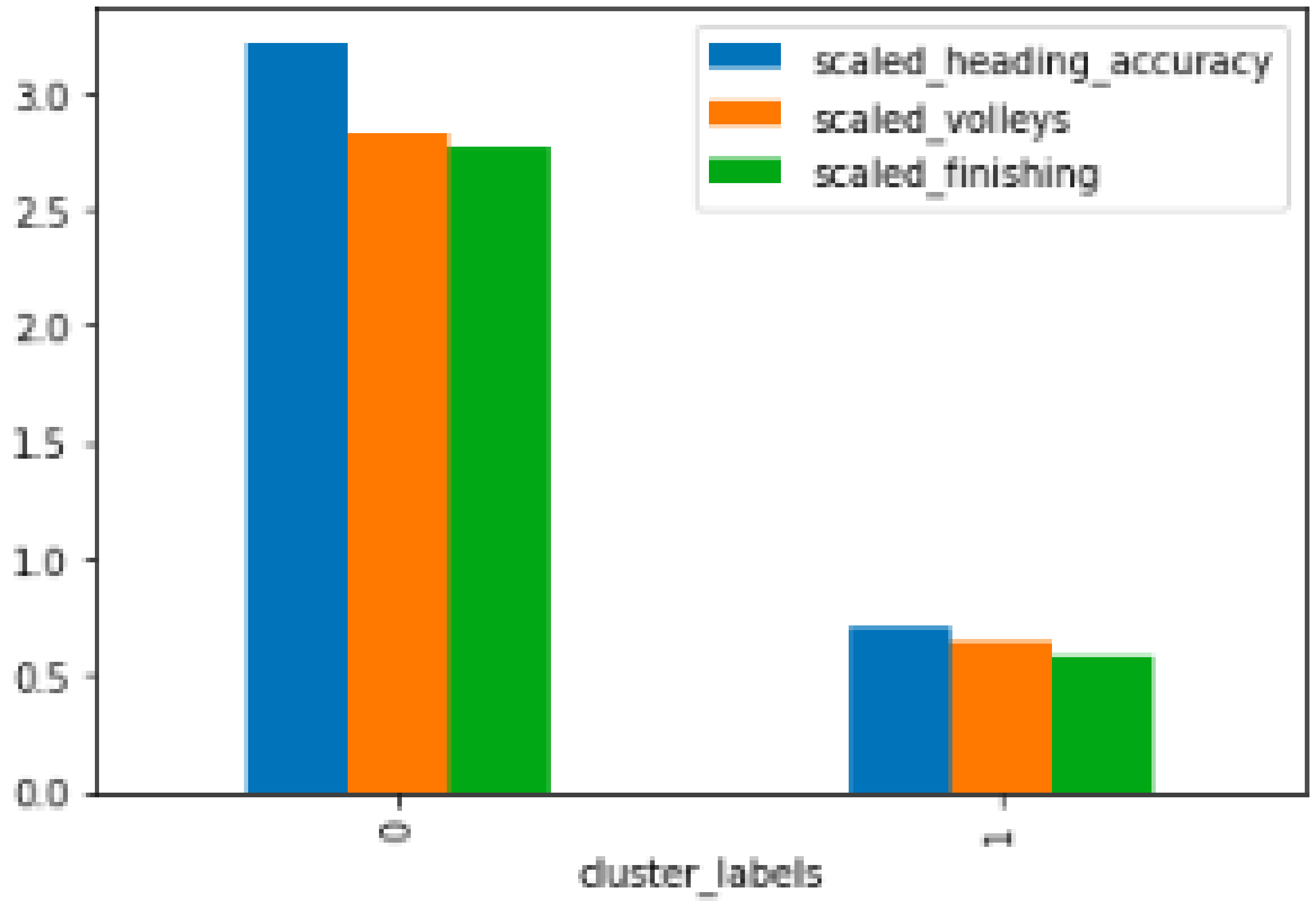
```
# Cluster sizes
print(fifa.groupby('cluster_labels')['ID'].count())
```

cluster_labels	count
0	886

Visualizations

- Visualize cluster centers
- Visualize other variables for each cluster

```
# Plot cluster centers
fifa.groupby('cluster_labels') \
  [scaled_features].mean()
  .plot(kind='bar')
plt.show()
```



Top items in clusters

```
# Get the name column of top 5 players in each cluster
for cluster in fifa['cluster_labels'].unique():
    print(cluster, fifa[fifa['cluster_labels'] == cluster]['name'].values[:5])
```

Cluster Label	Top Players
0	['Cristiano Ronaldo' 'L. Messi' 'Neymar' 'L. Suárez' 'R. Lewandowski']
1	['M. Neuer' 'De Gea' 'G. Buffon' 'T. Courtois' 'H. Lloris']

Feature reduction

- Factor analysis
- Multidimensional scaling

Final exercises!

CLUSTERING METHODS WITH SCIPY

Farewell!

CLUSTERING METHODS WITH SCIPY

Shaumik Daityari
Business Analyst

What comes next?

- Clustering is one of the exploratory steps
- More courses on DataCamp
- Practice, practice, practice!

Until next time

CLUSTERING METHODS WITH SCIPY