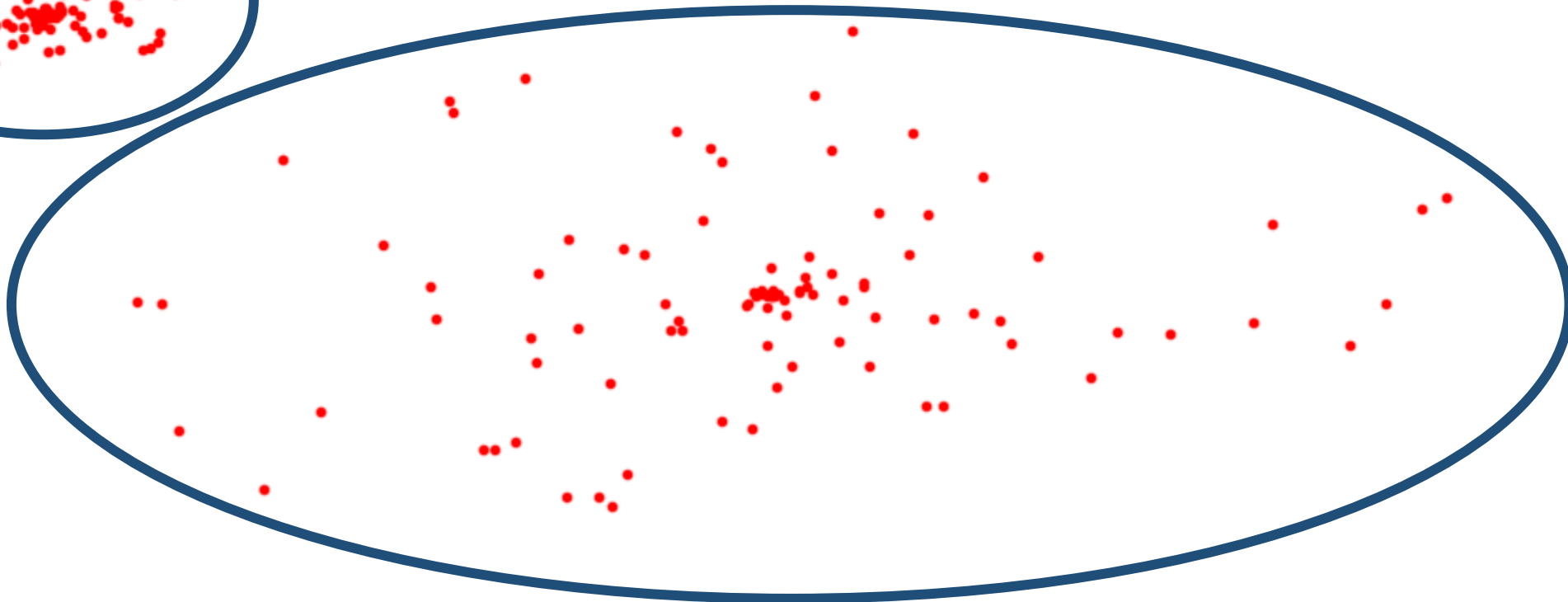
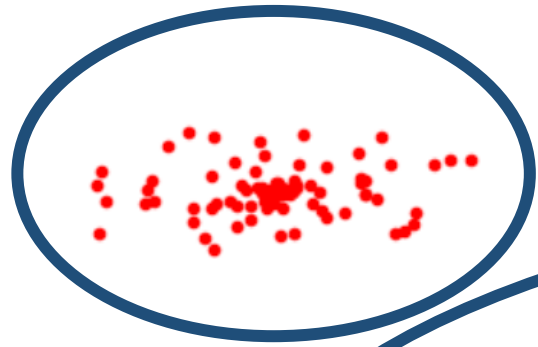
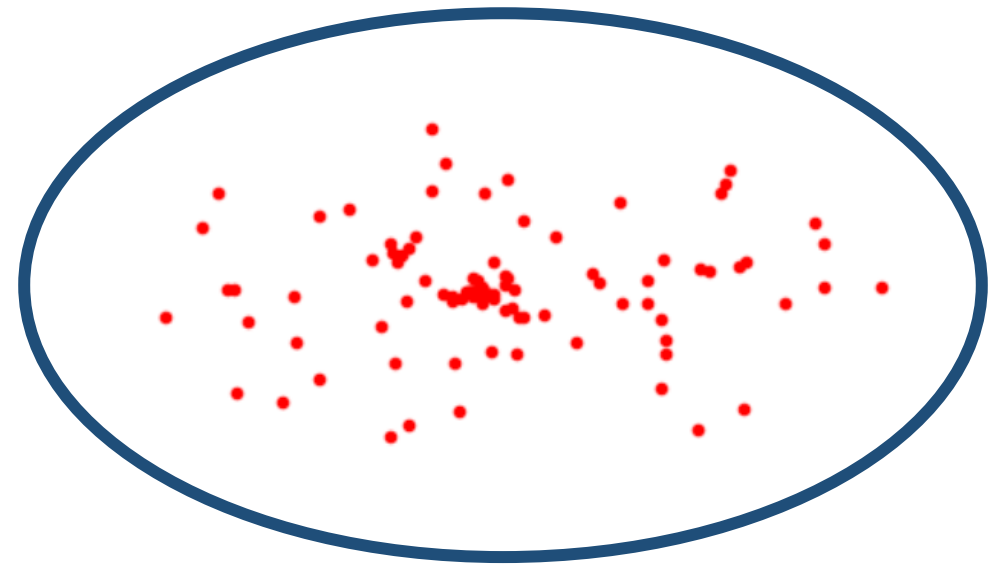


Clustering

- k-means
- `scipy.cluster.vq.kmeans`

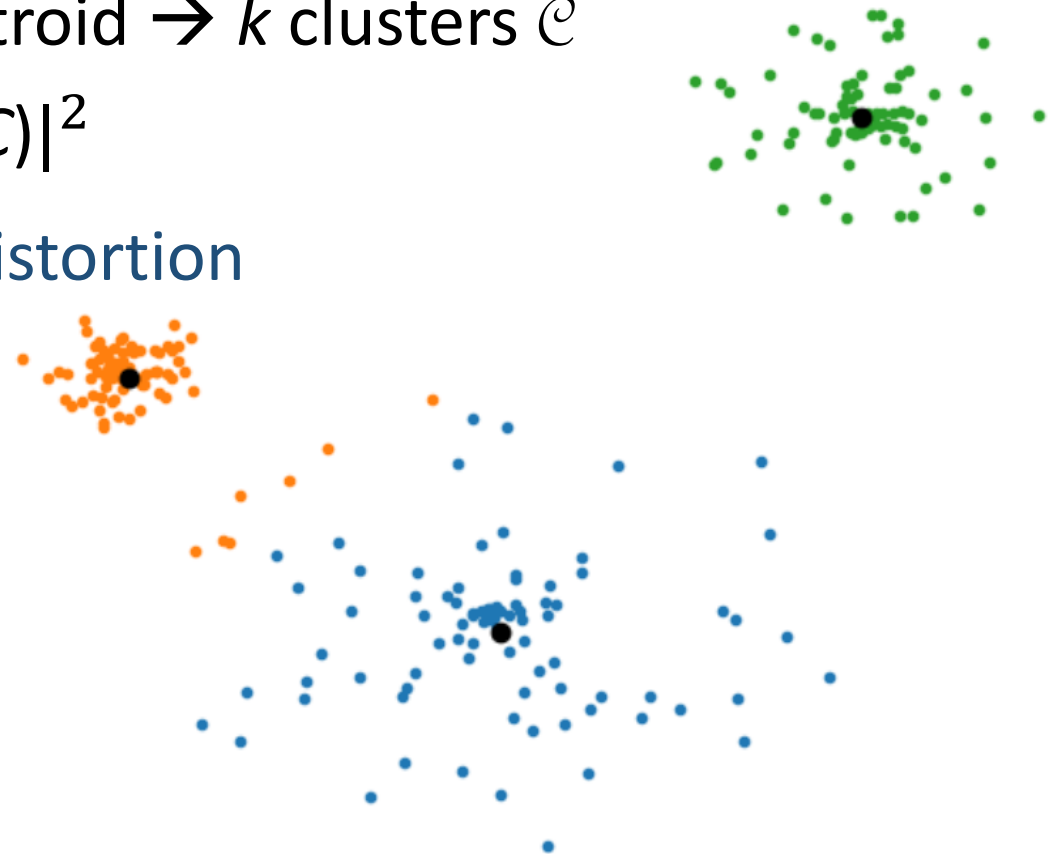
3 clusters / groups of points



Clustering = Optimization problem

Example: k-means

- Find k points *centroids*
- Assign each input point to nearest centroid $\rightarrow k$ clusters \mathcal{C}
- **distortion** = $\sum_{C \in \mathcal{C}} \sum_{p \in C} |p - \text{centroid}(C)|^2$
- **Goal** : Find k centroids that minimize distortion



k-means for $k = 1$

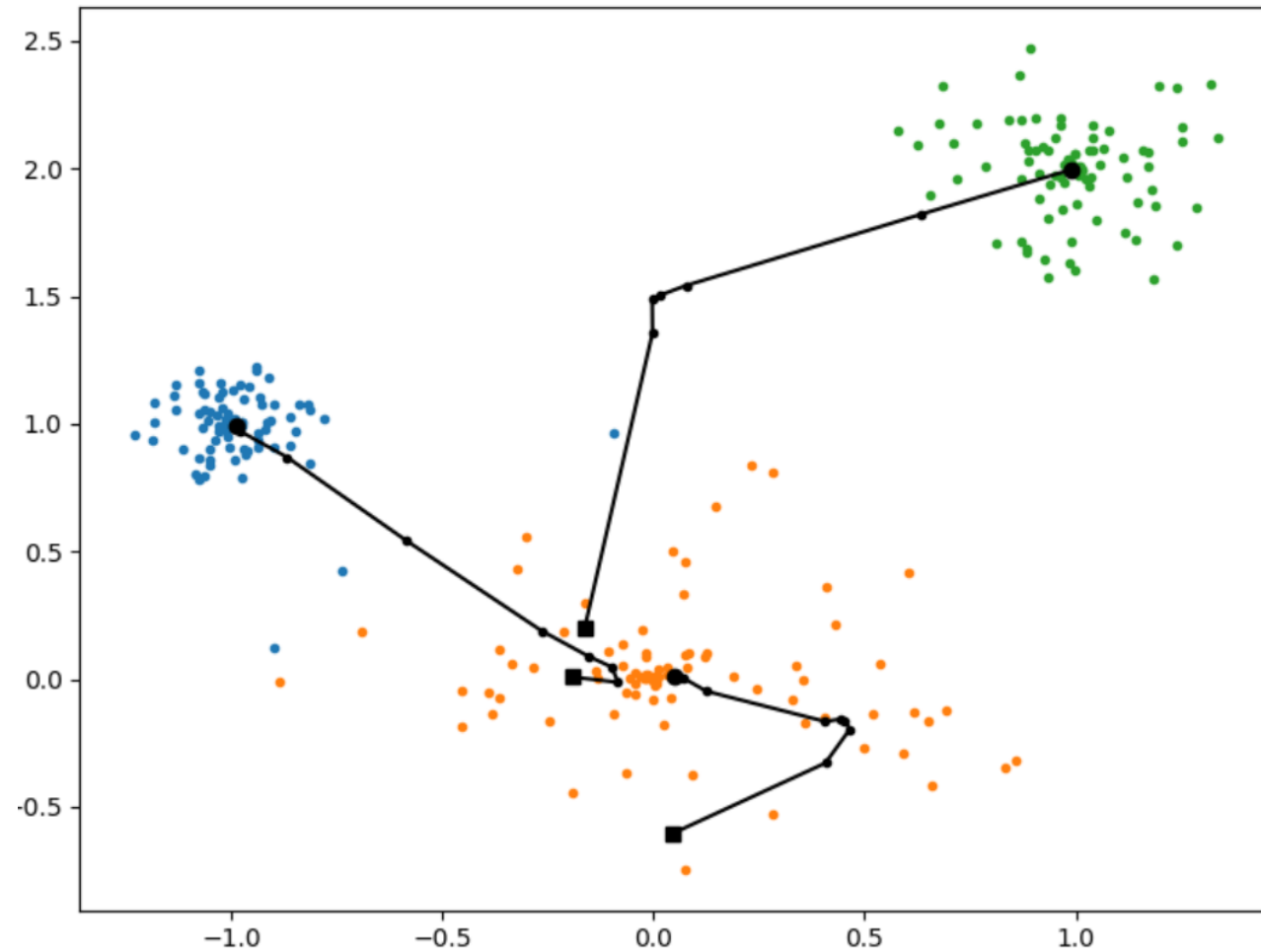
- Let the centroid point c for a point set C be the point minimizing the

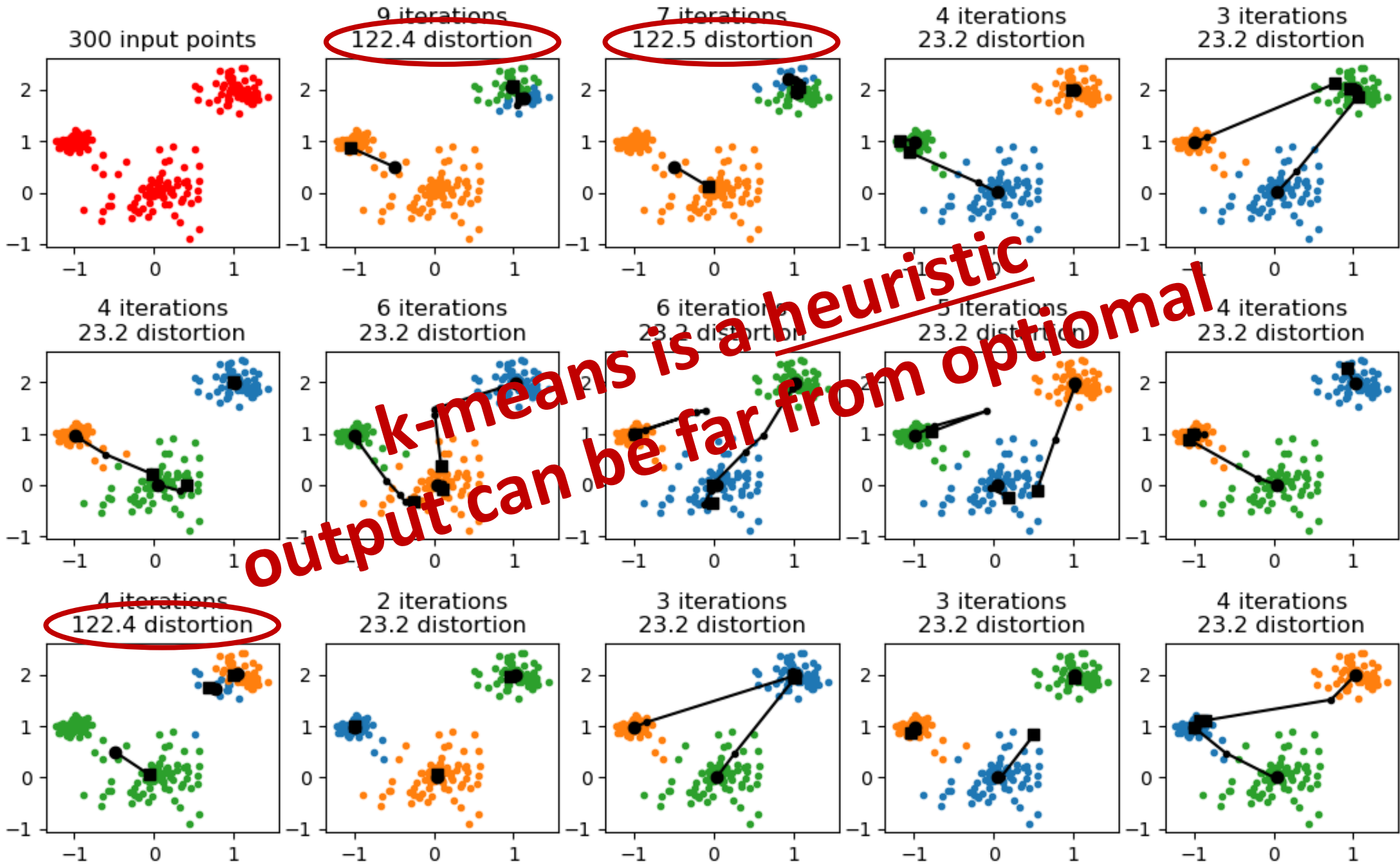
$$\text{distortion} = \sum_{p \in C} |p - c|^2$$

- Theorem $c = \text{average}(C)$

k-means - Lloyd's method (pseudo code)

```
centroids = k distinct random input points
while centroids change:
    create clusters C by assigning points to the nearest centroid
    centroids = average of each cluster
```





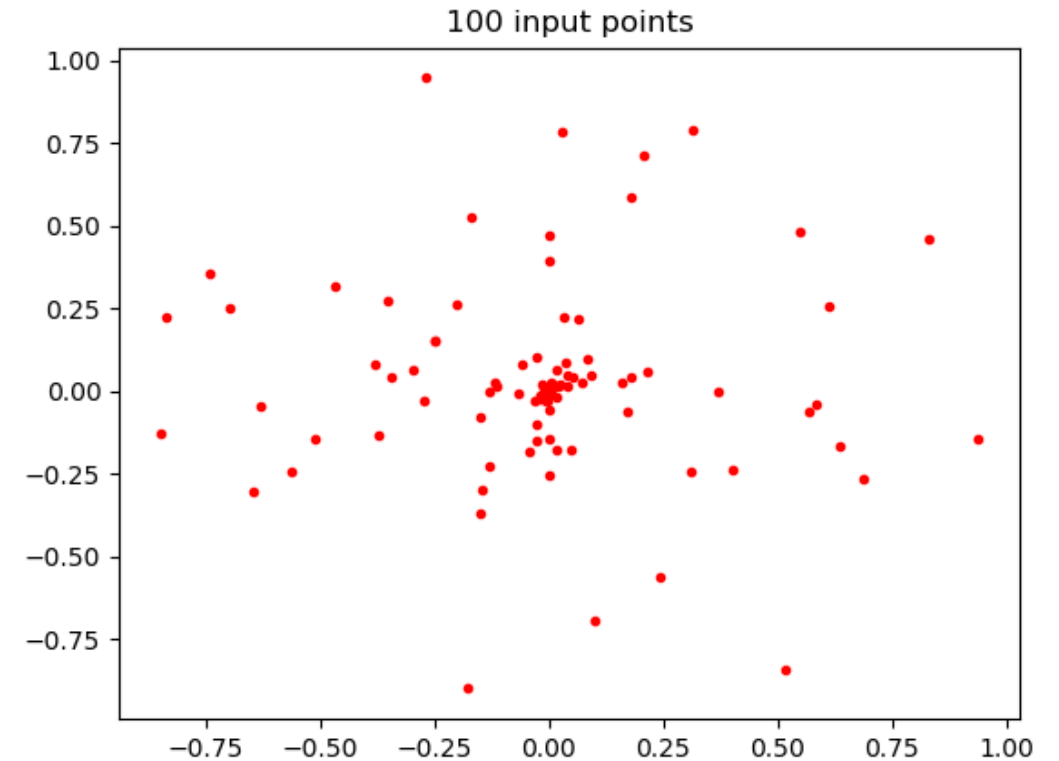
Generating random points (just one random approach)

`k_means.py`

```
from random import random
from math import pi, cos, sin

def random_point(x, y, radius):
    angle = 2 * pi * random()
    r = radius * random() ** 2
    return x + r * cos(angle), y + r * sin(angle)

def random_points(n, x, y, radius):
    for _ in range(n):
        yield random_point(x, y, radius)
```



k-means

```
k_means.py
```

```
from random import sample
from numpy import argmin

def k_means(points, k):
    centroid = sample(points, k)
    centroids = [ centroid ]

    while True:
        clusters = [[] for _ in centroid]
        for p in points:
            i = argmin([dist(p, c) for c in centroid])
            clusters[i].append(p)

        centroid = [tuple(map(mean, zip(*c))) for c in clusters]
        if centroid == centroids[-1]:
            break

        centroids.append(centroid)
        if min(len(c) for c in clusters) == 0:
            print("Not good - empty cluster")
            break

    return clusters
```


k-mean limitations

- Can easily converge to a solution far from a global minimum
 - Solution – try several times and take the best
- Clusters can become empty
 - Solution – discard and restart / take a random point out as a new centroid / take point furthest away from existing centroids /
- Sensitive to the scales of the different dimensions
 - Solution – apply some kind of initial normalization of coordinates

k-means - better bounds

- The **k-means++** algorithm achieves an **expected guarantee** to be at most a factor $8(\ln k + 2)$ from the optimal [Vassilvitskii & Arthur]
- There exist **polynomial time approximation schemes** that find a solution that is guaranteed $1 + \epsilon$ of the optimal (but running time exponential in k and dimension of points) [Har Peled et al.]
- **In practice: A heuristic is most often the algorithm of choice**

scipy.cluster.vq.kmeans

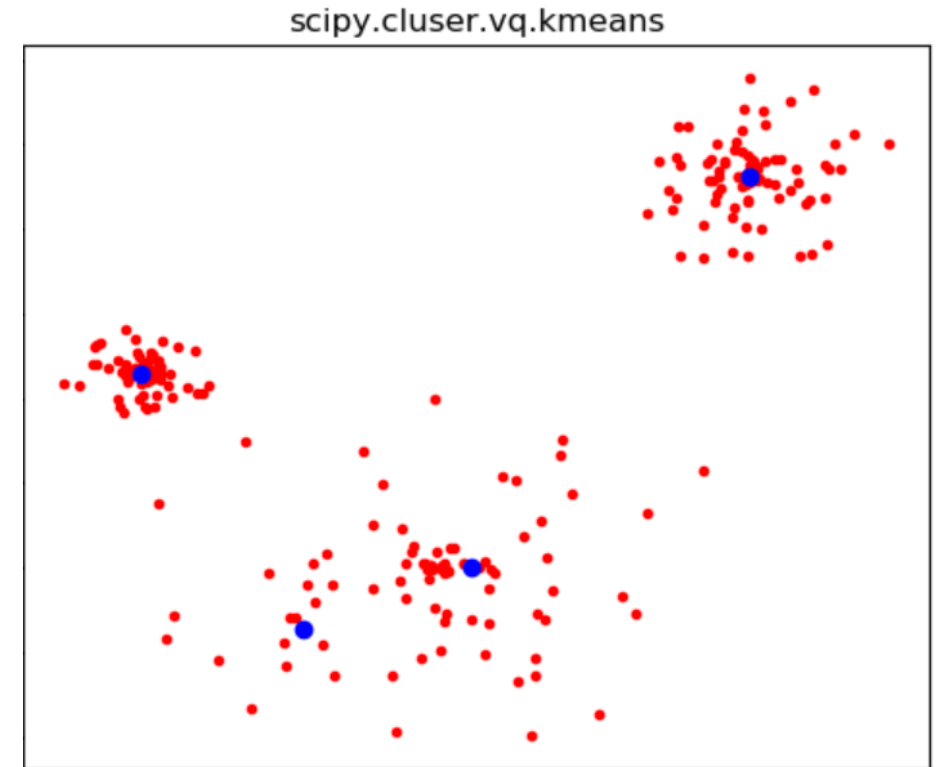
k_means.py

```
from scipy.cluster.vq import kmeans, whiten
import matplotlib.pyplot as plt

points = whiten(points) # normalize variance of points

plt.plot(*zip(*points), 'r.')
plt.plot(*zip(*kmeans(points, K)[0]), "bo")
plt.title("scipy.cluster.vq.kmeans")
```

Note: According to the documentation "whiten must be called prior to passing an observation matrix to kmeans"



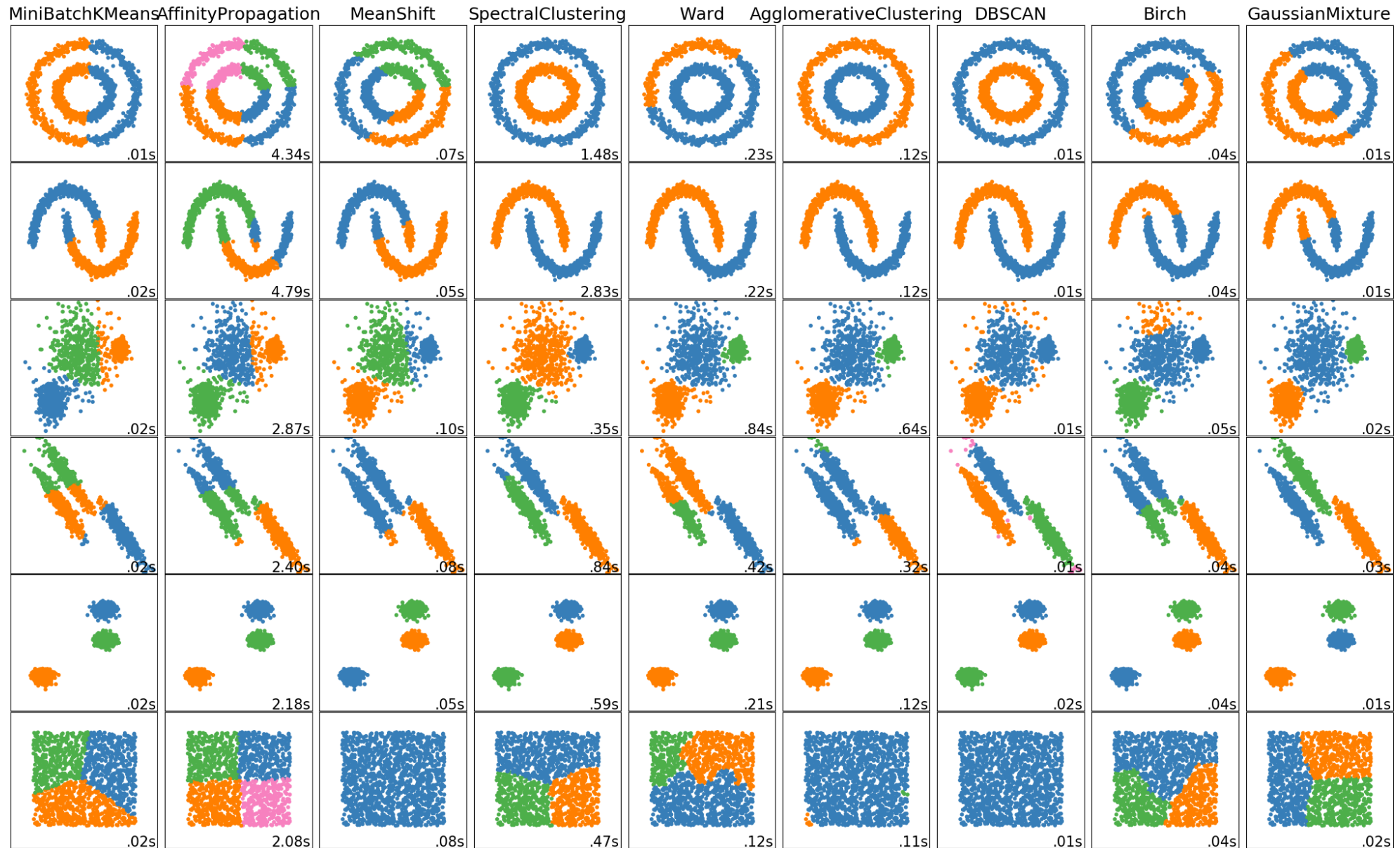
scipy.cluster.vq.whiten

- Normalizes / scales each dimension to have unit variance 1.0

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Other Python clustering methods - `sklearn.cluster`



Data Mining Algorithms

- k-means, and more generally clustering, is just one field in the area of *Data Mining*
- For more information see the webpage [Top 10 Data Mining Algorithms, Explained](#) a follow up to the below paper
- X. Wu et al., *Top 10 algorithms in data mining*, Knowledge and Information Systems, 14(1):1–37, 2008. DOI [10.1007/s10115-007-0114-2](https://doi.org/10.1007/s10115-007-0114-2)