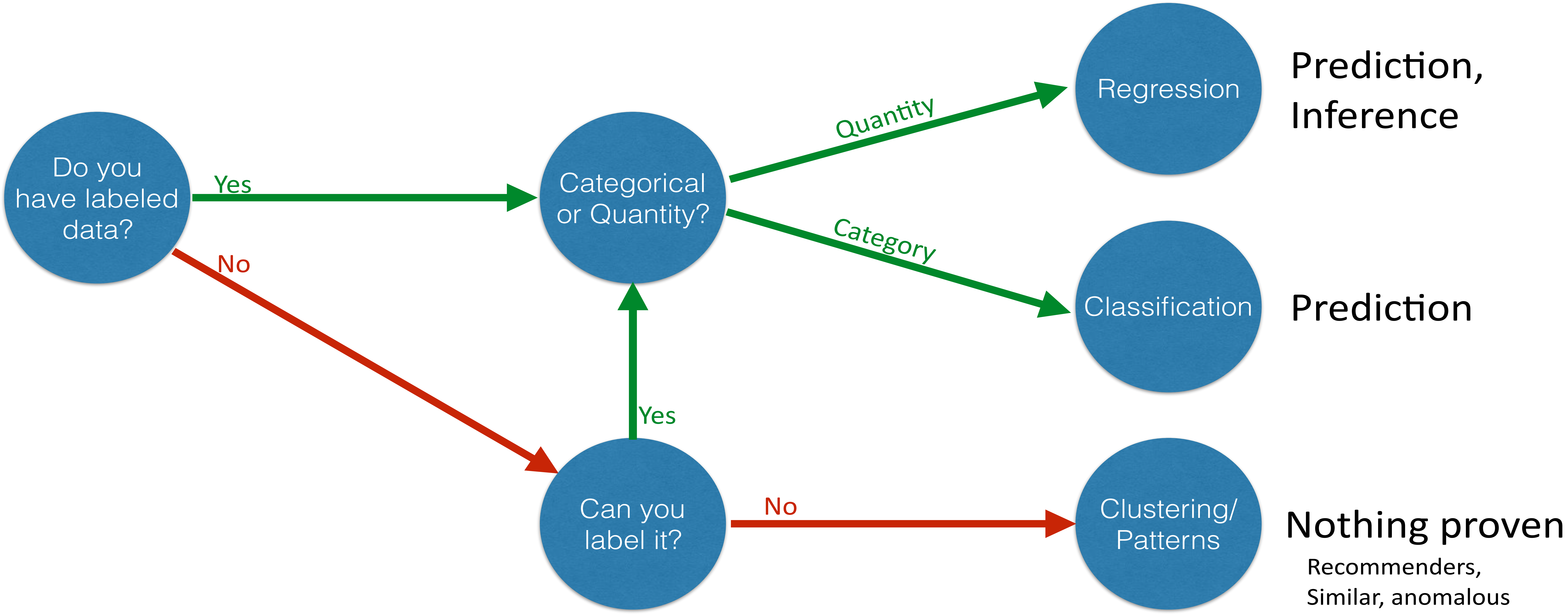
Module 6: Unsupervised Learning: Clustering

Agenda for Today

- Measuring Distances
- Math free overview of clustering techniques



Do you have labeled data?

Yes

No

Categorical or Quantity?

Can you label it?

Yes

No

Quantity

Category

Regression

Classification

Clustering/ Patterns

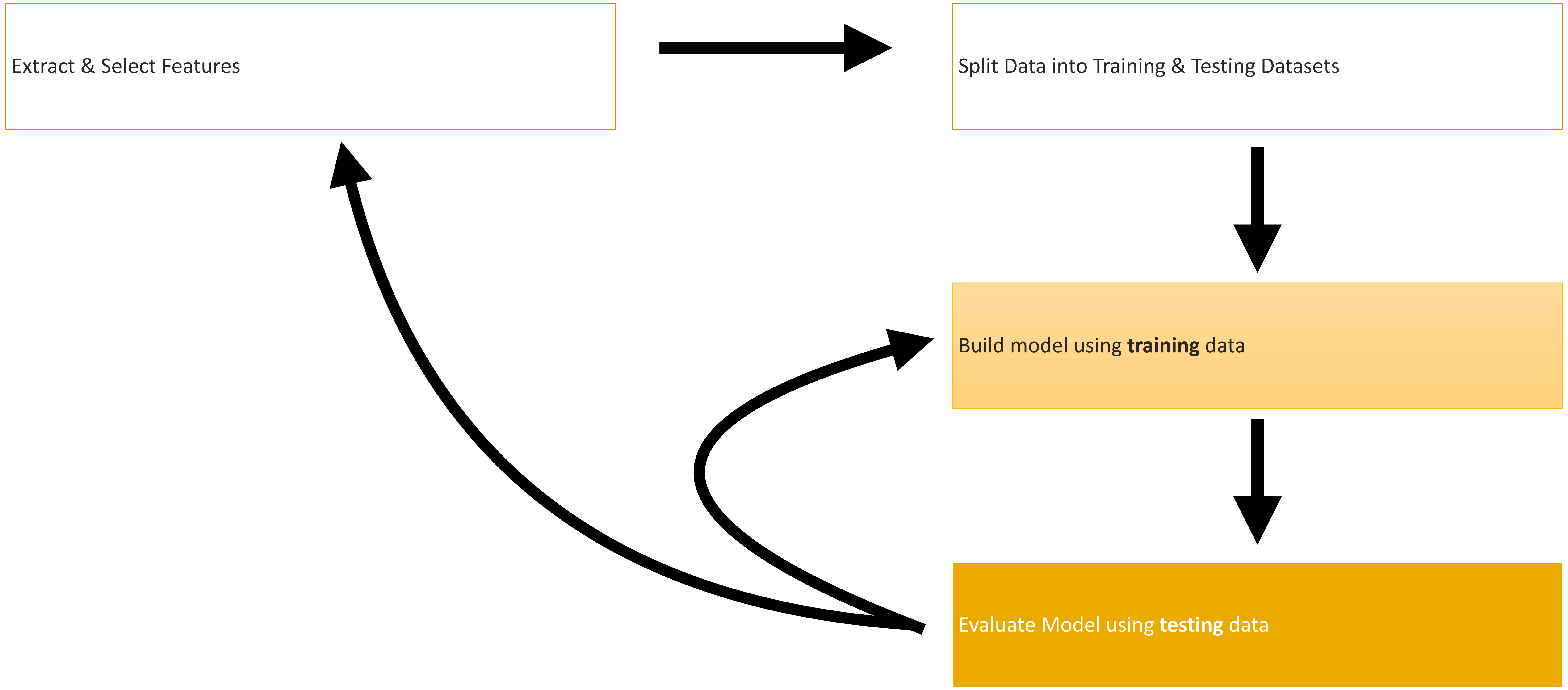
Prediction, Inference

Prediction

Nothing proven

Recommenders, Similar, anomalous

Supervised ML Process



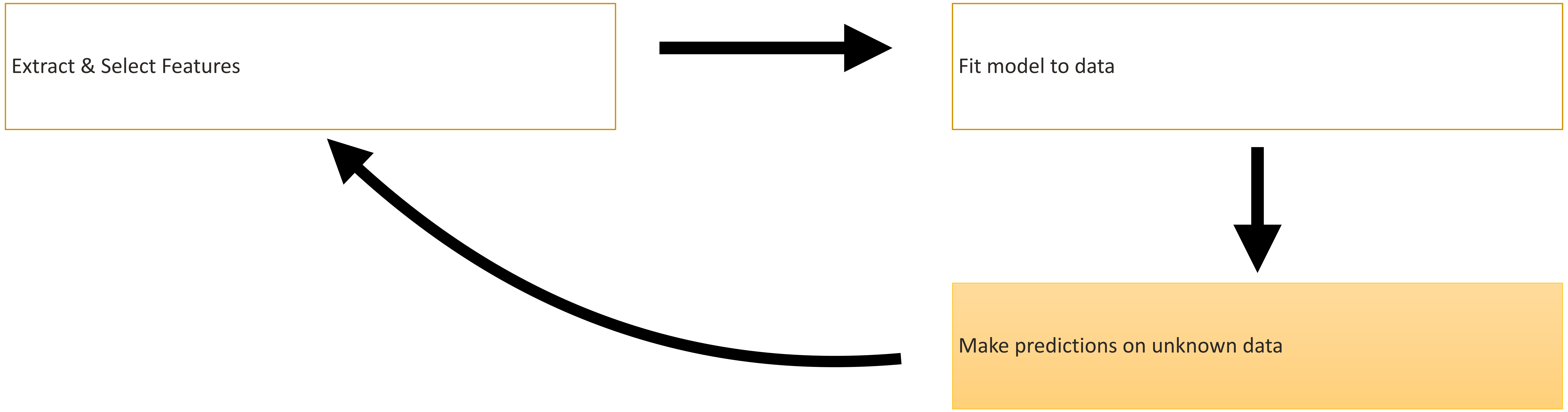
Extract & Select Features

Split Data into Training & Testing Datasets

Build model using **training** data

Evaluate Model using **testing** data

Unsupervised ML Process



Extract & Select Features

Fit model to data

Make predictions on unknown data

Unsupervised Clustering Algorithm

1. Select Features
2. Calculate a distance measure
3. Apply a clustering algorithm
4. Validate?

Which Departments are Similar?

	Malware events
Dept1	6
Dept2	1
Dept3	8

Which Departments are Similar?

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

Which Departments are Similar?

	Malware events	Phishing	Open Tickets
Dept1	6	6	3
Dept2	1	2	1
Dept3	8	1	9

Computing Distance

	Malware events
Dept1	6
Dept2	1
Dept3	8

Compare:
Dept1 to Dept2: | 6 - 1 | = 5
Dept2 to Dept3: | 1 - 8 | = 7
Dept1 to Dept3: | 6 - 8 | = 2

Two-Dimensional Distance

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

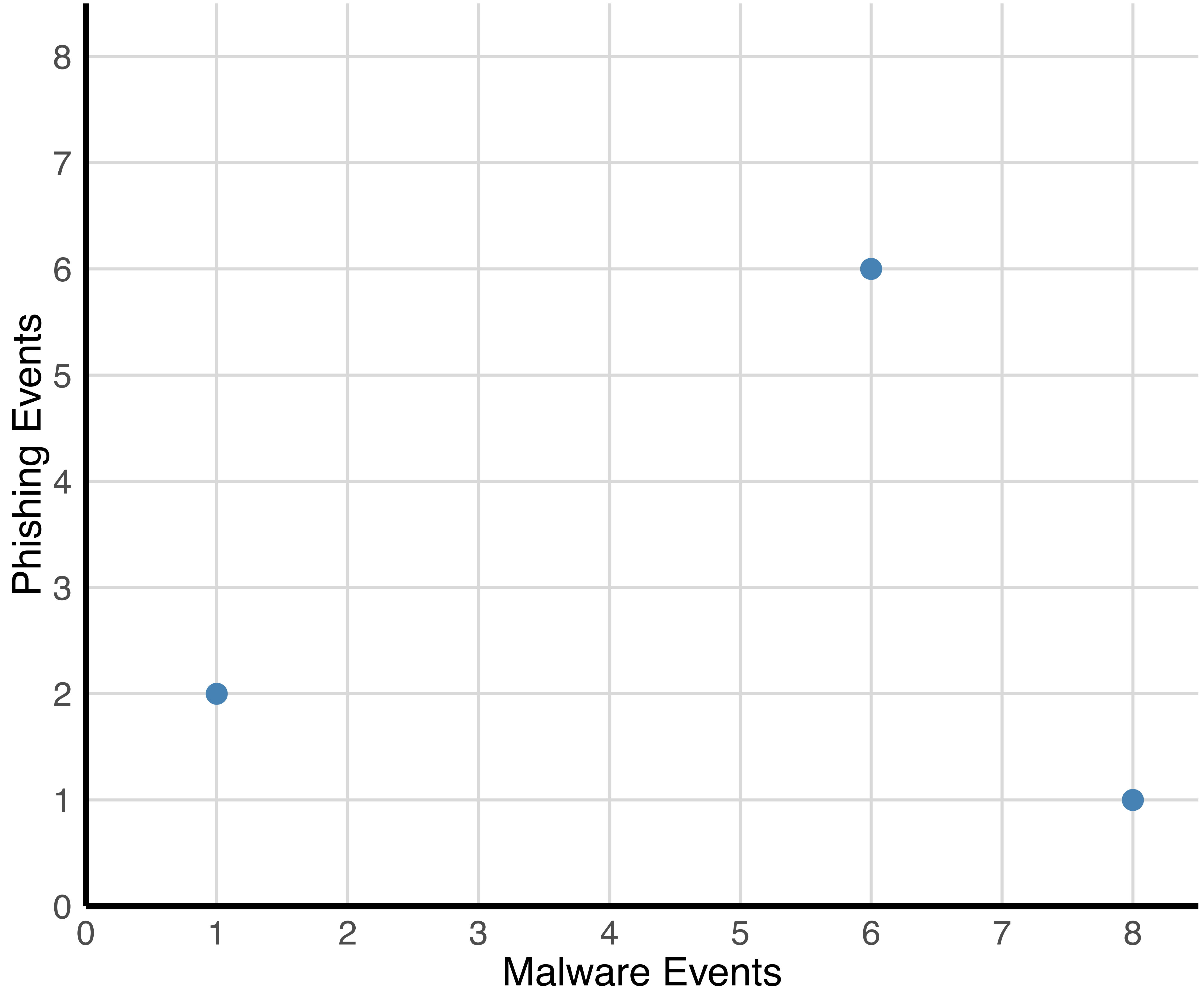
Multiple Distance methods

- Euclidean
- Manhattan
- Maximum
- Canberra
- Binary
- Minkowski

… (to name a few)

Two-Dimensional Distance

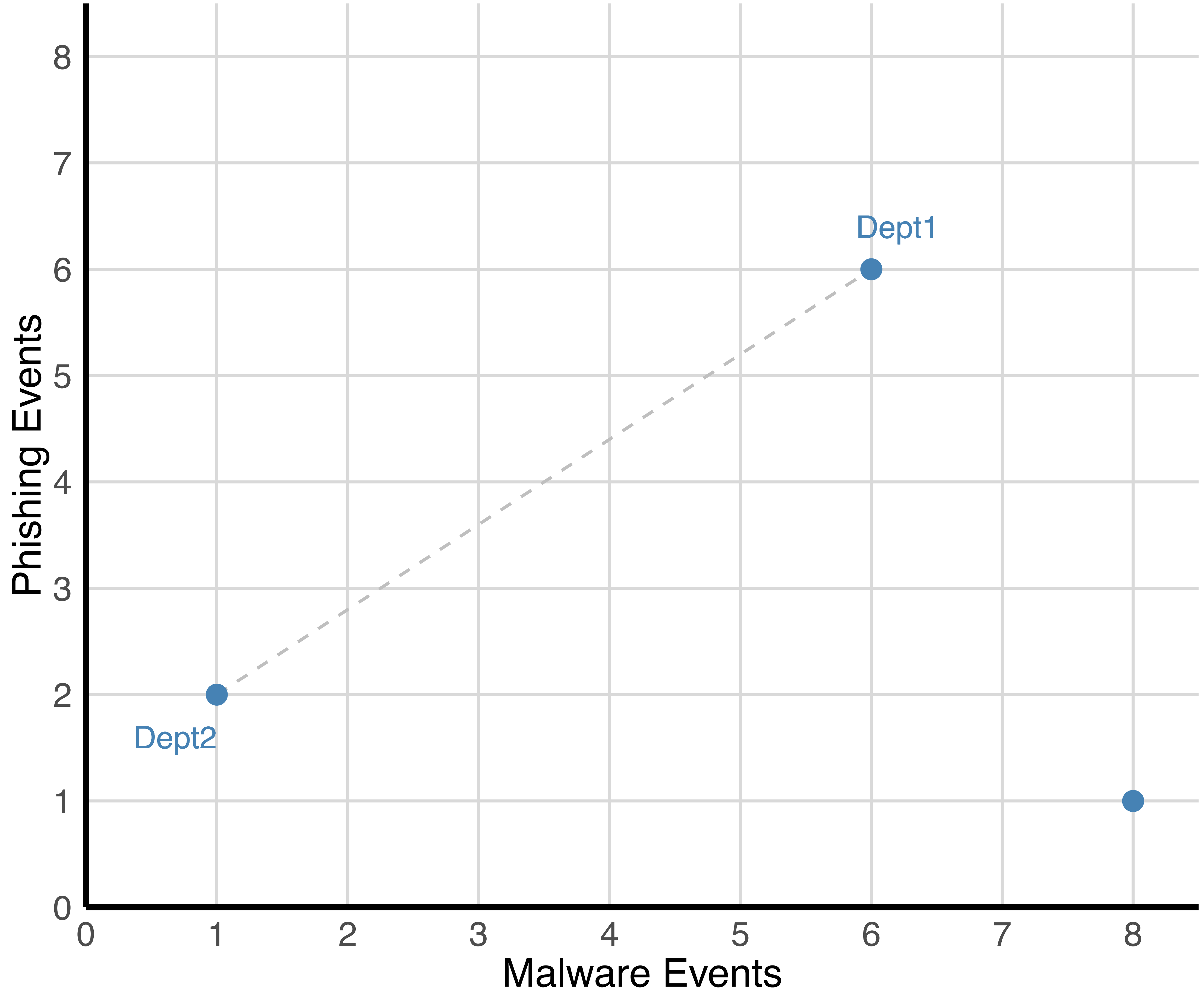
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Euclidean very common and easy to understand

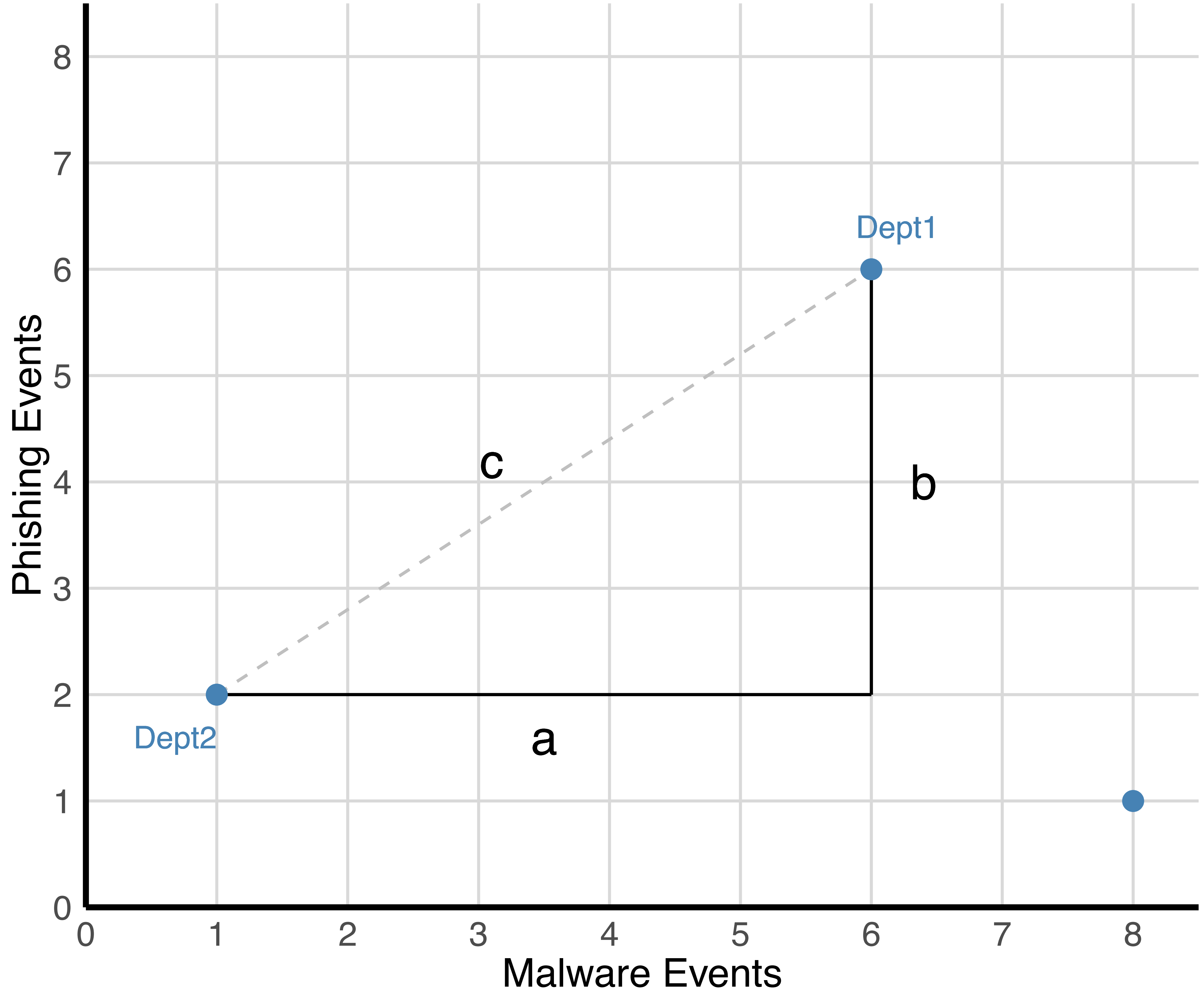
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Euclidean very common and easy to understand: $a^2 + b^2 = c^2$

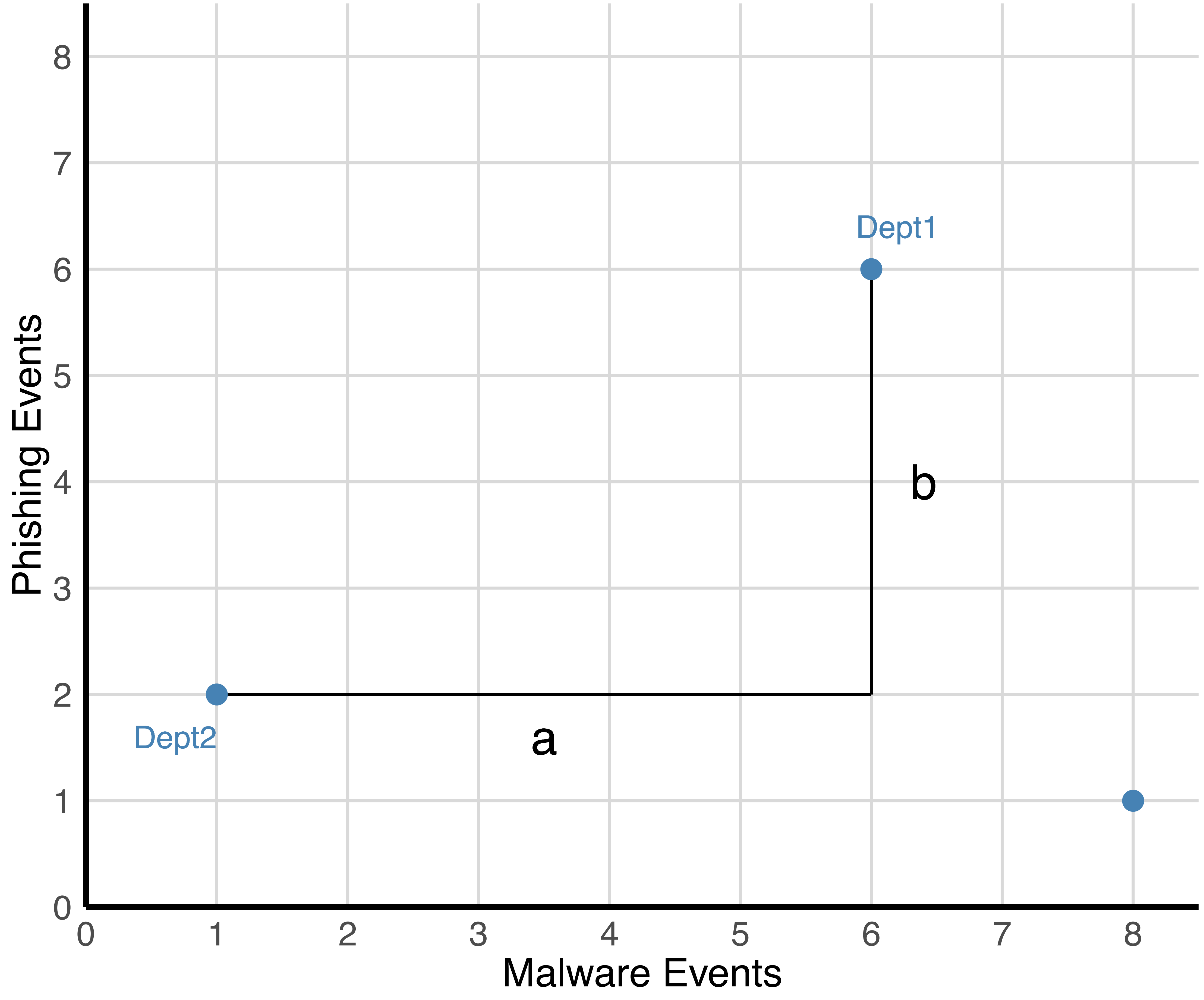
	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Two-Dimensional Distance

Manhattan also easy to comprehend: a + b

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1



Computing Distance

	Malware events	Phishing
Dept1	6	6
Dept2	1	2
Dept3	8	1

Compare:

Dept1 to Dept2: sqrt((6-1)^2 + (6-2)^2) = **6.4**

Dept2 to Dept3: … = **7.1**

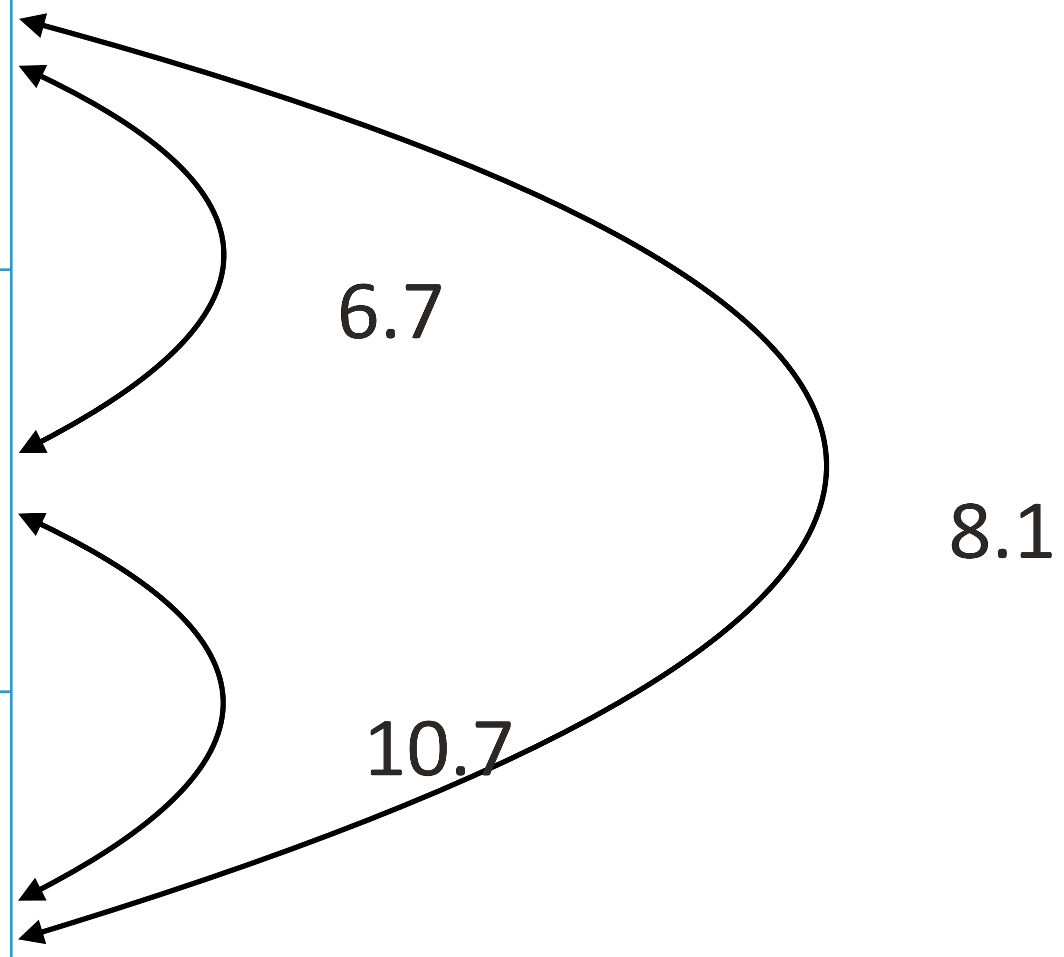
Dept1 to Dept3: … = **5.4**

Euclidean Distance calculations

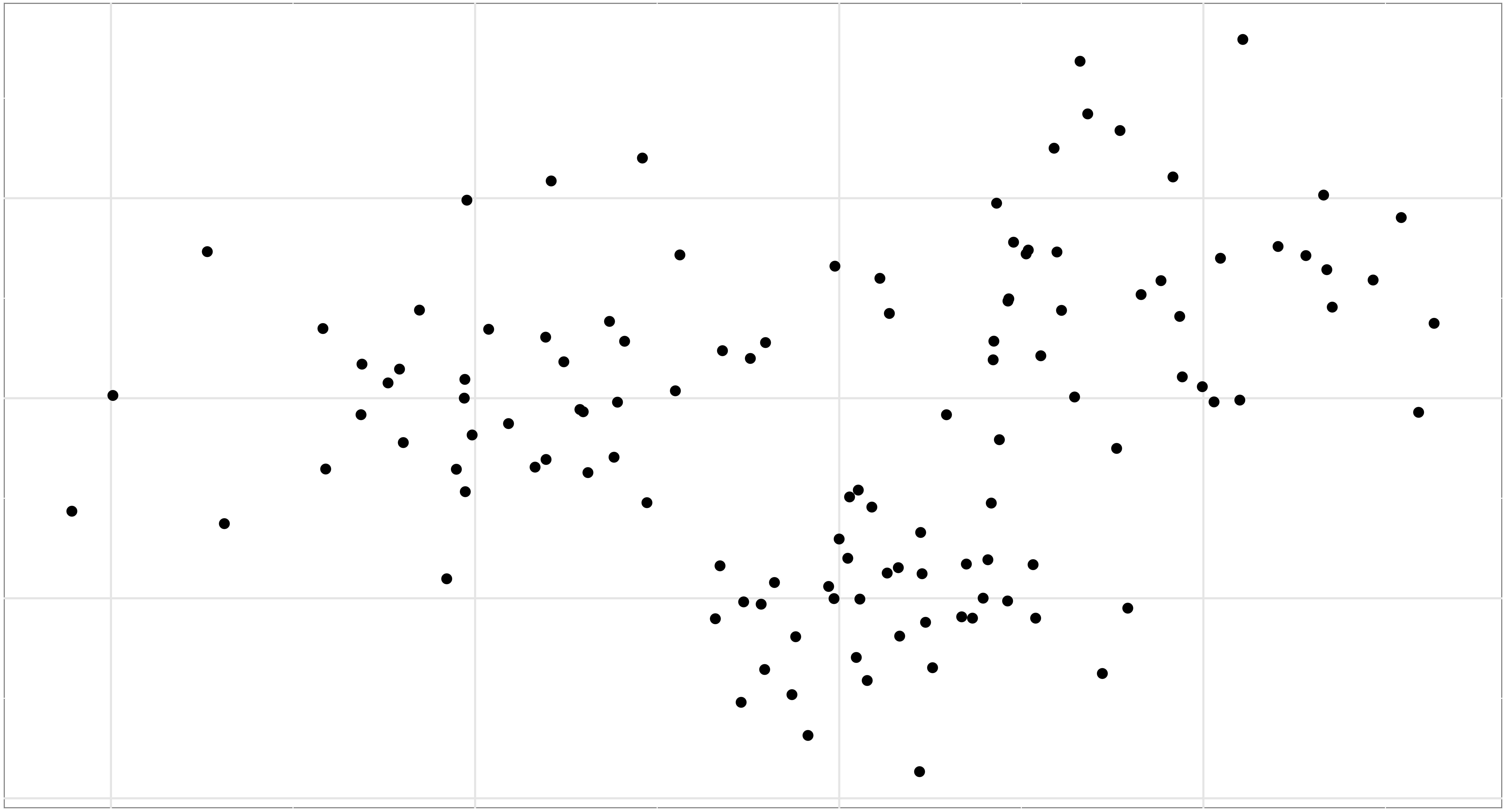
```
def dist(x,y):
    return np.sqrt(np.sum((x-y)**2))

> mat = np.array([[ 6,6,3 ], [1,2,1], [8,1,9]])
> dist(mat[0], mat[1])
6.708203932499369

> dist(mat[1], mat[2])
10.677078252031311

> dist(mat[0], mat[2])
8.06225774829855
```

Which Departments are Similar?

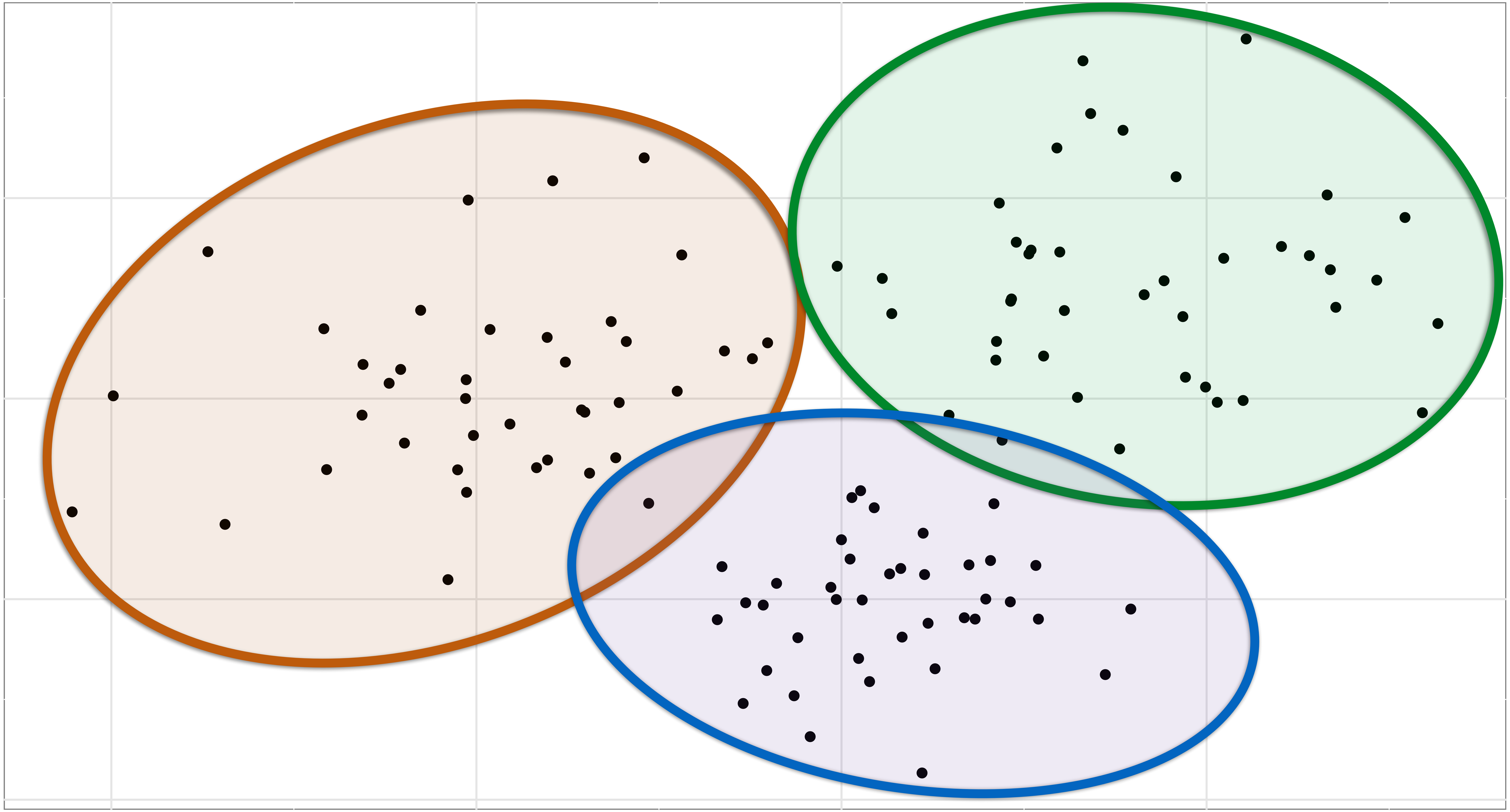
	Malware events	Phishing	Open Tickets
Dept1	6	6	3
Dept2	1	2	1
Dept3	8	1	9



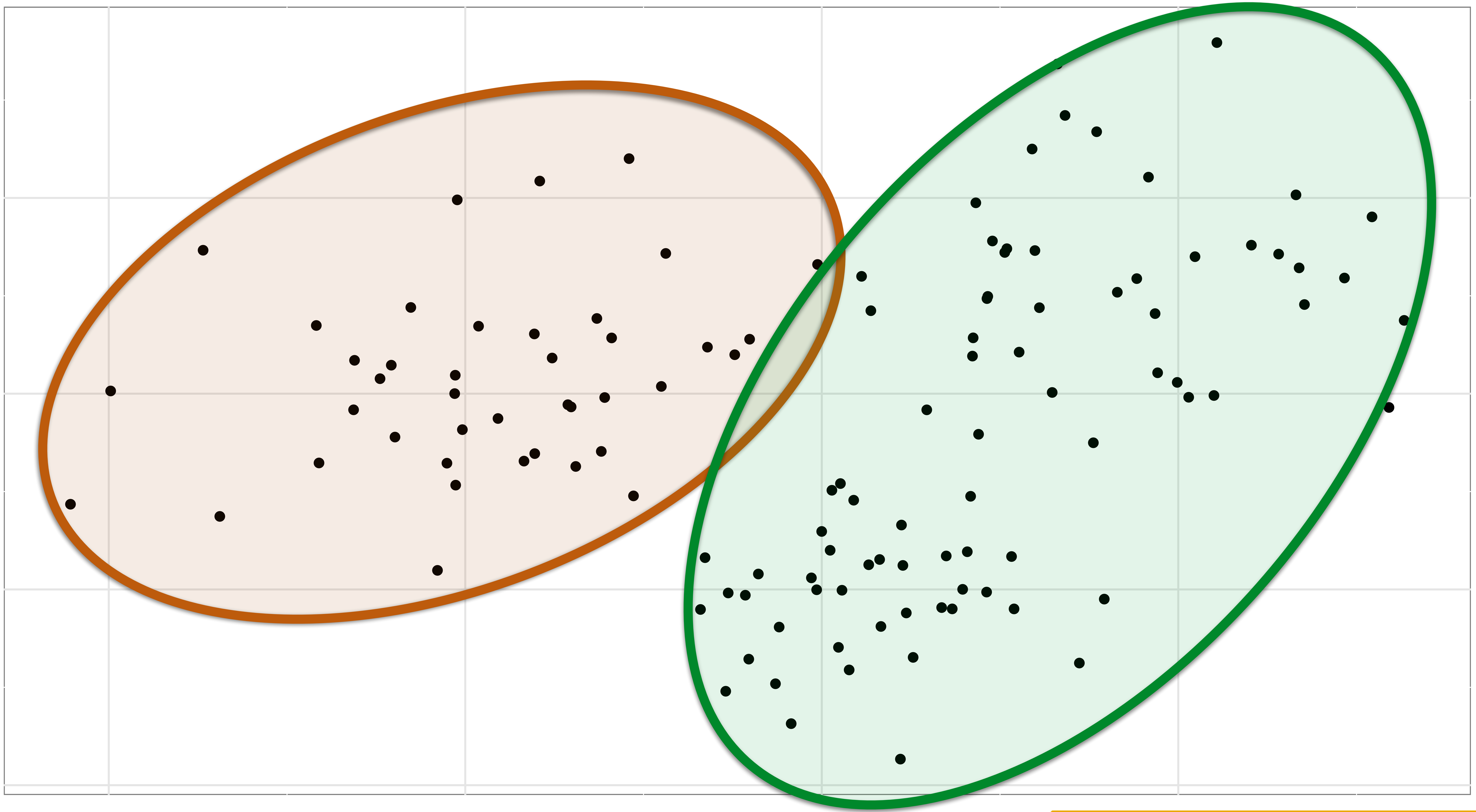
Clustering…



Clustering…



Clustering…

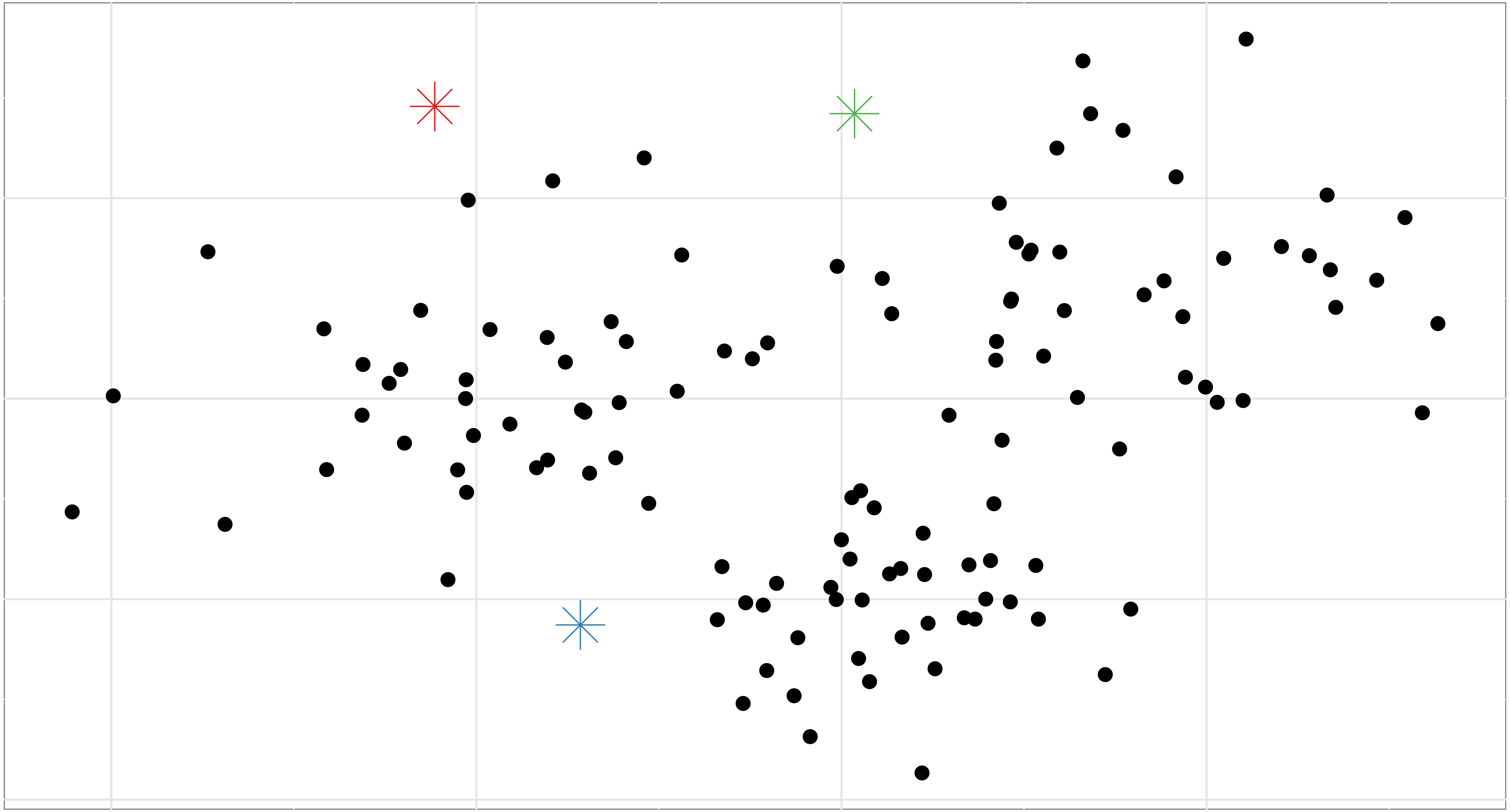


K-Means

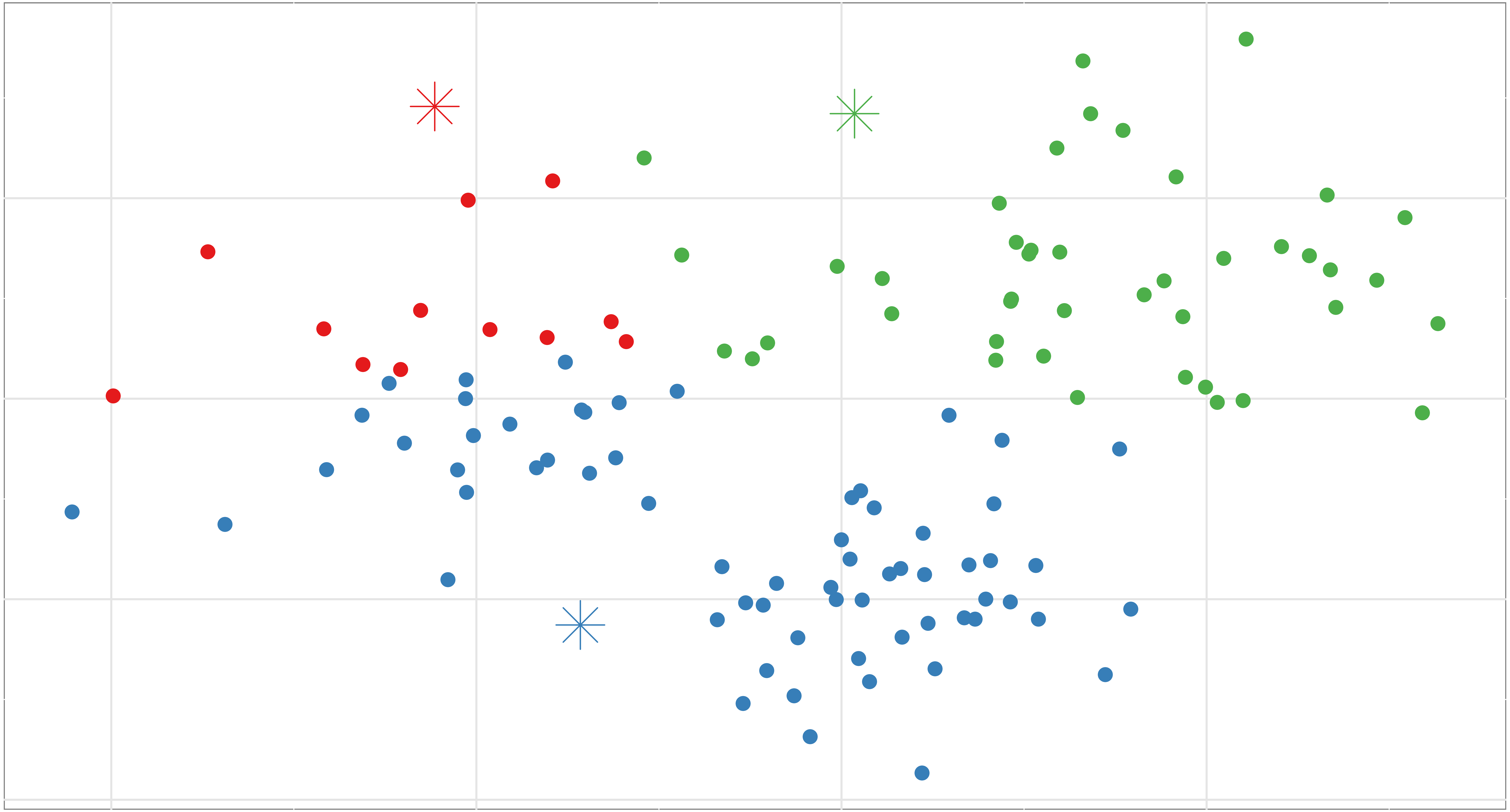
Before starting, pick the number of clusters, K

1. Pick K random centroids within data range
2. Assign each data point to the nearest centroid
3. Move centroid to center of assigned points
4. Repeat steps 2 and 3 until centroid stops shifting

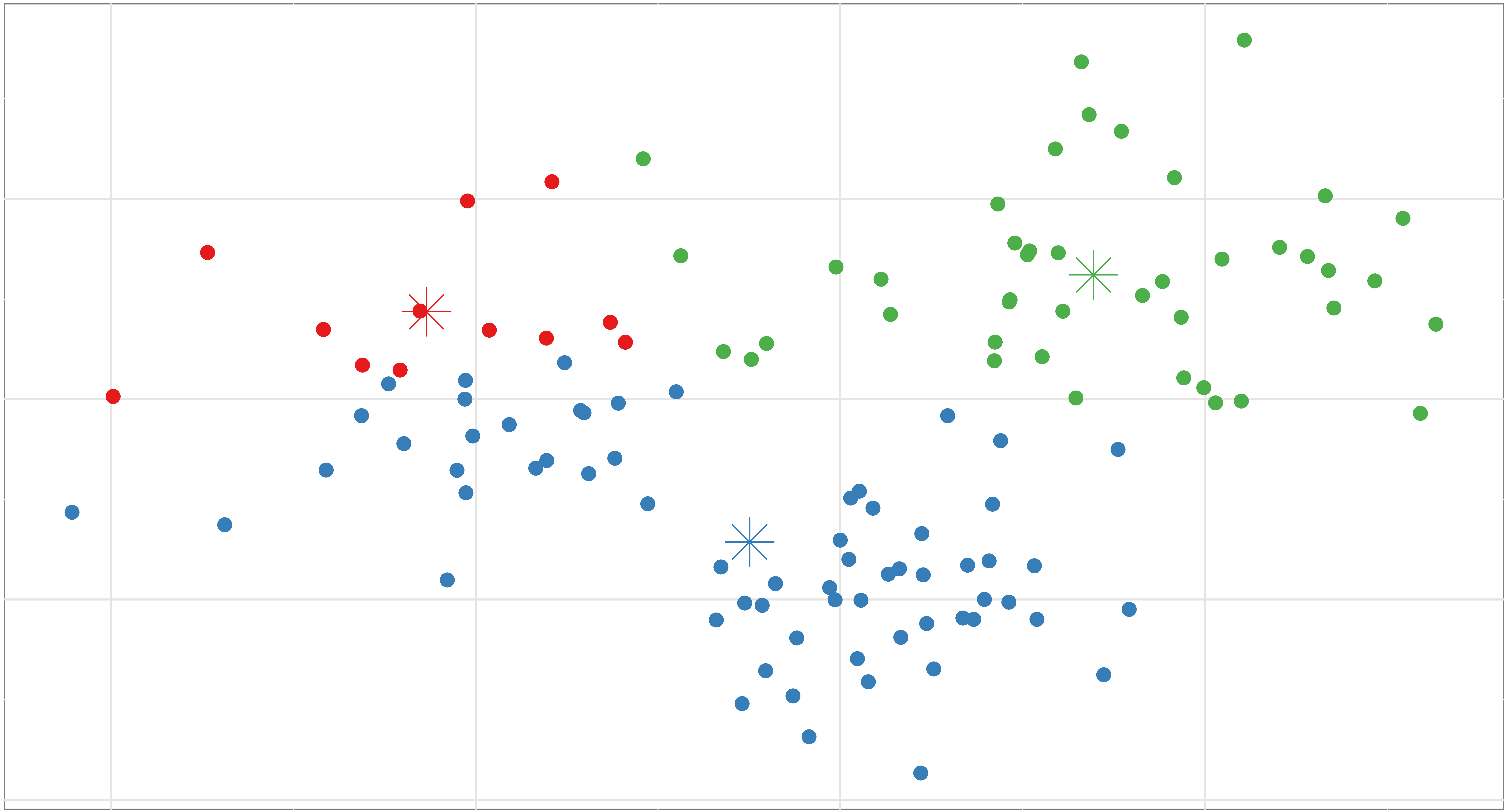
Step 1: Pick 3 random centroids within data range



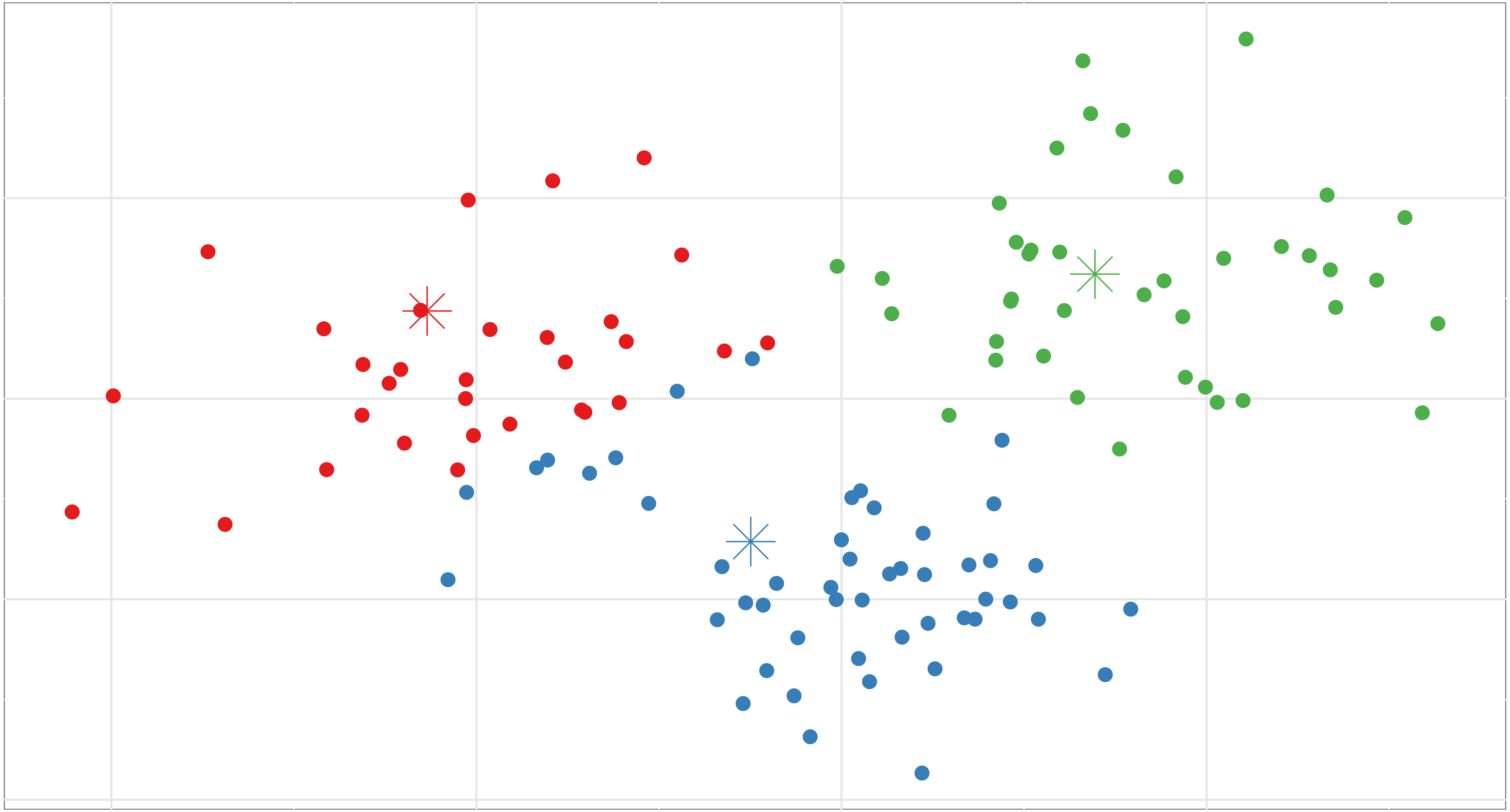
Step 2: Assign each data point to the nearest centroid (1)



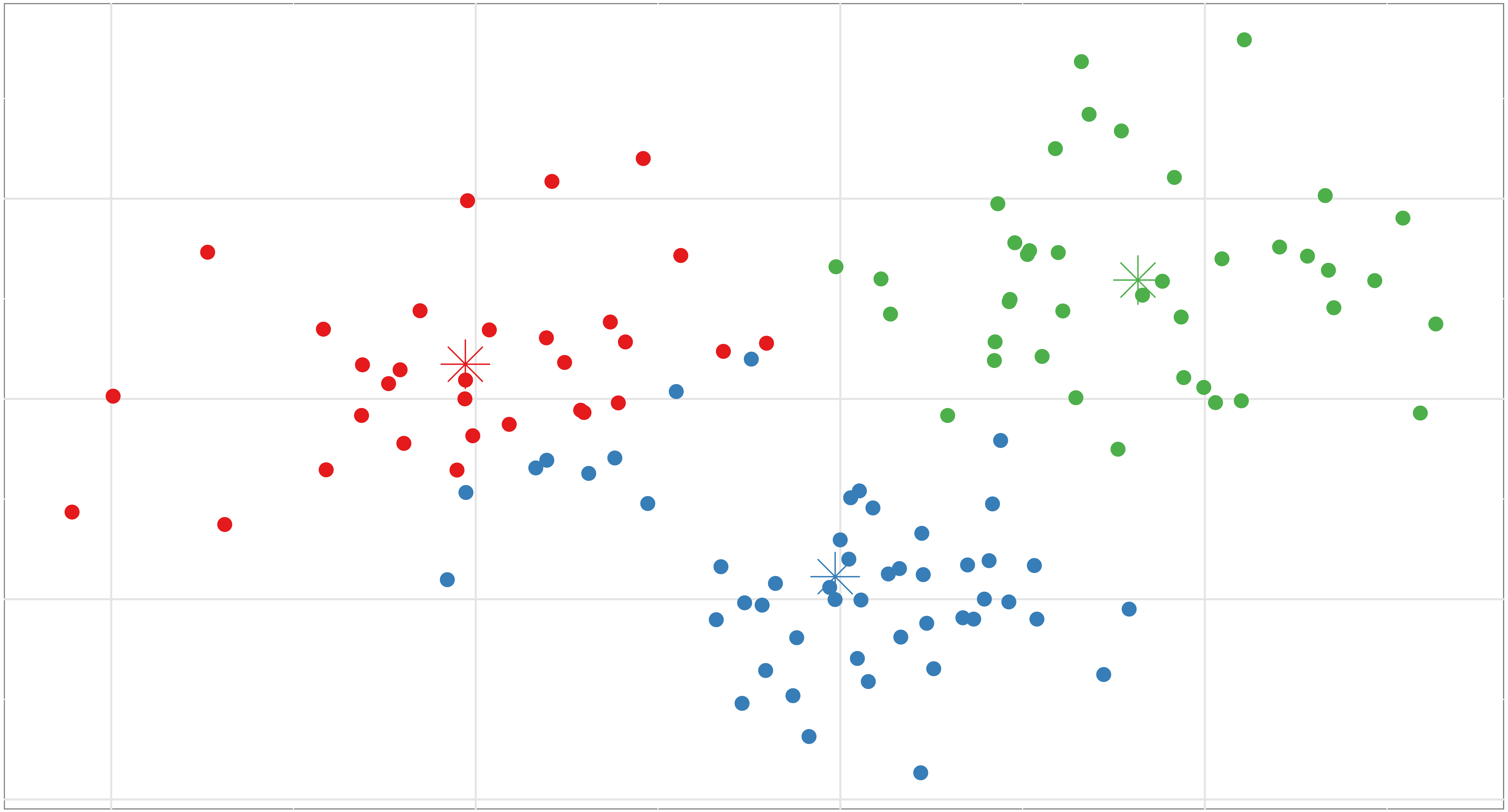
Step 3: Move centroid to center of assigned points (1)



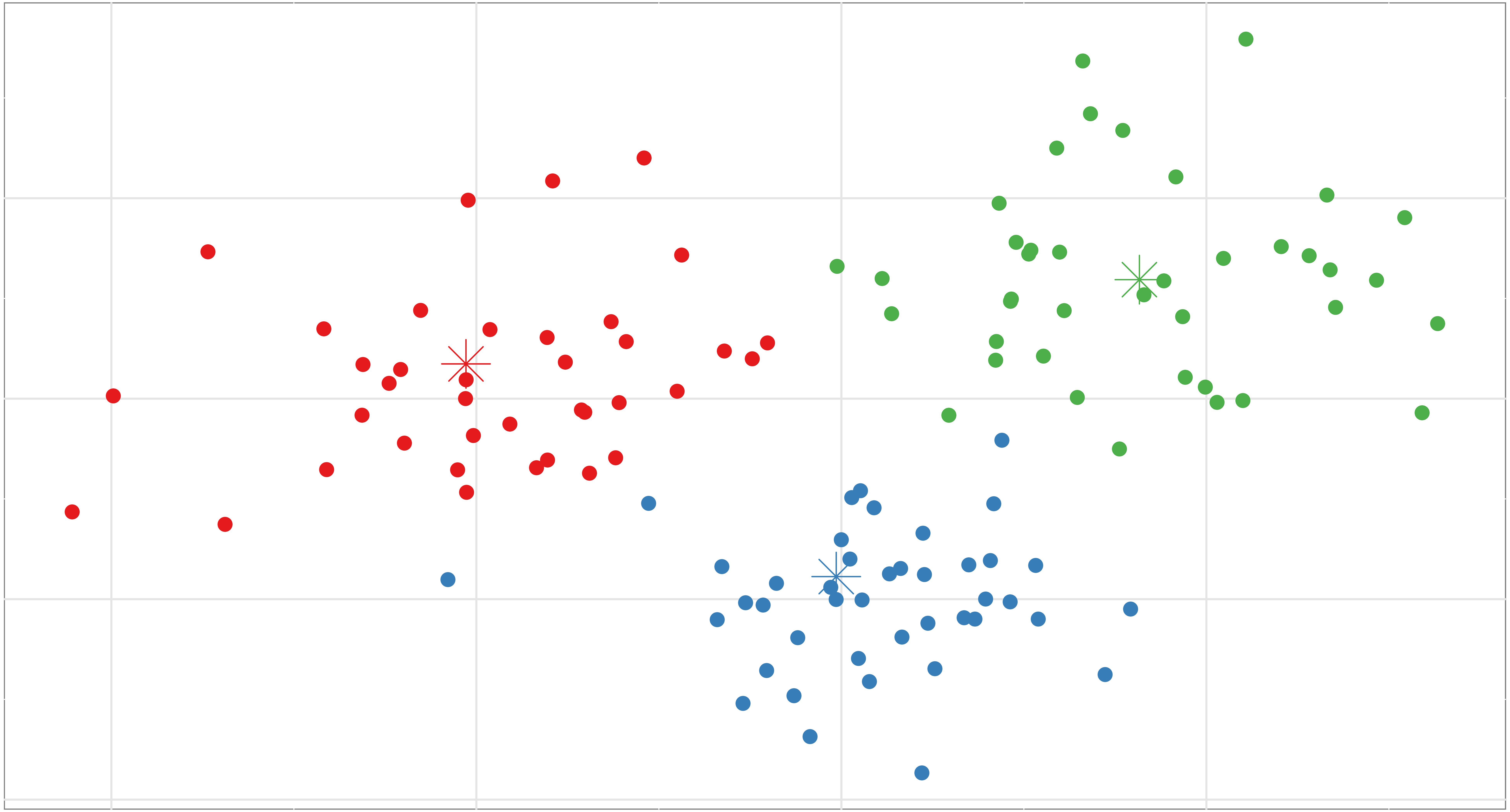
Step 2: Assign each data point to the nearest centroid (2)



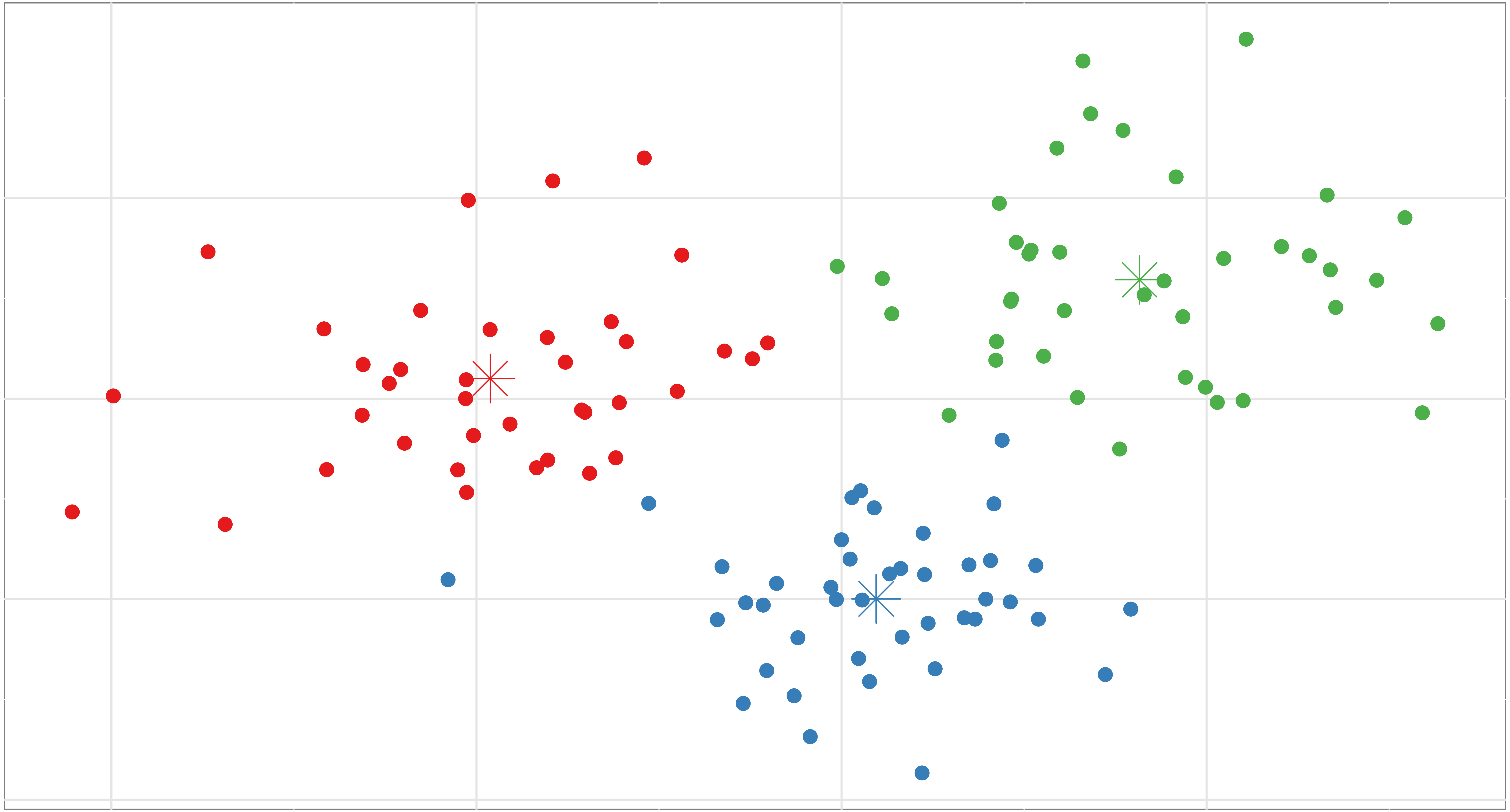
Step 3: Move centroid to center of assigned points (2)



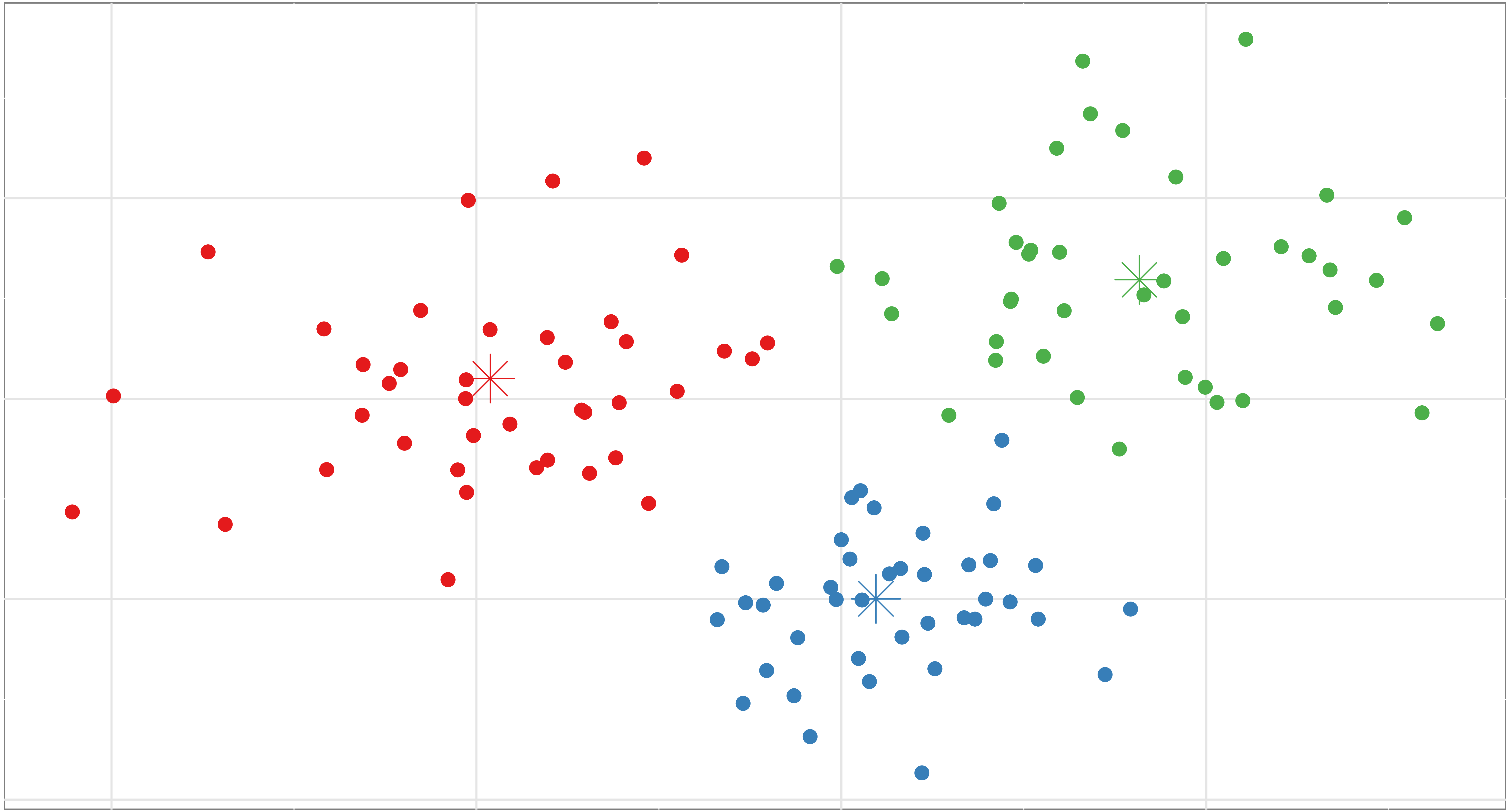
Step 2: Assign each data point to the nearest centroid (3)



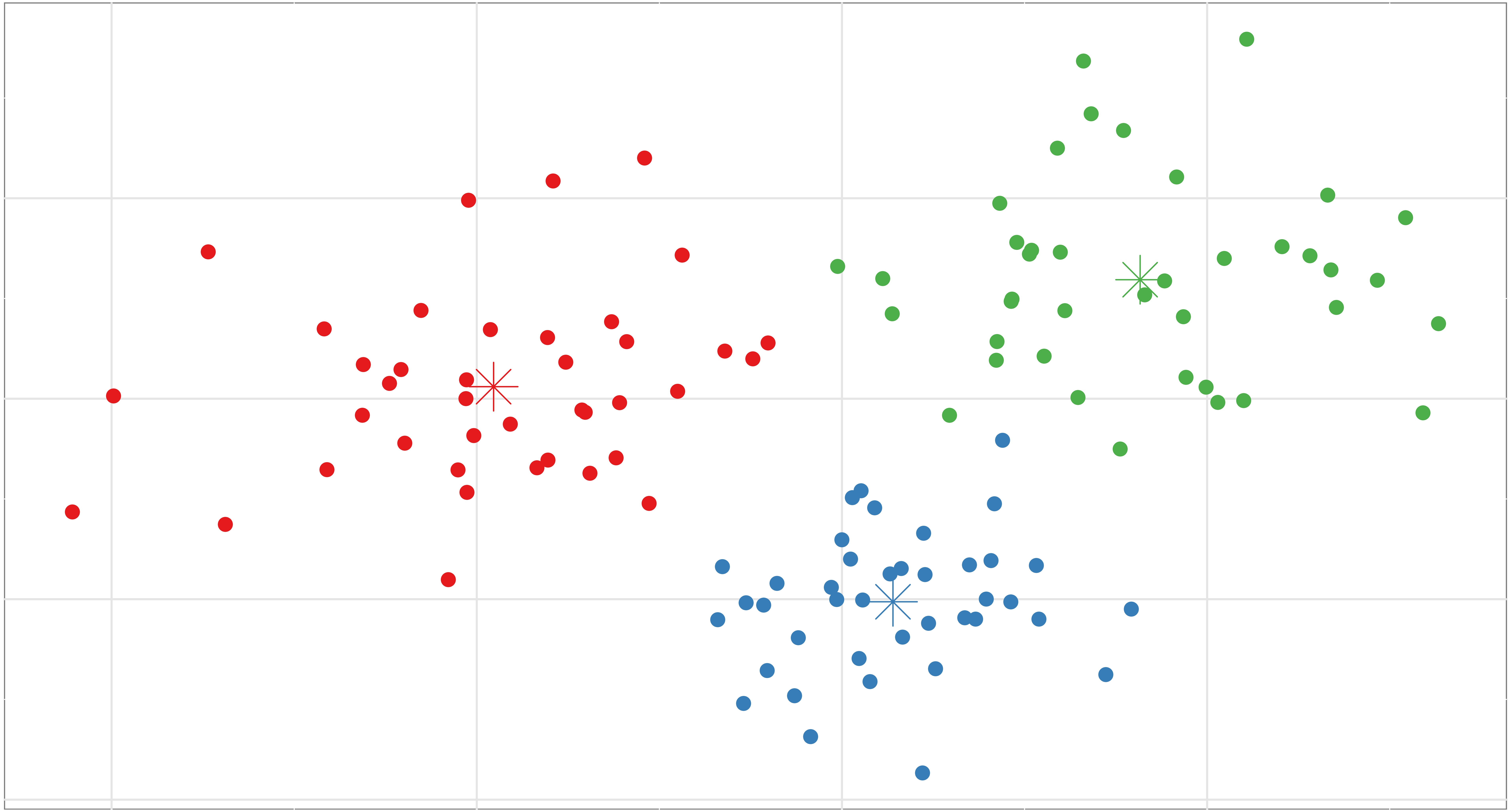
Step 3: Move centroid to center of assigned points (3)



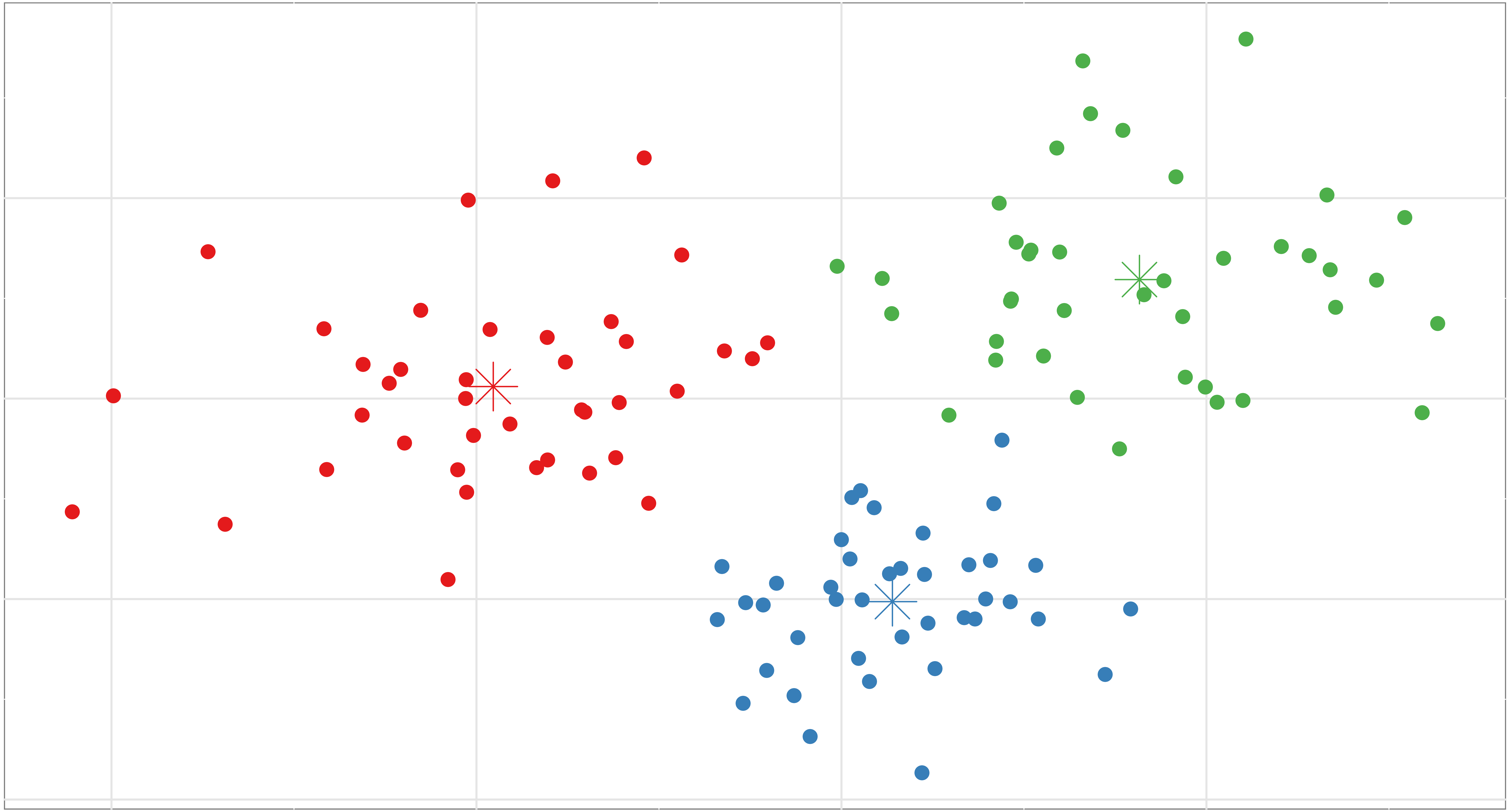
Step 2: Assign each data point to the nearest centroid (4)



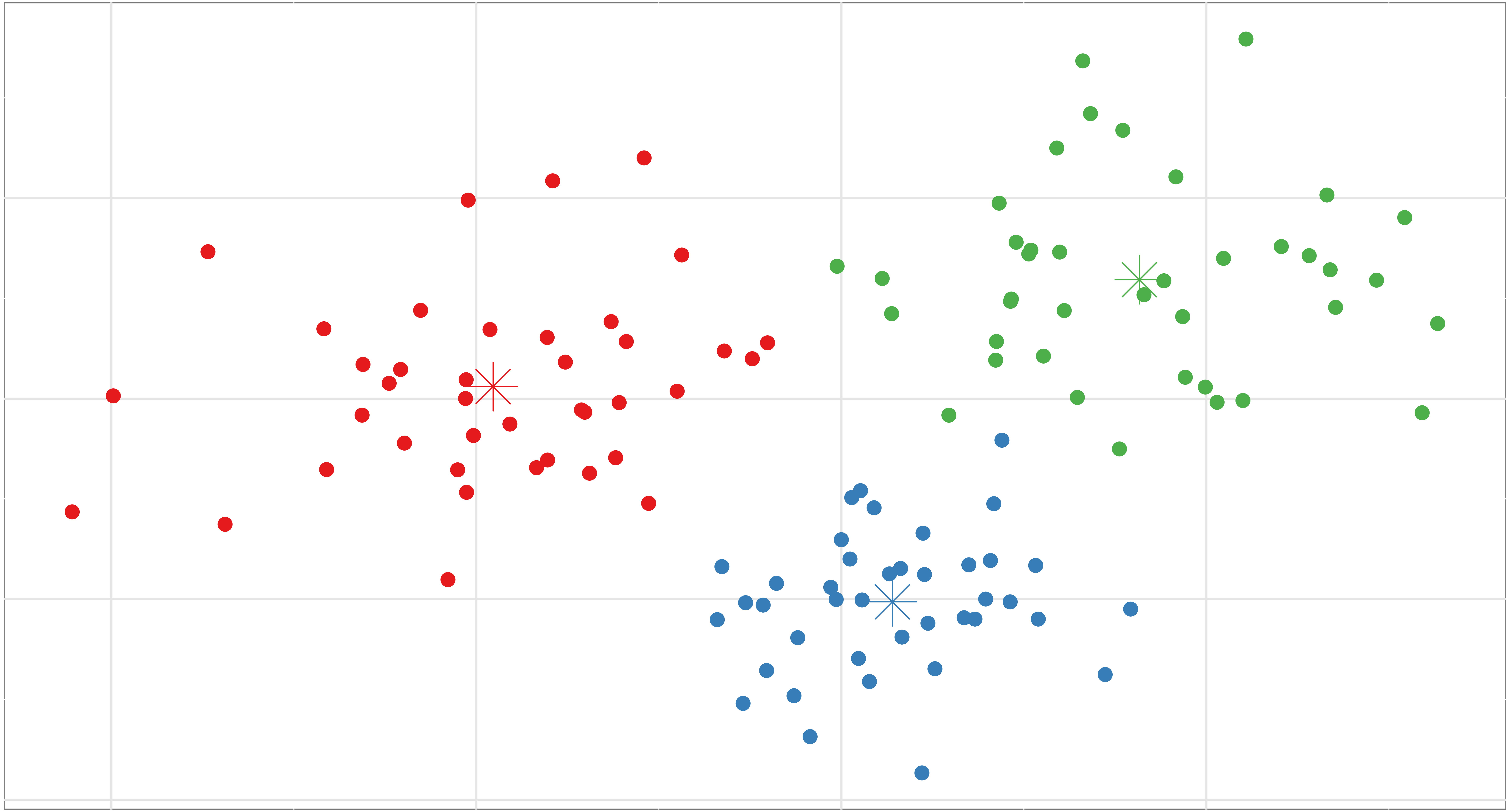
Step 3: Move centroid to center of assigned points (4)



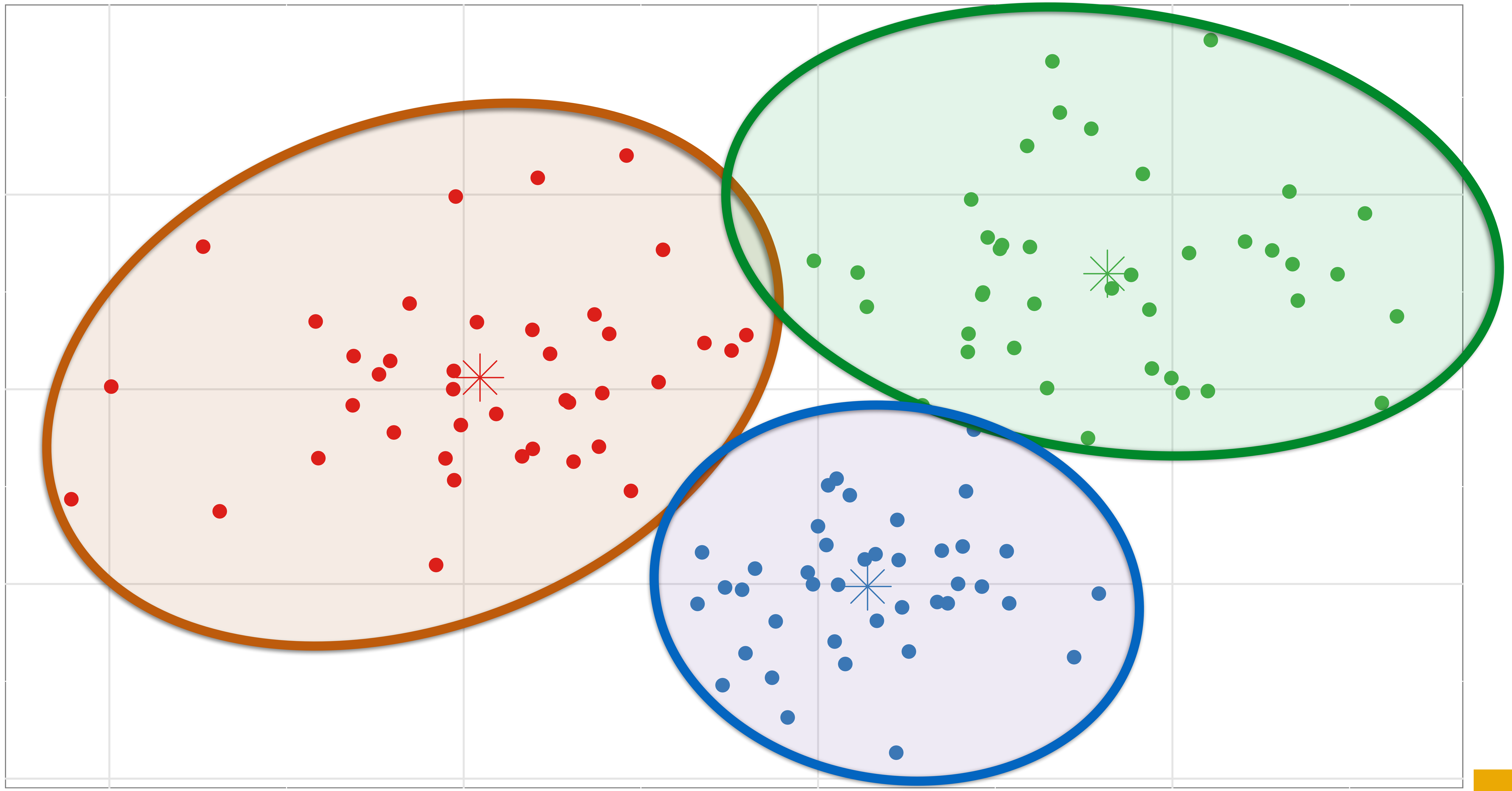
Step 2: Assign each data point to the nearest centroid (5)



Step 3: Move centroid to center of assigned points (5)



Step 3: Move centroid to center of assigned points (5)



K-Means: Got a problem with it?

Before starting, pick the number of clusters, K

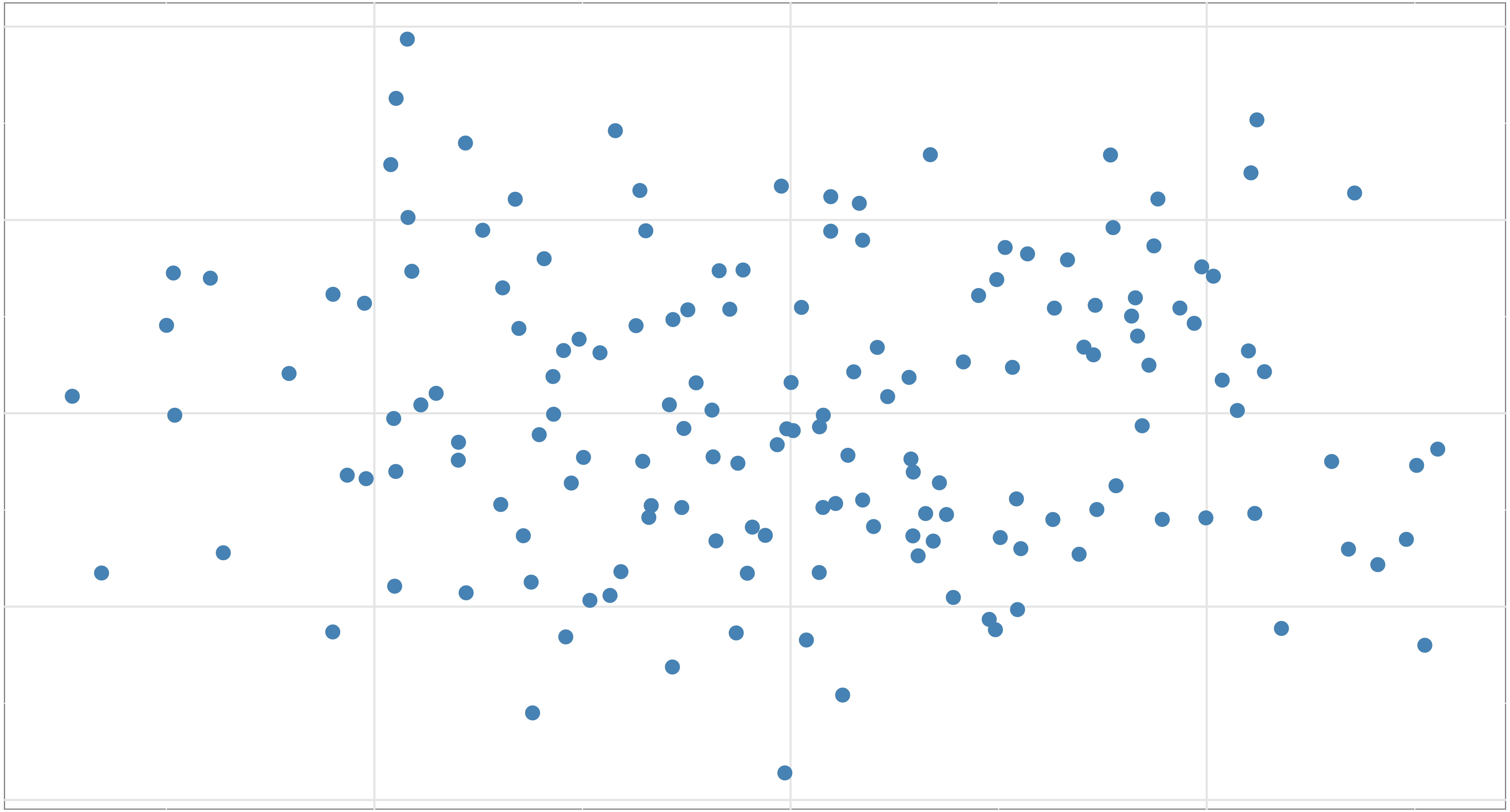
1. Pick K random centroids within data range
2. Assign each data point to the nearest centroid
3. Move centroid to center of assigned points
4. Repeat steps 2 and 3 until centroid stops shifting

K-Means: Got a problem with it?

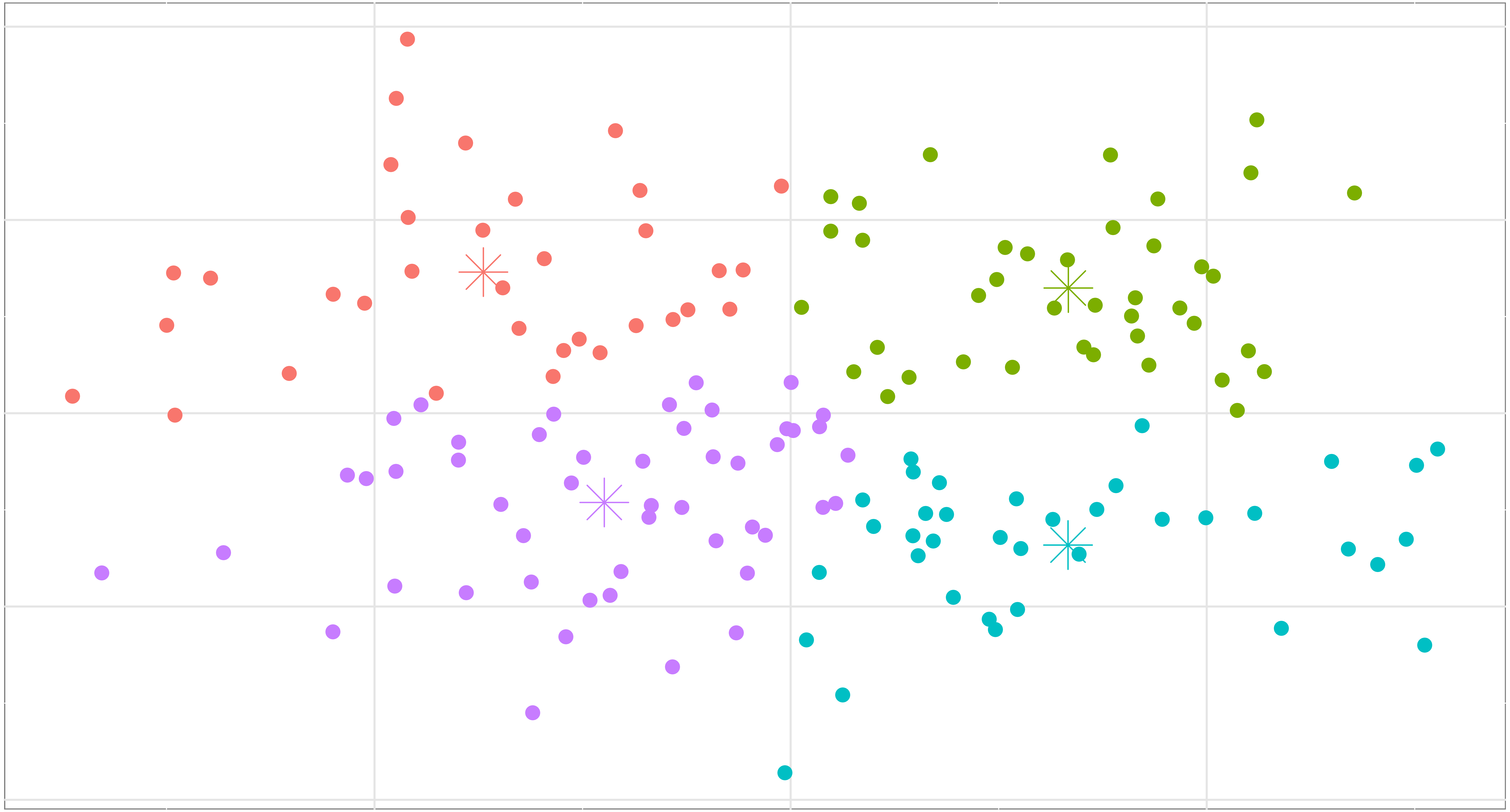
Before starting, pick the number of clusters, K Subjective

1. Pick K random centroids within data range Not Repeatable
2. Assign each data point to the nearest centroid Sensitive to Scale
3. Move centroid to center of assigned points
4. Repeat steps 2 and 3 until centroid stops shifting

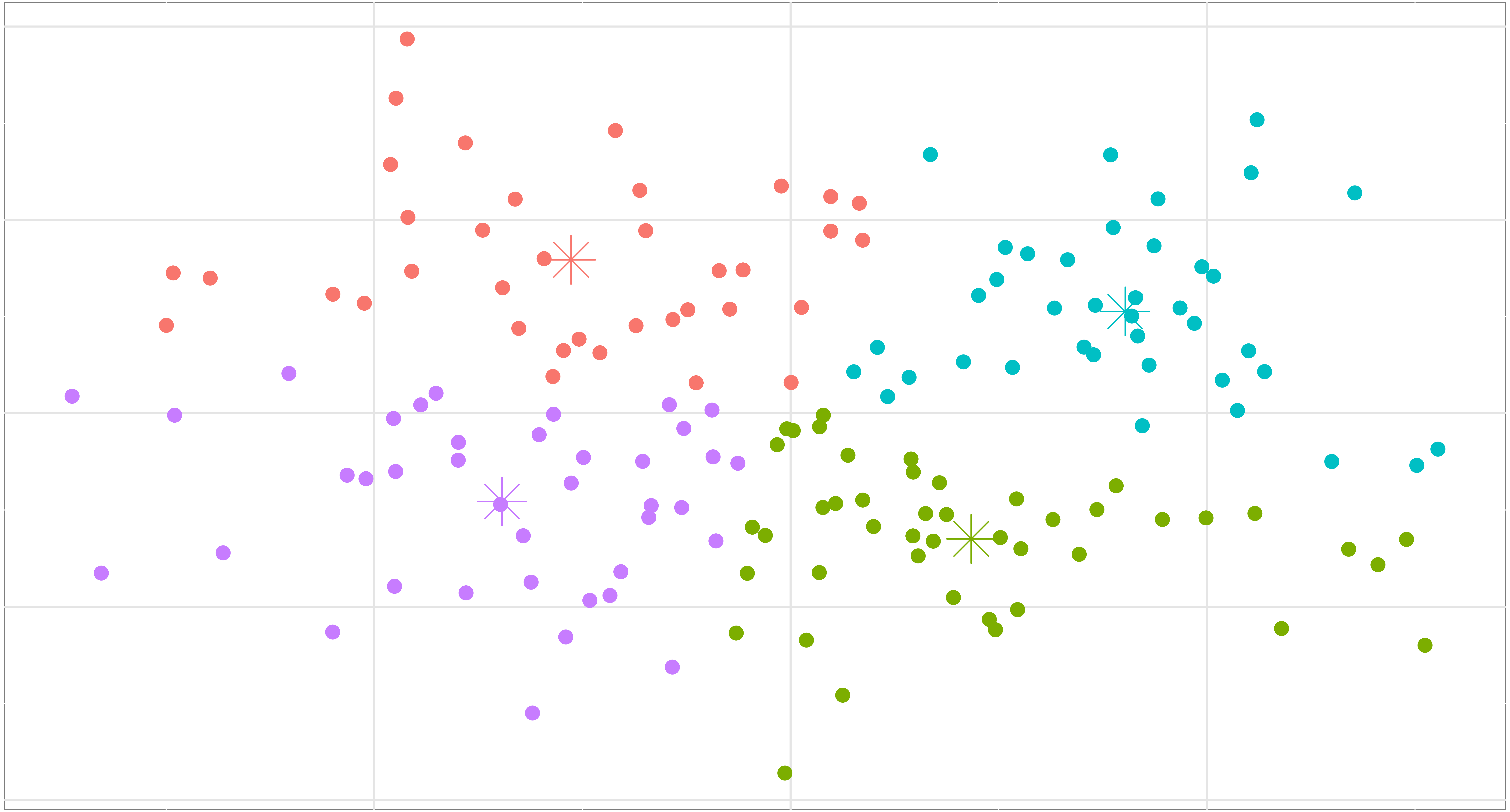
How many clusters?



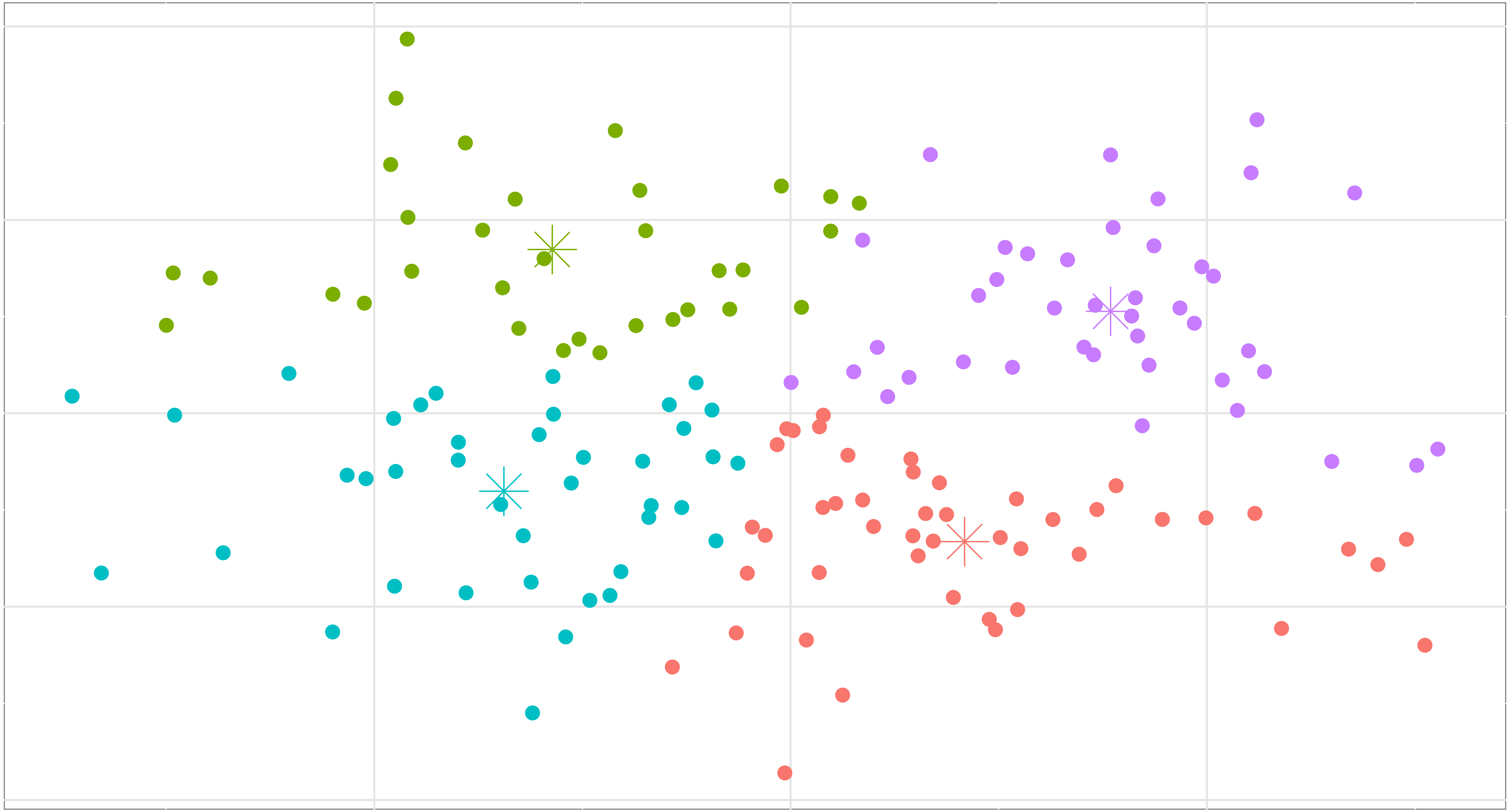
Random Start…



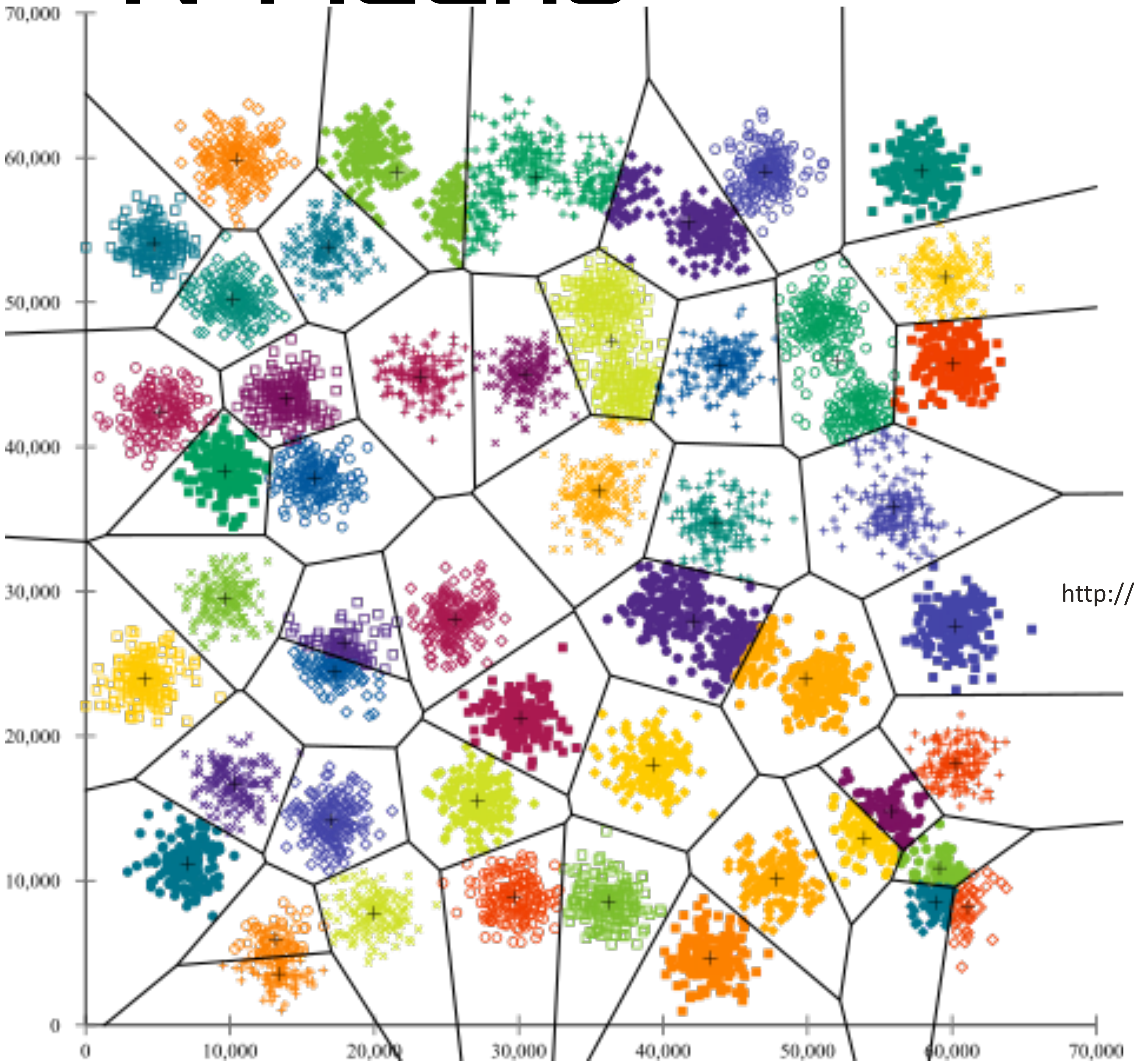
Random Start...



Random Start...



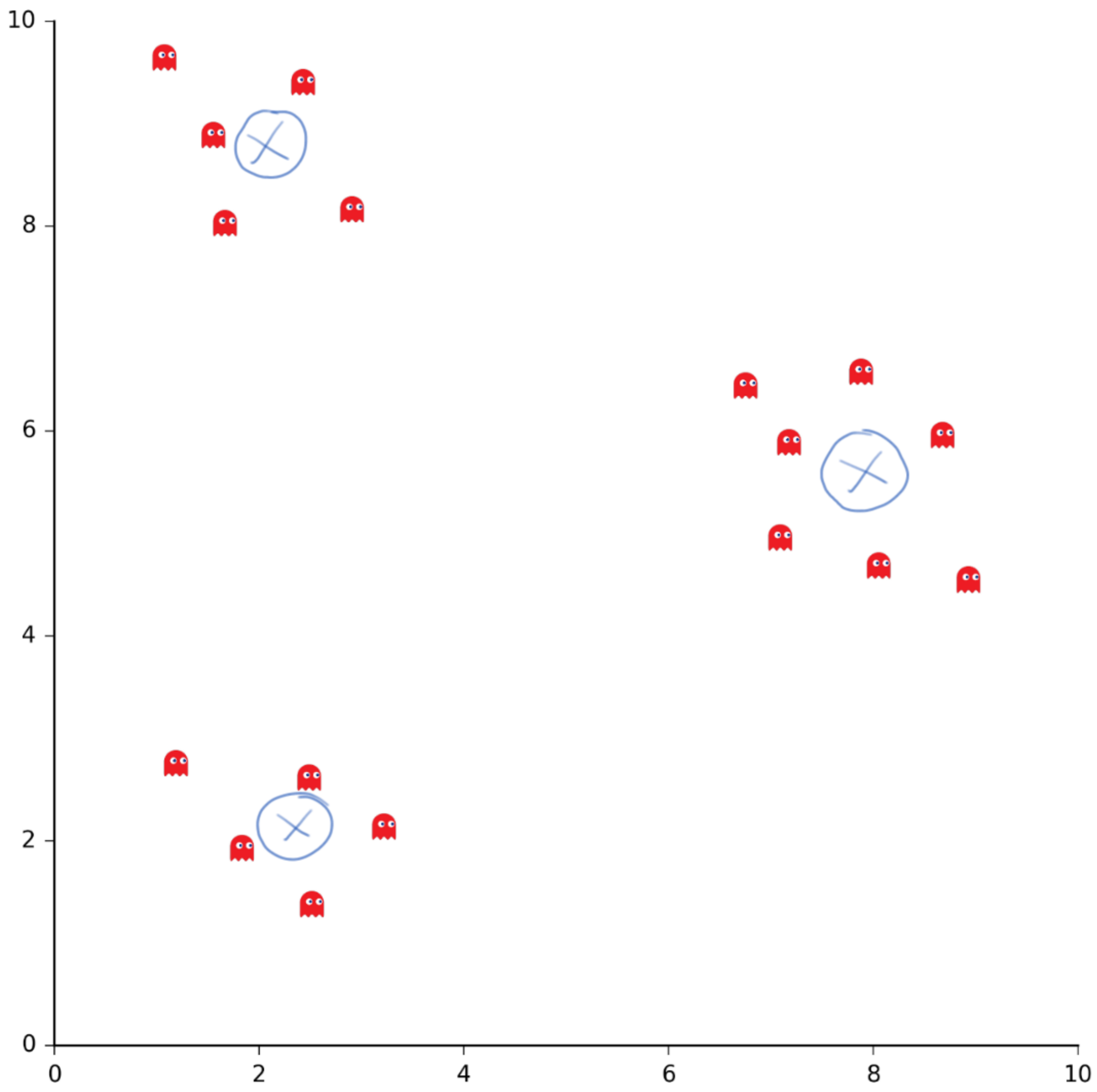
K-Means



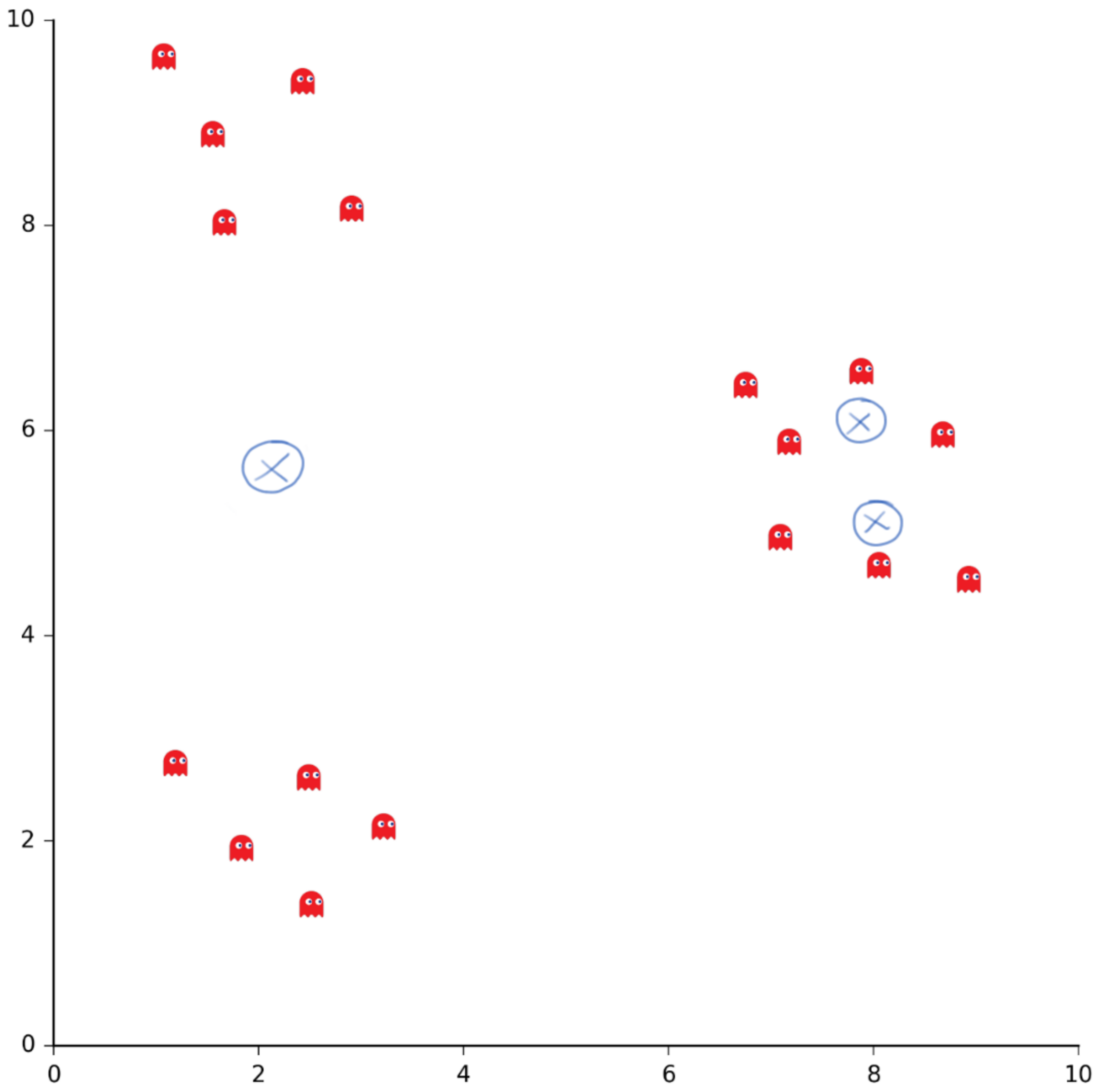
"…it's too easy to throw k-means on your data, and nevertheless get a result out (that is pretty much random, but you won't notice)."
— Anony-Mousse

http://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means

Which one is correct?

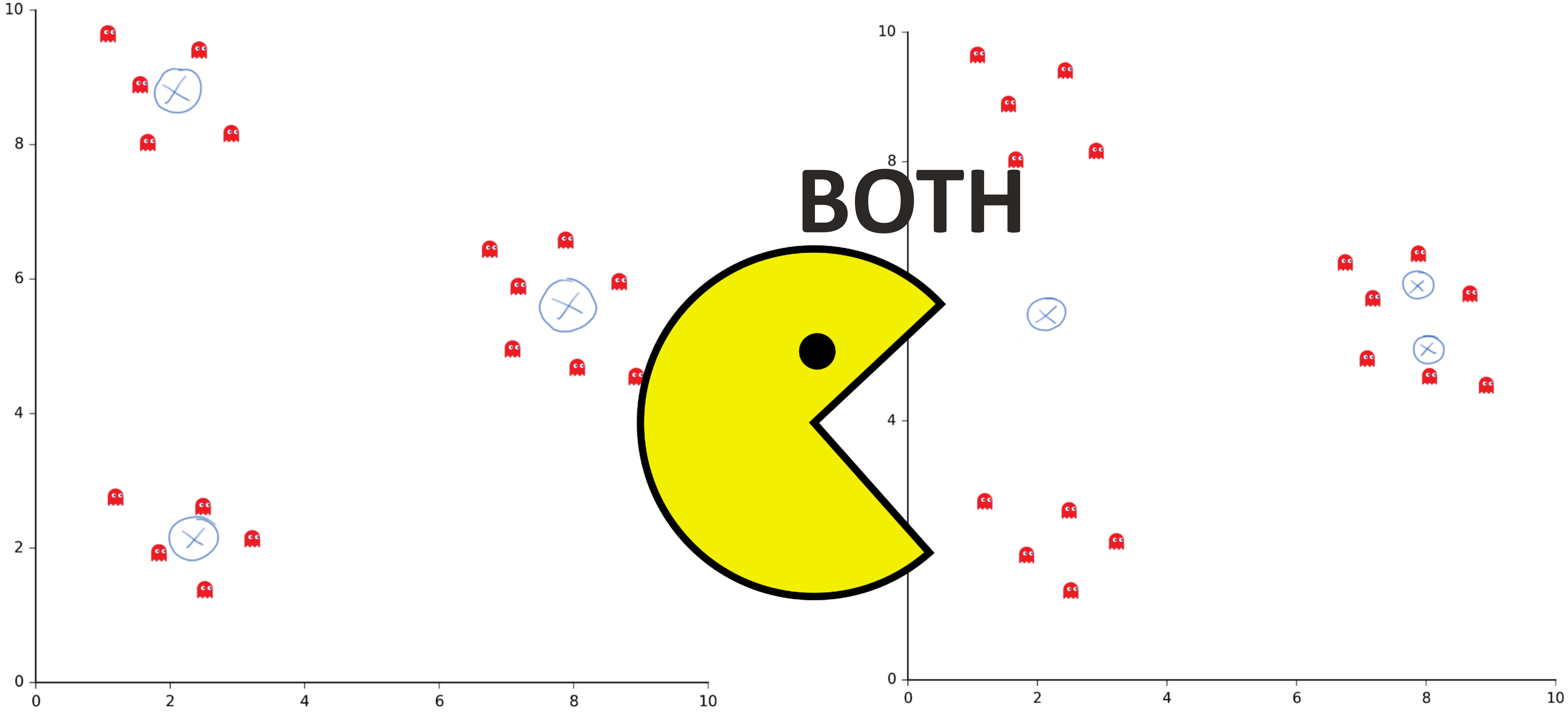


A



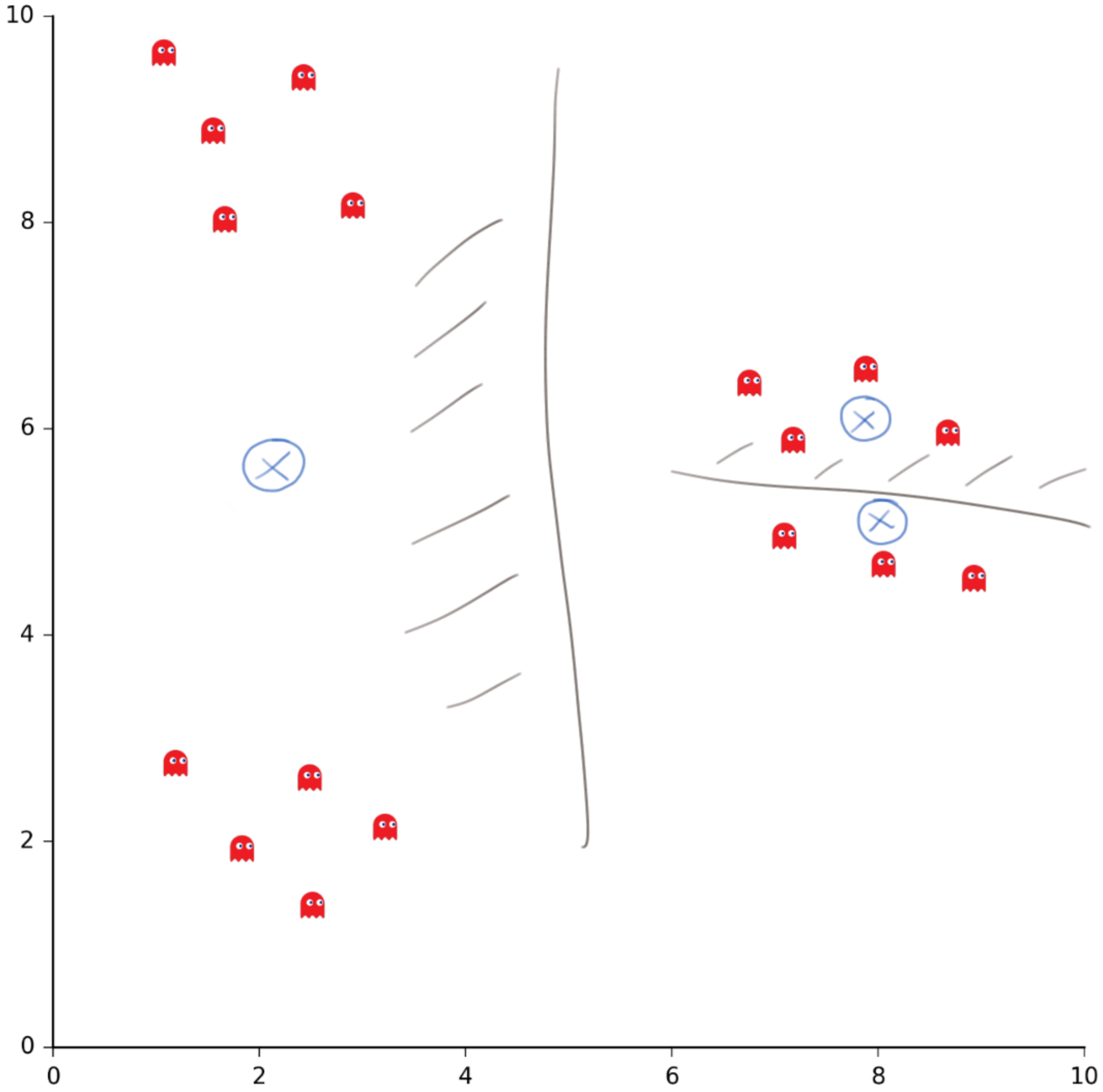
B

Which one is correct?



BOTH

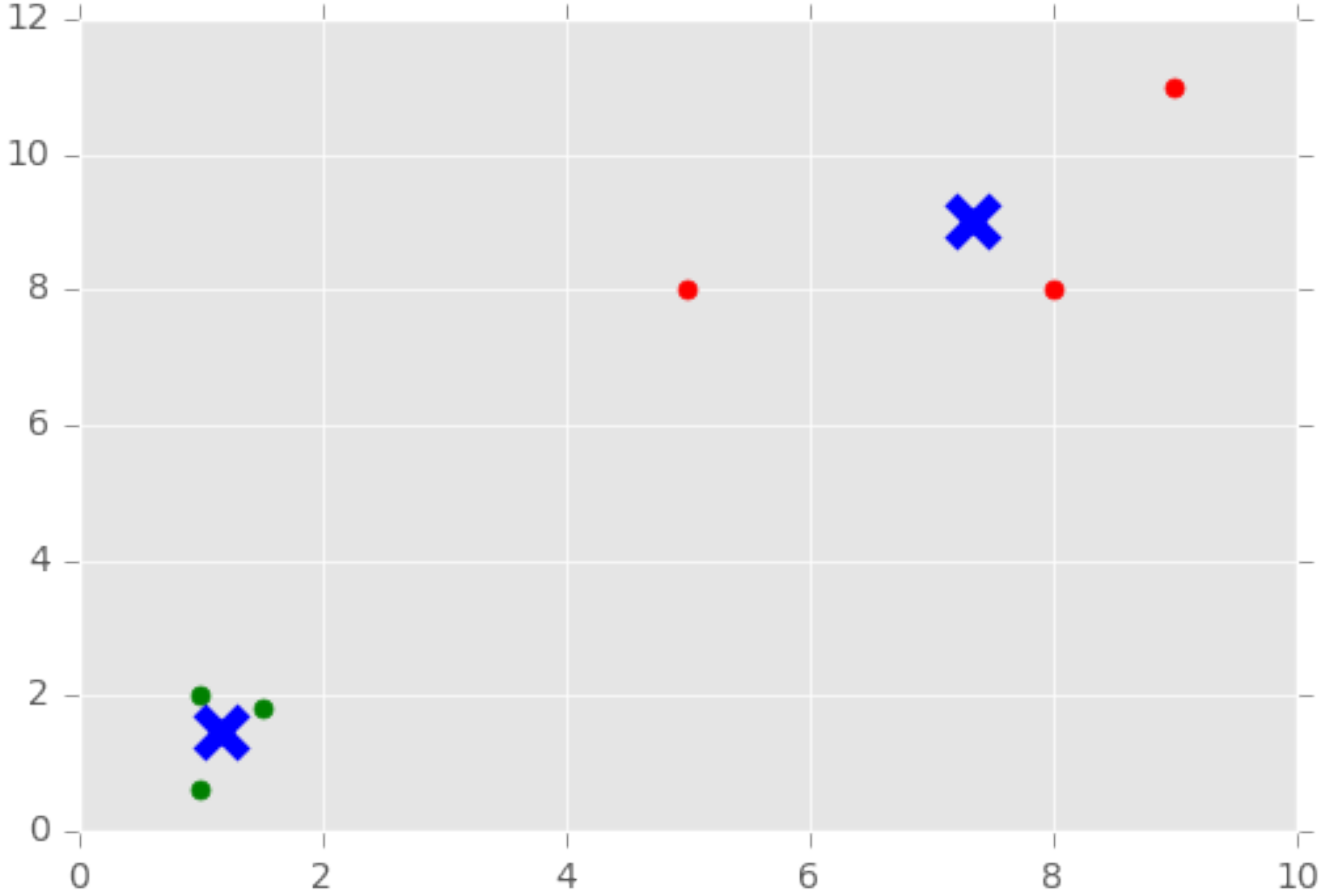
Pain of optimization…Being stuck at sub-optimal local minimum…



Initial guess matters!
Same outcome cannot be guaranteed

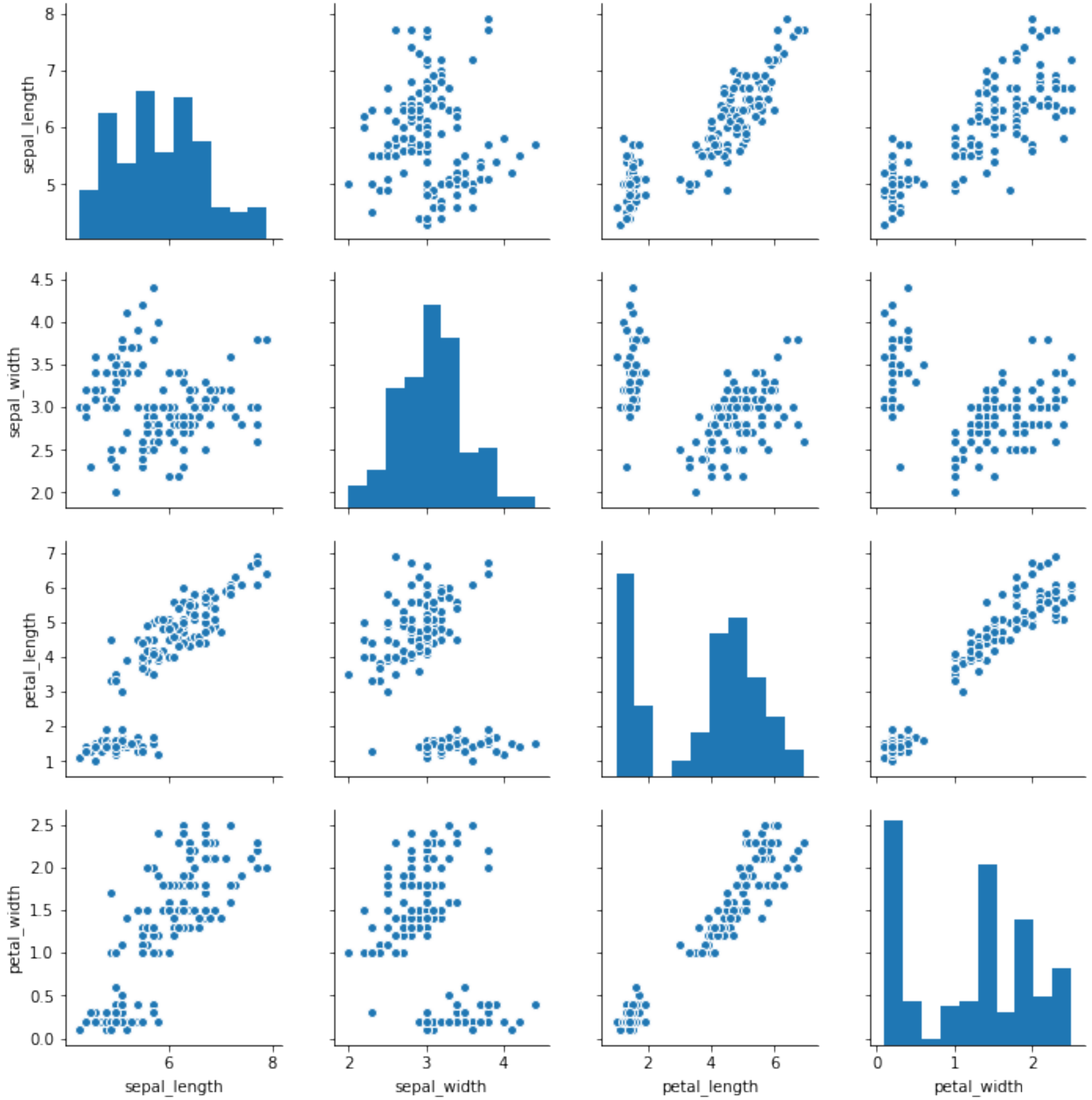
K-Means in practice (Python version)

```
#Import from Scikit-learn
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_f
```



The Dataset

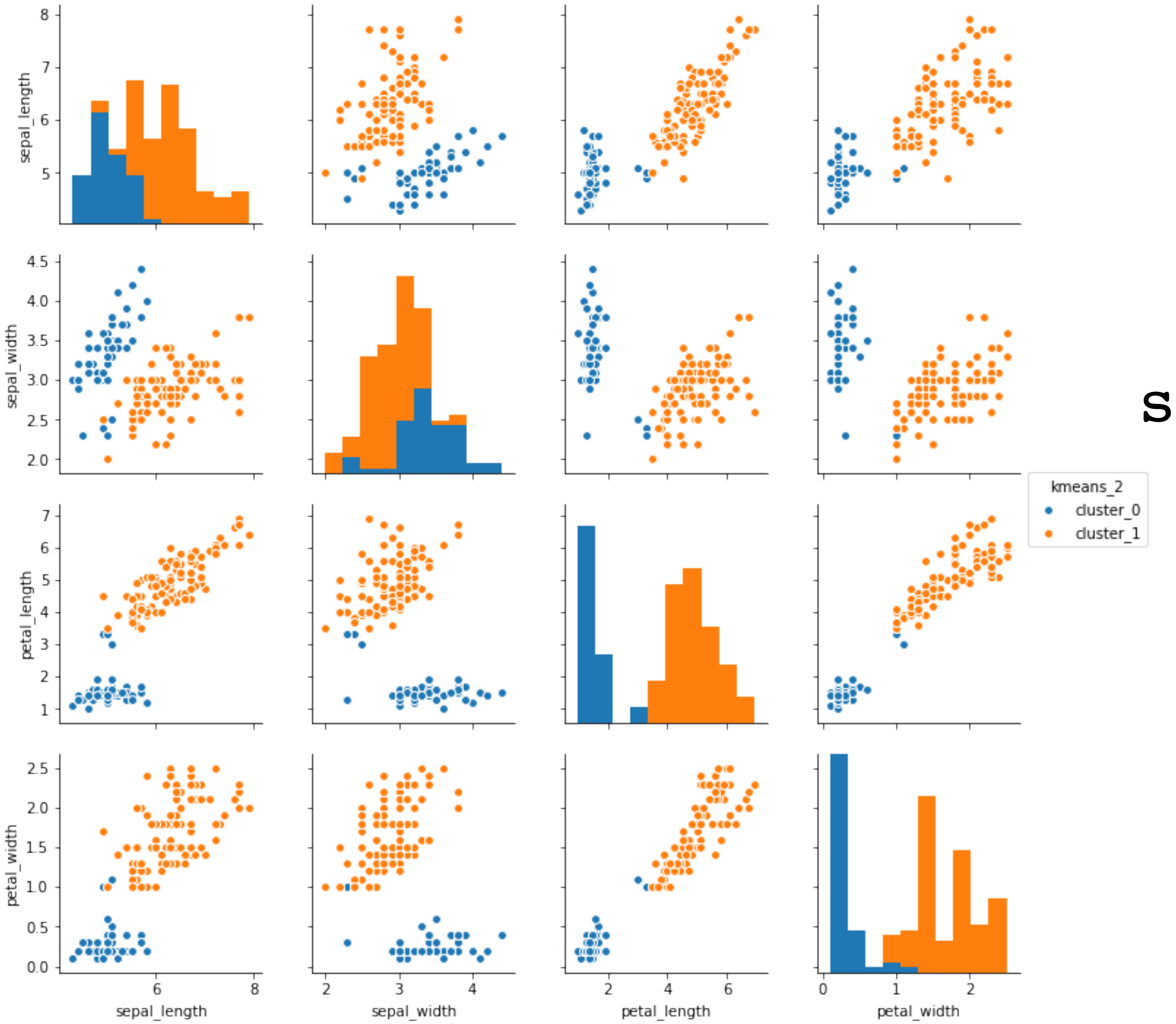
```
sns.pairplot(<data>)
```



K-Means Clustering

```
kmeans = KMeans(n_clusters=2, n_init=2)
kmeans.fit( <data> )
```

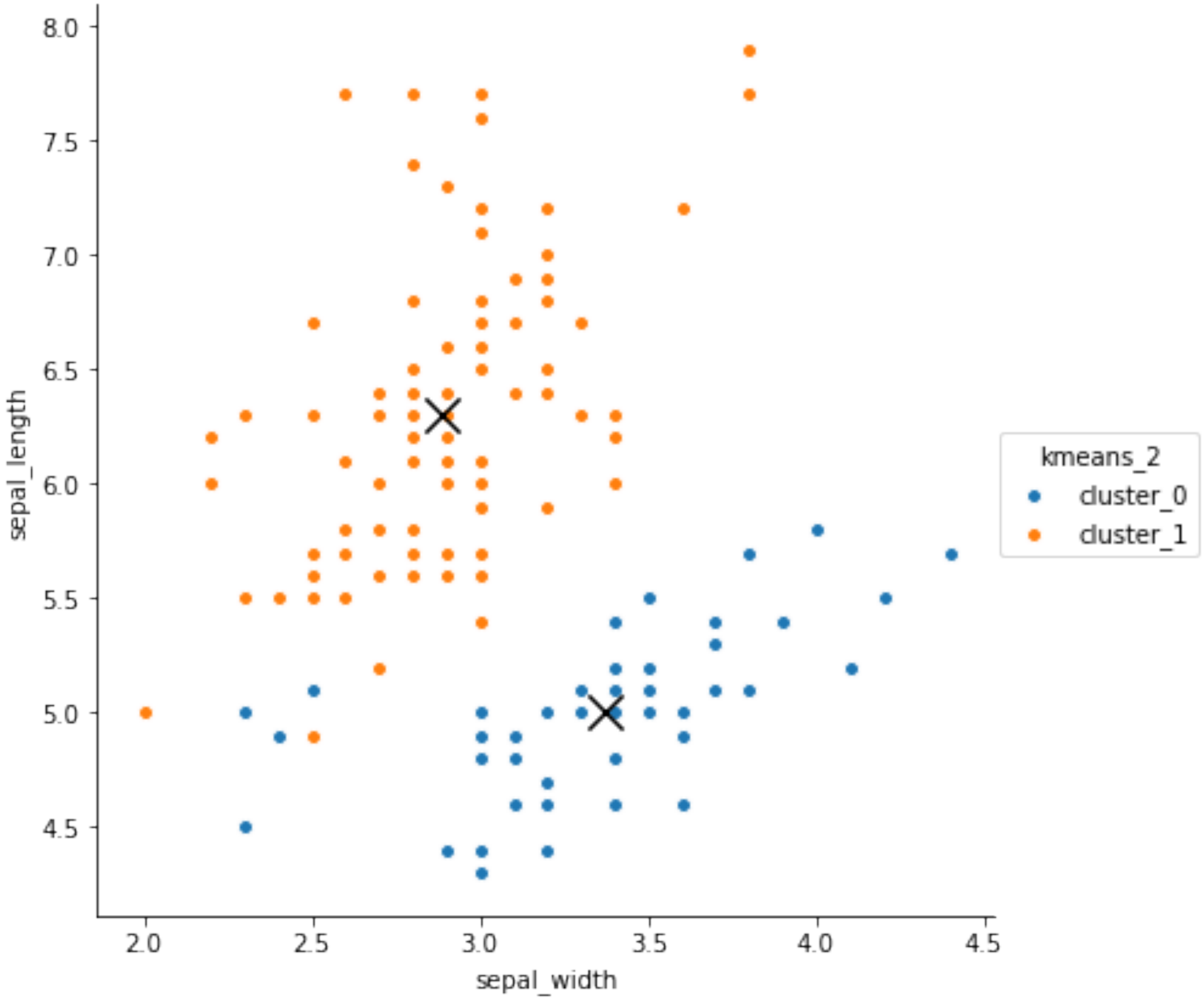
K-Means Clustering



```
sns.pairplot(<data>,hue="kmeans_2")
```

K-Means Clustering

```
sns.pairplot(<data>,x_vars="col_1",y_vars="col_2",hue="kmeans_2",size=6)
plt.scatter(<cluster_centers>,<col_2>, linewidths=3, marker='x', s=200,
c='black')
```



K-Means is affected by the scale of every feature.

Feature Scaling

For k-means clustering, features must be scaled to the same ranges of values to contribute "equally" to the euclidean distance calculation.

Each row is transformed per-column by:

- Subtracting from the element in each row the mean for each feature (column) and then taking this value and
- Dividing by that feature's (column's) standard deviation.

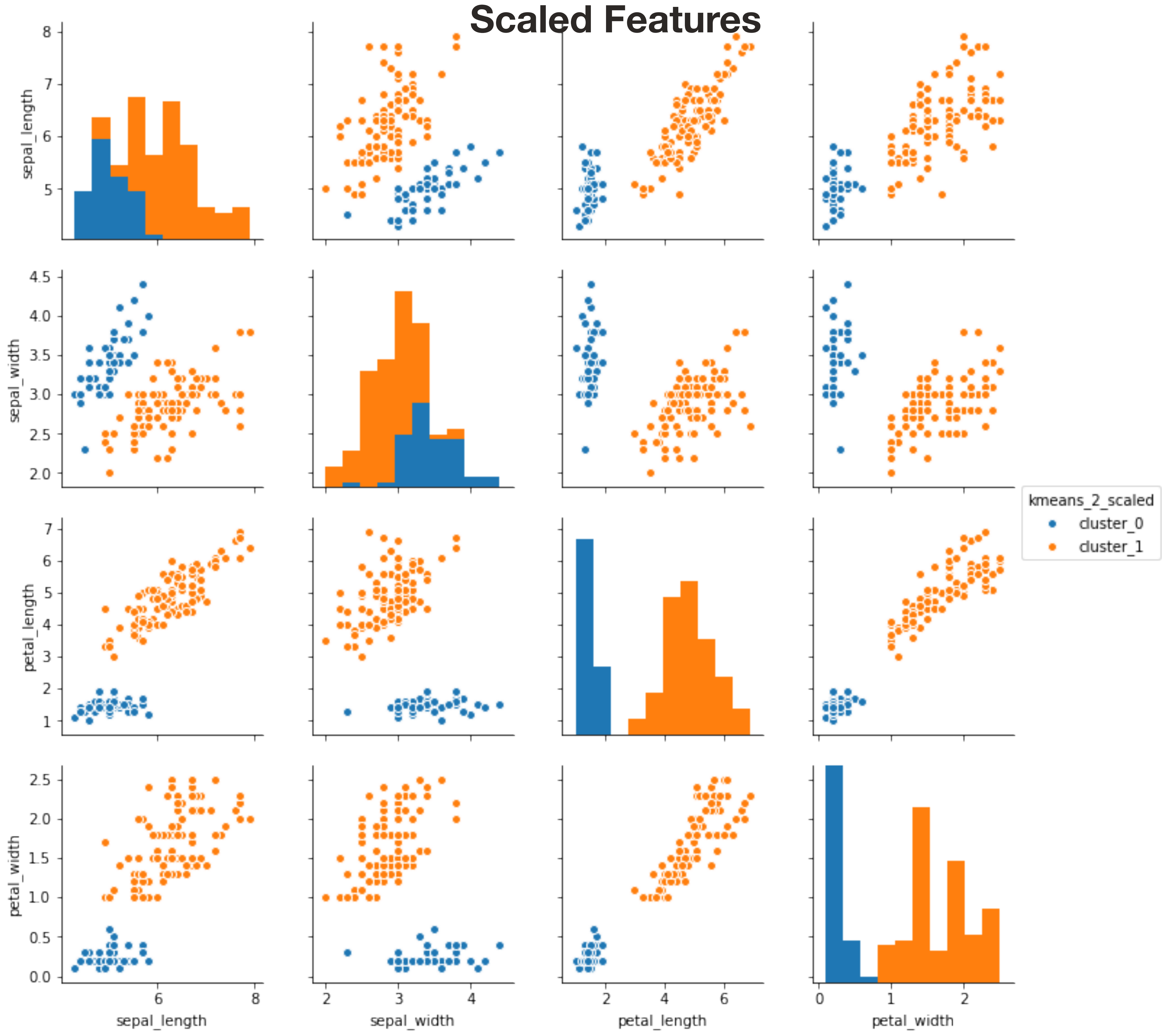
Feature Scaling

```
# center and scale the data
scaler = StandardScaler()

raw_data_scaled = scaler.fit_transform( <data> )

data_scaled = pd.DataFrame( raw_data_scaled,
columns=features )
```

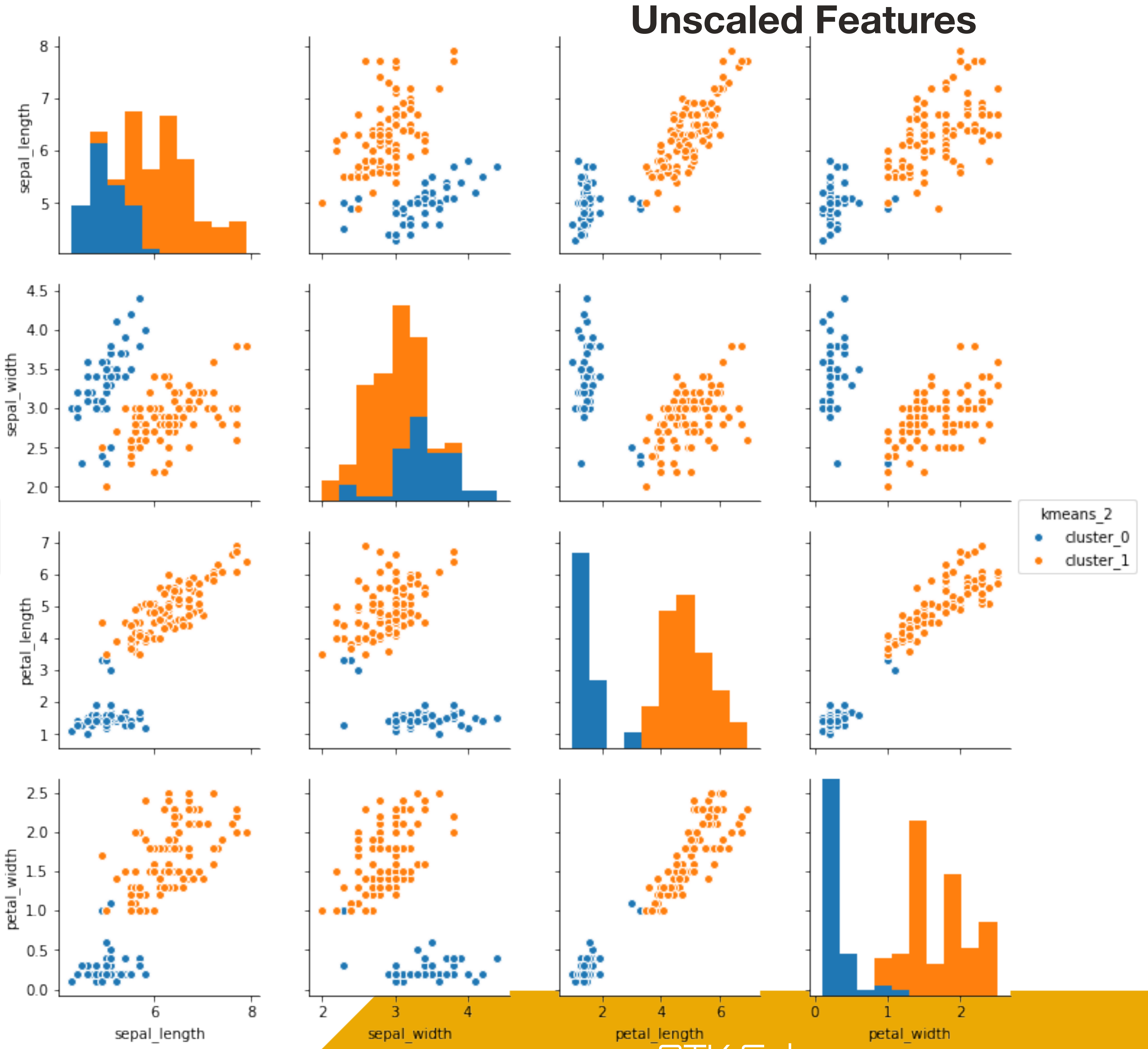
Feature Scaling

```
# K-means on scaled data
km = KMeans(n_clusters=2, n_init=2)
km.fit( <scaled_data> )
```

Feature Scaling

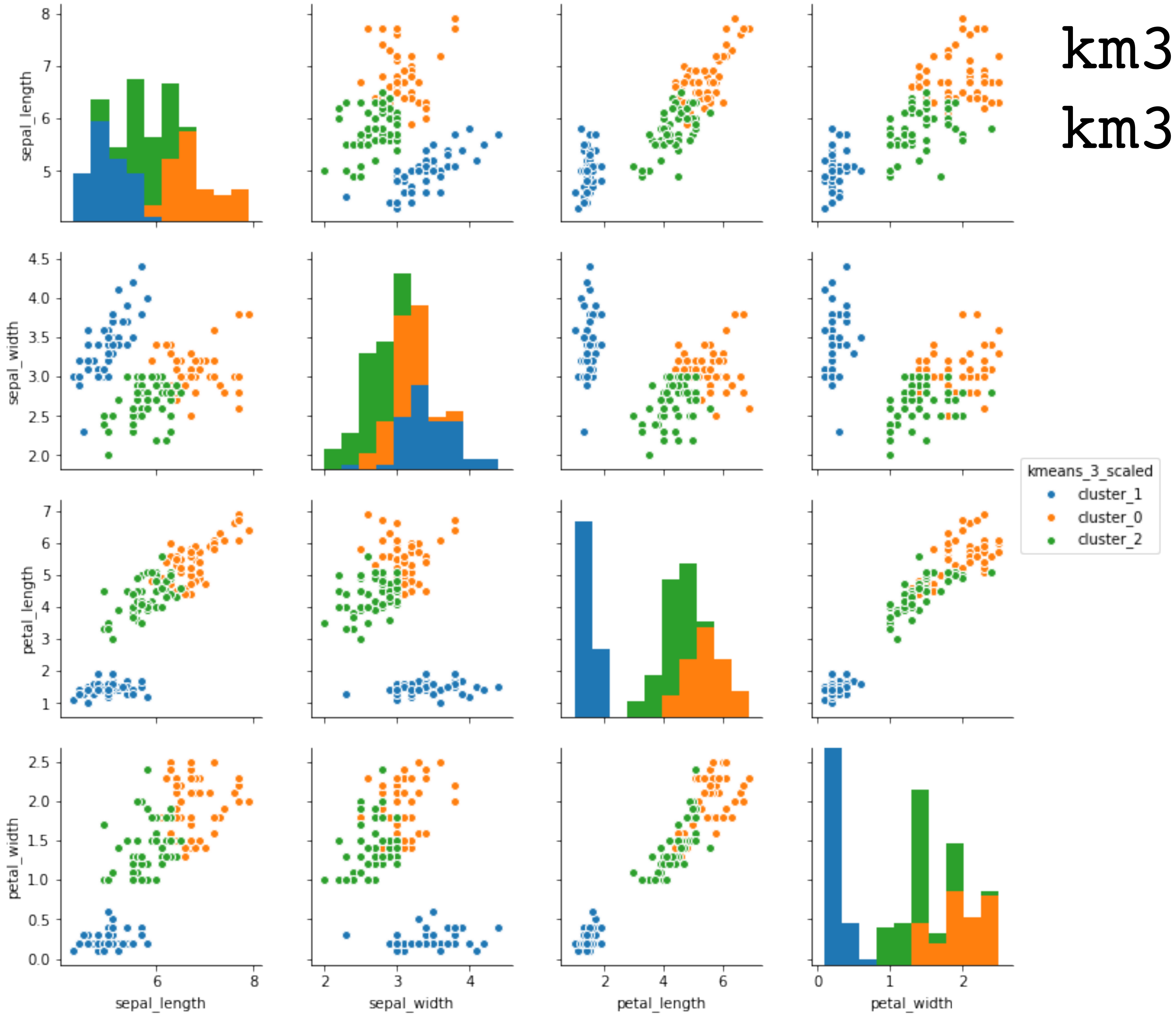


Scaled Features



Unscaled Features

More Clusters

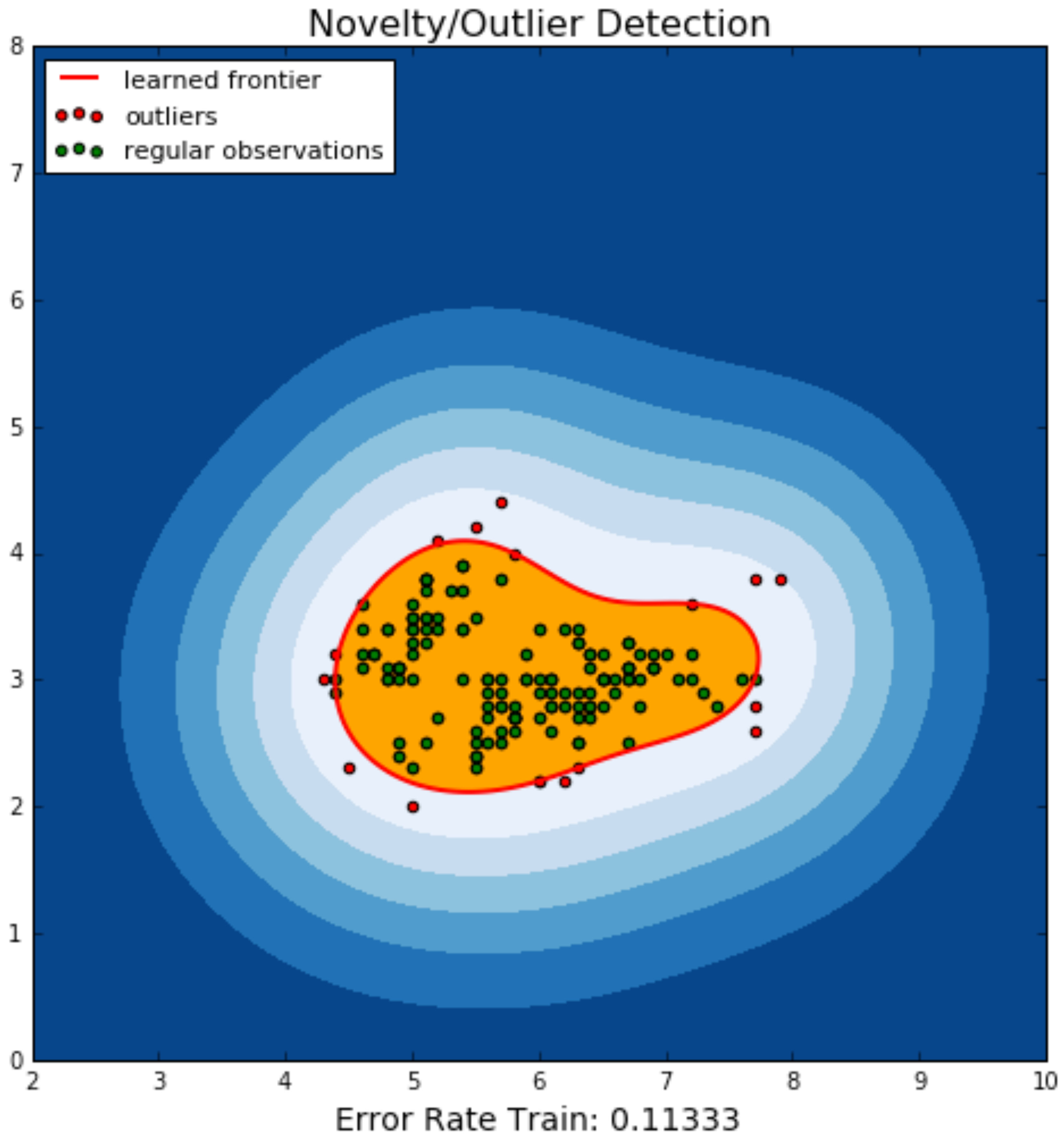


```
km3 = KMeans(n_clusters=3, n_init=2)
km3.fit(scaled_data)
```

Outlier Detection

```
clf = svm.OneClassSVM( tol=0.001, nu=0.1)
clf.fit(X)
target_pred_outliers=clf.predict(X)
```

Delete n% of “outlier data”, here ~10%



http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

Evaluating your Model

The Silhouette Coefficient is a common metric for evaluating clustering "performance" in situations when the "true" cluster assignments are not known.

b = mean distance to next nearest cluster

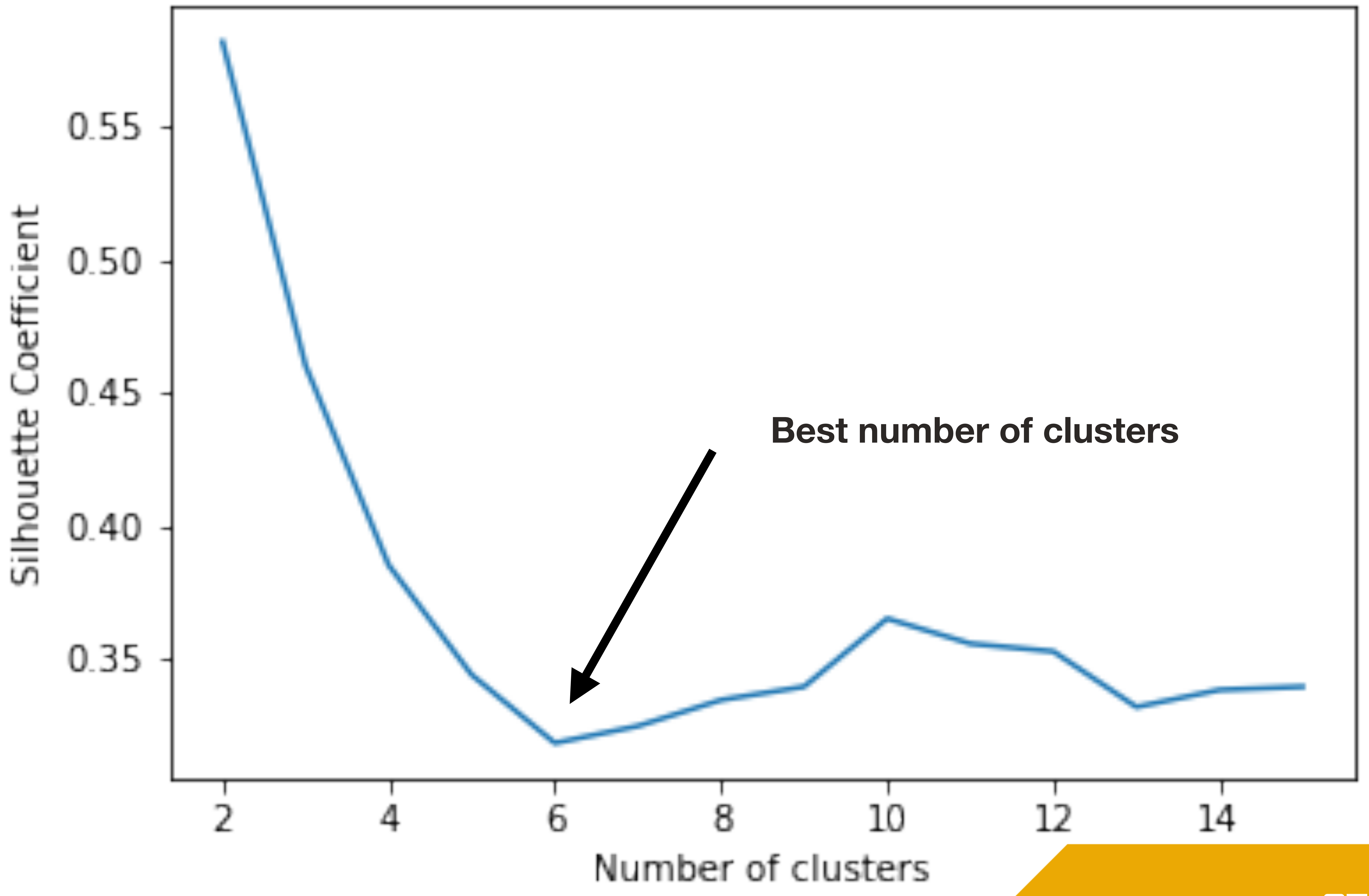
a = mean distance to other points in cluster

```
silhouette_coeff = (b - a) / max(a,b)
```

Evaluating your Model

```
k_range = range(2,16)
scores = []
for k in k_range:
    km_ss = KMeans(n_clusters=k, random_state=1)
    km_ss.fit(iris_data_scaled)
    scores.append(silhouette_score(<data>,
km_ss.labels_))
```

Evaluating your Model

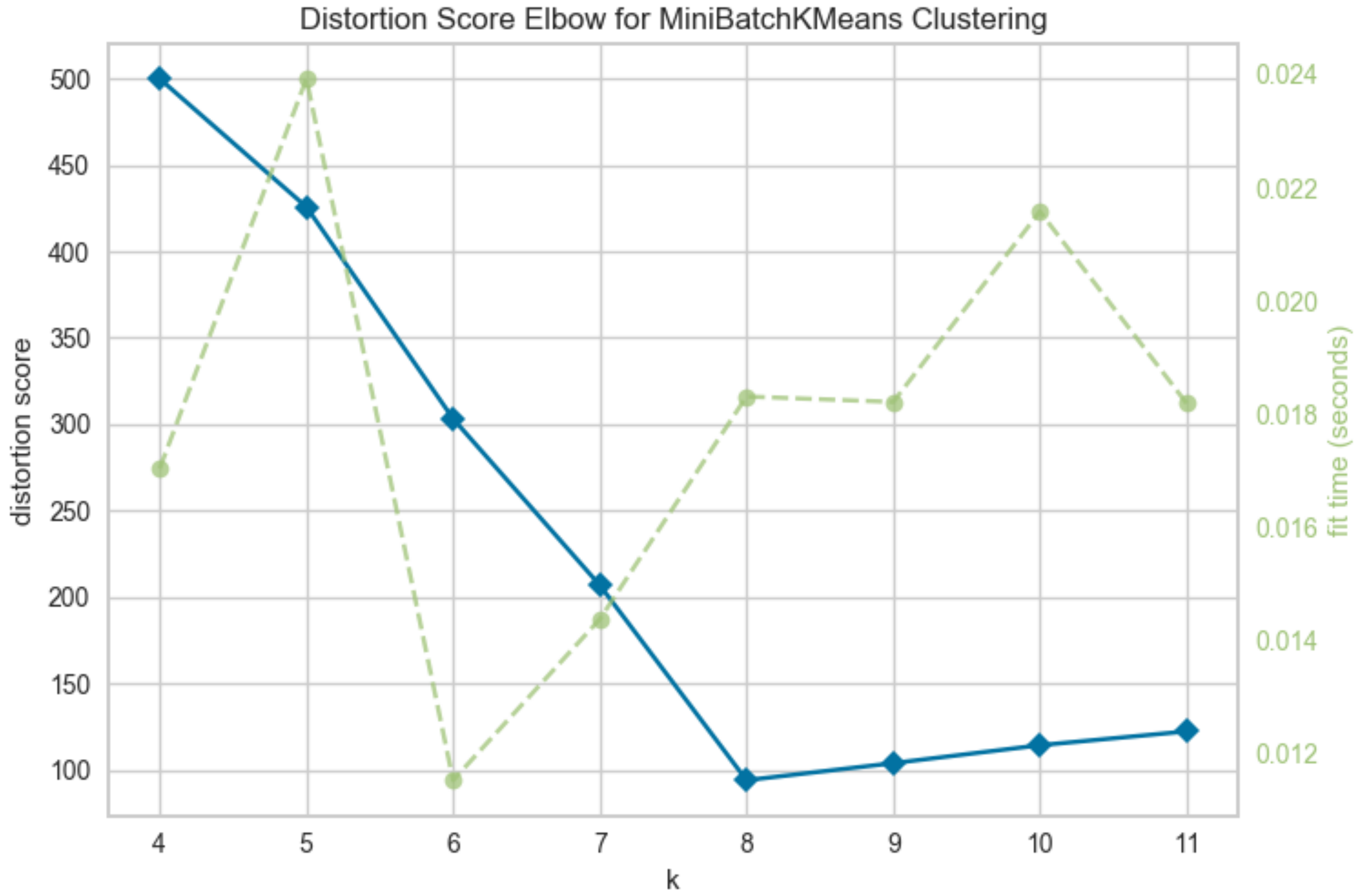


Best number of clusters

Evaluating your Model

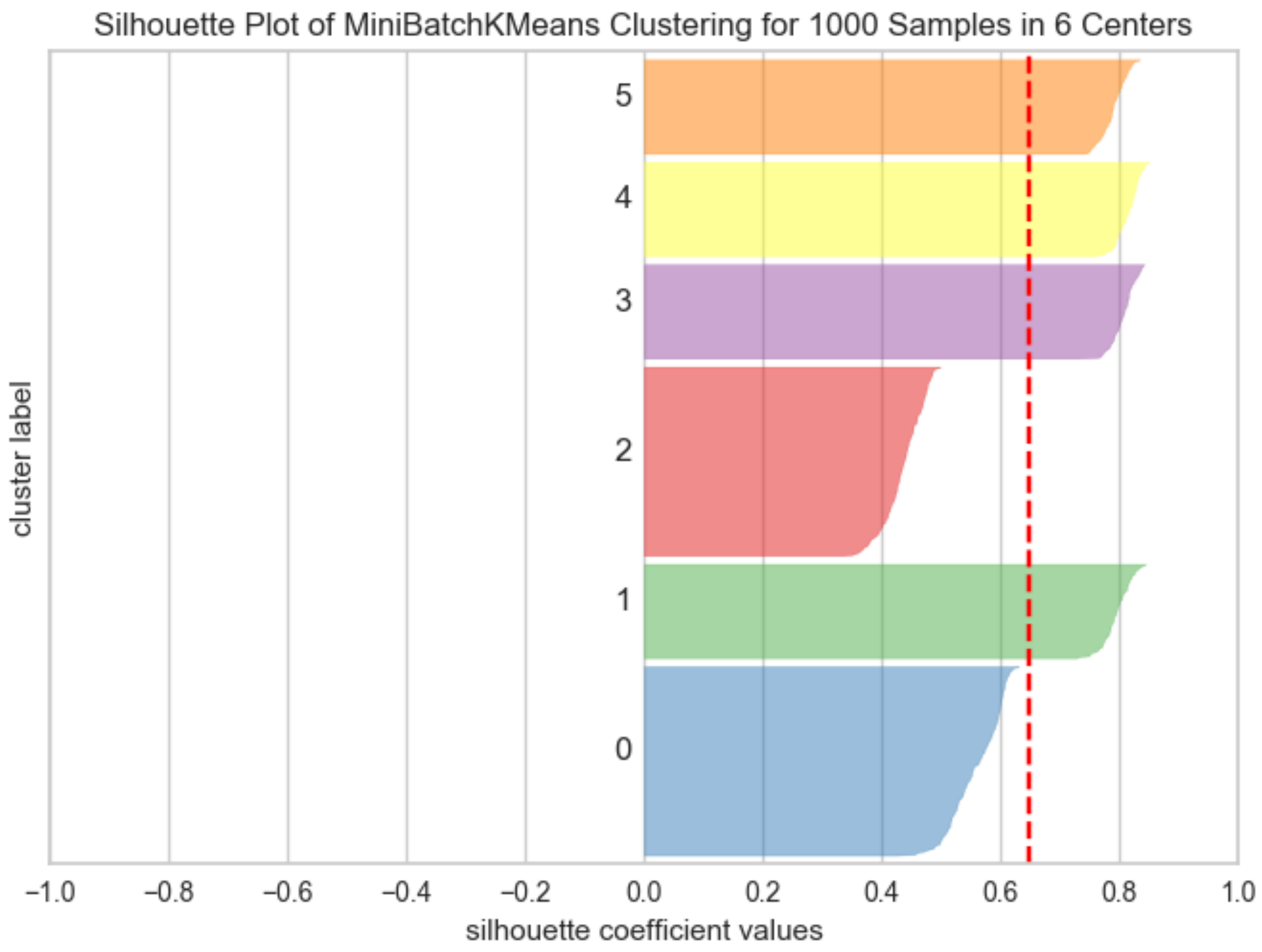
```
from yellowbrick.cluster import KElbowVisualizer
visualizer = KElbowVisualizer(KMeans(), k=(4,12))
```

```
visualizer.fit(X)
visualizer.poof()
```



Evaluating your Model

```
from yellowbrick.cluster import SilhouetteVisualizer
model = MiniBatchKMeans(6)
visualizer = SilhouetteVisualizer(model)

visualizer.fit(X)
visualizer.poof()
```



DBSCAN

DBSCAN stands for **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise.

Whereas K-means does not care about the density of data, DBSCAN does, under the assumption that regions of high density in your data should be treated as clusters.

DBSCAN

DBSCAN does not allow you to specify how many clusters you want. Instead, you specify 2 parameters:

- **ϵ (epsilon)**: This is the maximum distance between two points to allow them to be neighbors
- **min_samples**: The number of neighbors a given point is allowed to have to be able to be part of a cluster

Any points that don't satisfy the criteria of being close enough to other points are labeled outliers and all fall into a single "cluster" (their cluster label by default is -1).

DBSCAN

DBSCAN works as follows:

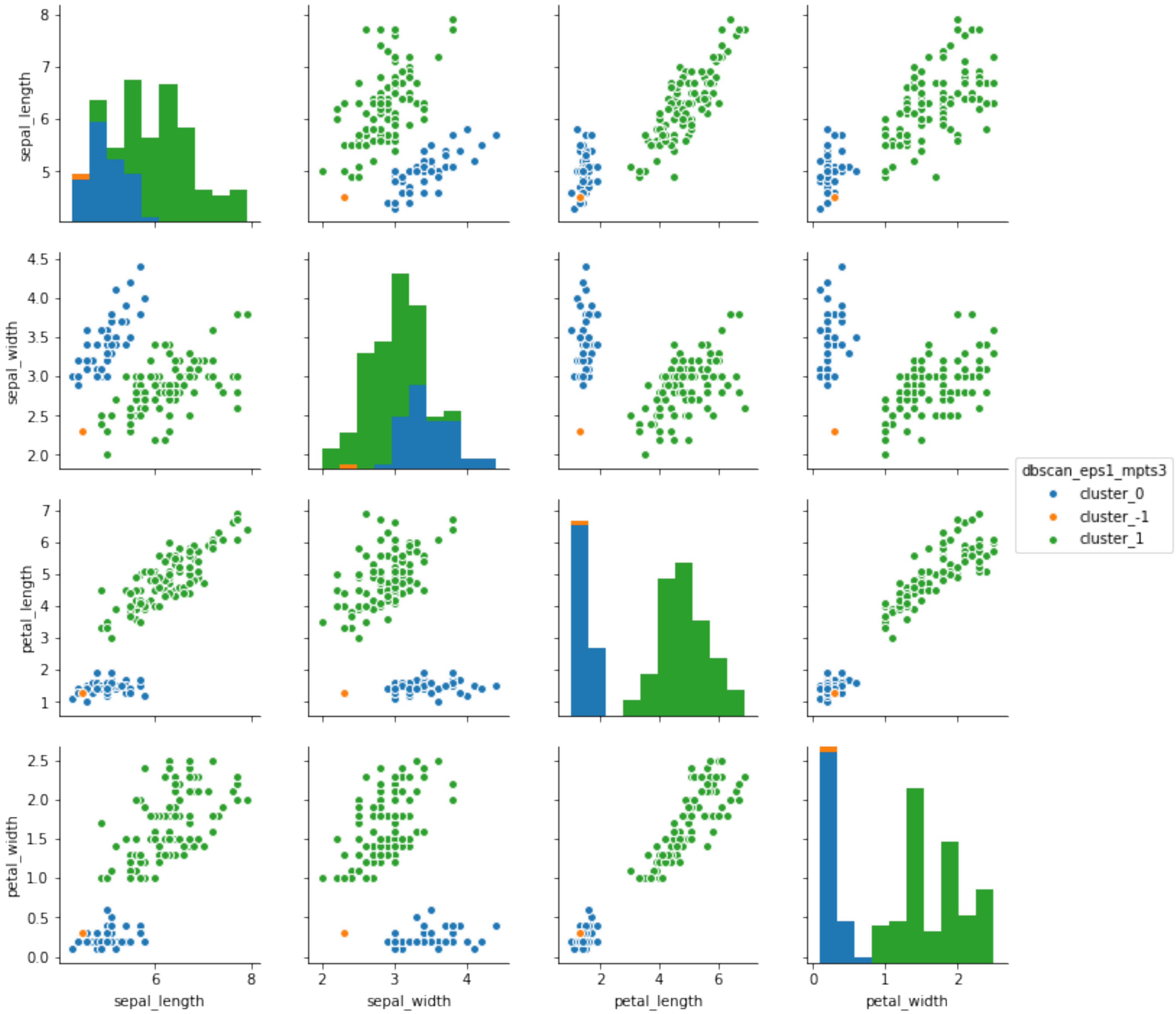
1. Choose an arbitrary starting point in your dataset that has not been seen.
2. Retrieve this point's ϵ-neighborhood (all points that are within a distance ϵϵ from it), and if it contains at least ***min_samples**, a cluster is started.
3. Otherwise, the point is labeled as an outlier (-1). Note: This point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.
4. If a point is found to be a dense part of a cluster, its ϵ-neighborhood is also part of that cluster. All points that are found within the ϵ-neighborhood are added, as is their own ϵ-neighborhood when they are also dense.
5. Continue until the density-connected cluster is completely found.
6. Find a new unvisited point to process and repeat.

DBSCAN

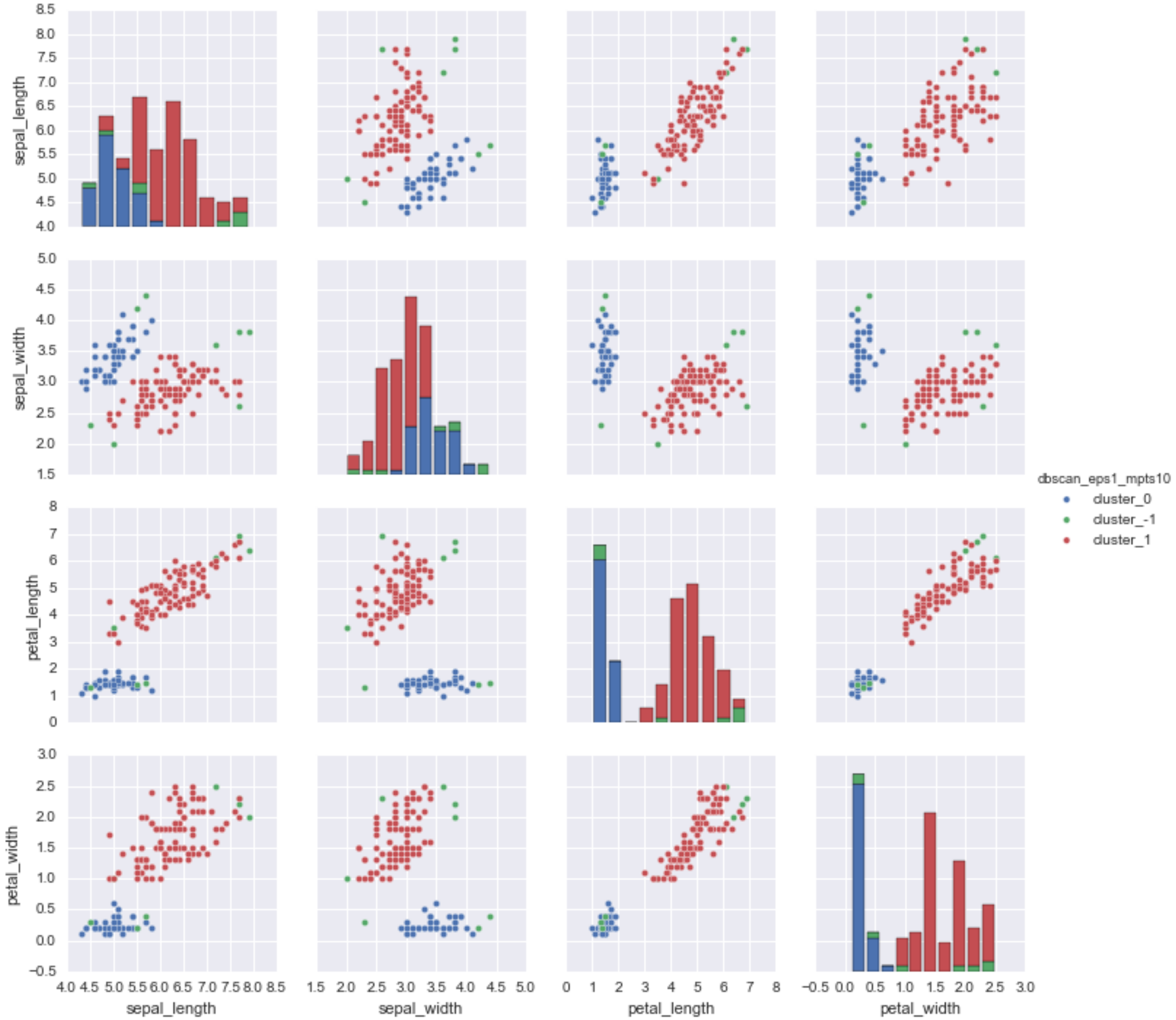
```
db = DBSCAN(eps=1, min_samples=3)
db.fit(<scaled_data>)
```

DBSCAN

```
data_no_names['dbscan_eps1_mpts3'] = [ "cluster_" + str(label) for label in db.labels_ ]
sns.pairplot(data_no_names,hue="dbscan_eps1_mpts3")
```

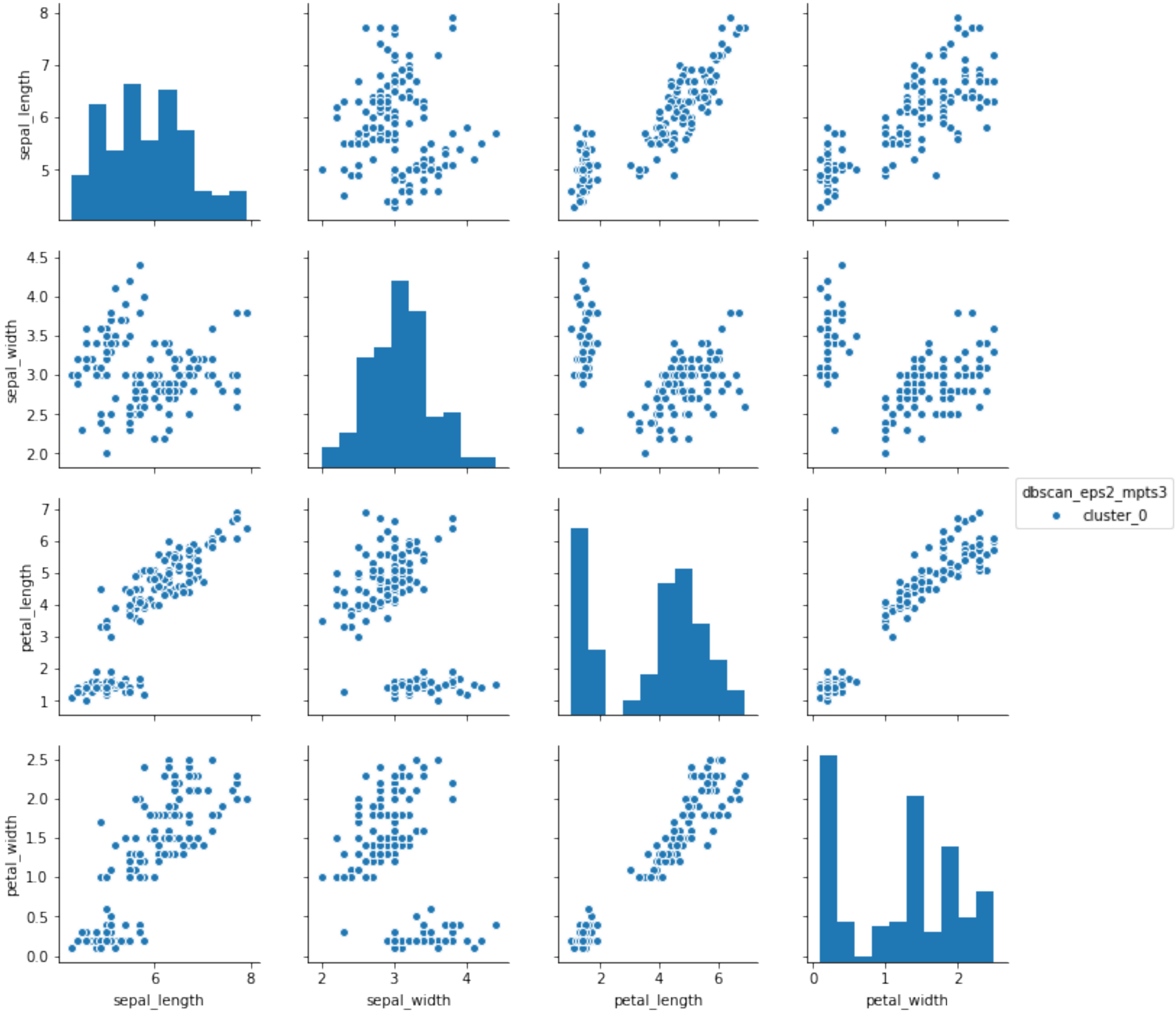


DBSCAN



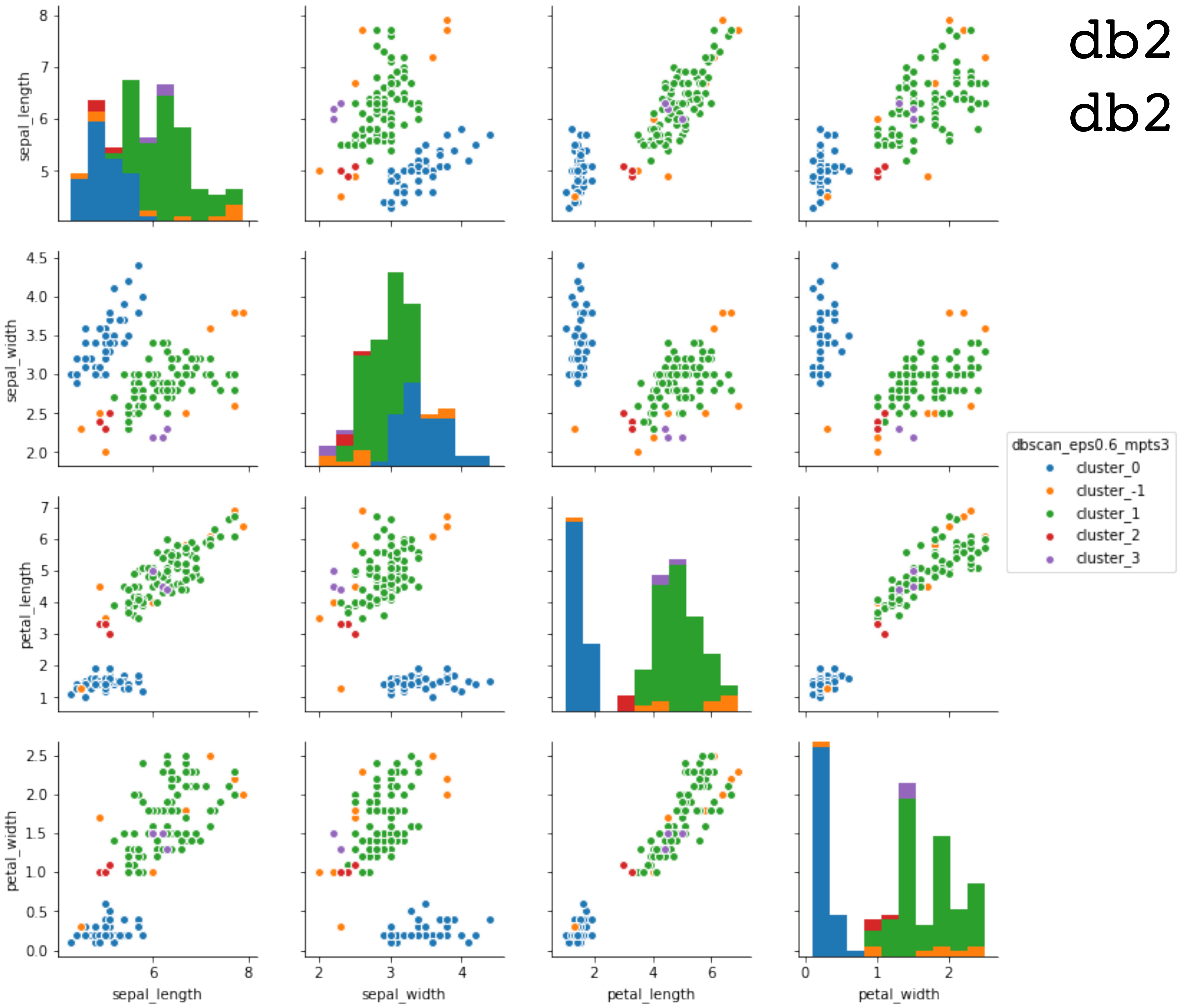
```
db2 = DBSCAN(eps=1, min_samples=10)
db2.fit(data_scaled)
```

DBSCAN



```
db2 = DBSCAN(eps=2, min_samples=3)
db2.fit(iris_data_scaled)
```

DBSCAN



```
db2 = DBSCAN(eps=0.6, min_samples=3)
db2.fit(iris_data_scaled)
```

Other uses of unsupervised learning: Dimensionality Reduction

Other uses of unsupervised learning: Dimensionality Reduction

- Dimensionality reduction reduces the number of features in a dataset without losing the information in a data set
- Common technique: **Principal Component Analysis (PCA)**

Other uses of unsupervised learning: Dimensionality Reduction

- PCA attempts to combine the highly correlated features and represent this data with a smaller number of linearly uncorrelated features.
- The algorithm keeps performing this correlation reduction, finding the directions of maximum variance in the original high dimensional data and projecting them onto a smaller dimensional space.
- These newly derived components are known as principal components.

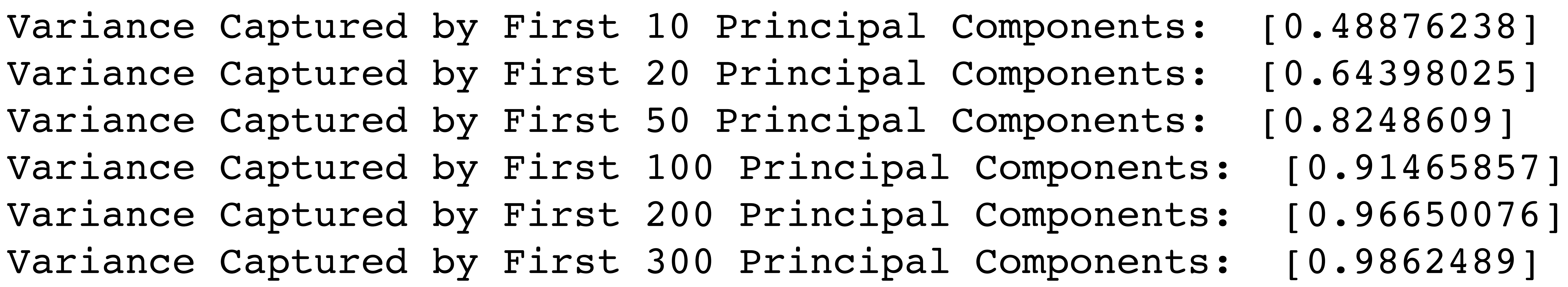
PCA Example

```
from sklearn.decomposition import PCA

n_components = 784
whiten = False

pca = PCA(n_components=n_components, whiten=whiten)

features_train_PCA = pca.fit_transform(features_train_PCA)
features_train_PCA = pd.DataFrame(data= features_train_PCA,
index=train_index)

print("Variance Explained by all 784 principal components: ", \
      sum(pca.explained_variance_ratio_))
```

The variance of all features is 1.0 (everything)

https://github.com/aapatel09/handson-unsupervised-learning/blob/master/03_dimensionality_reduction.ipynb

PCA Example



```
Variance Captured by First 10 Principal Components:  [0.48876238]
Variance Captured by First 20 Principal Components:  [0.64398025]
Variance Captured by First 50 Principal Components:  [0.8248609]
Variance Captured by First 100 Principal Components:  [0.91465857]
Variance Captured by First 200 Principal Components:  [0.96650076]
Variance Captured by First 300 Principal Components:  [0.9862489]
```

Bottom line: 98% of the information is contained in 300 features.   You can therefore, reduce the size of your dataset by half, theoretically without significantly reducing performance.

https://github.com/aapatel09/handson-unsupervised-learning/blob/master/03_dimensionality_reduction.ipynb

Questions?

In Class Exercise

Please complete Worksheet 6.0: Clustering