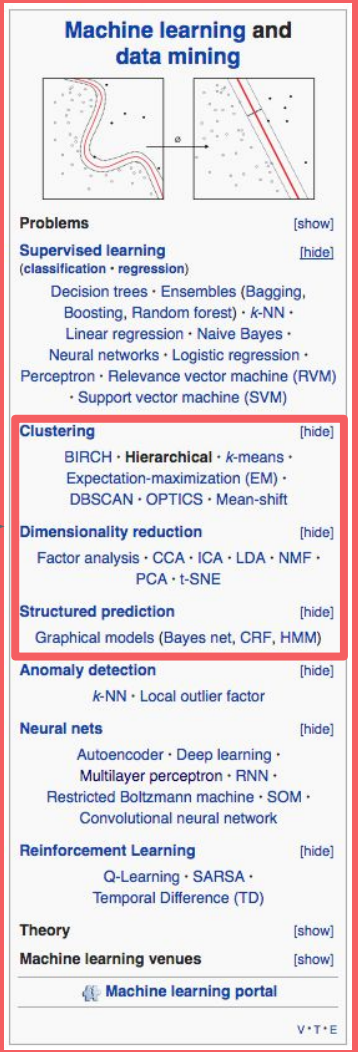
Clustering

Supervised vs Unsupervised

The main difference: one uses labeled data to help predict outcomes, while the other does not.

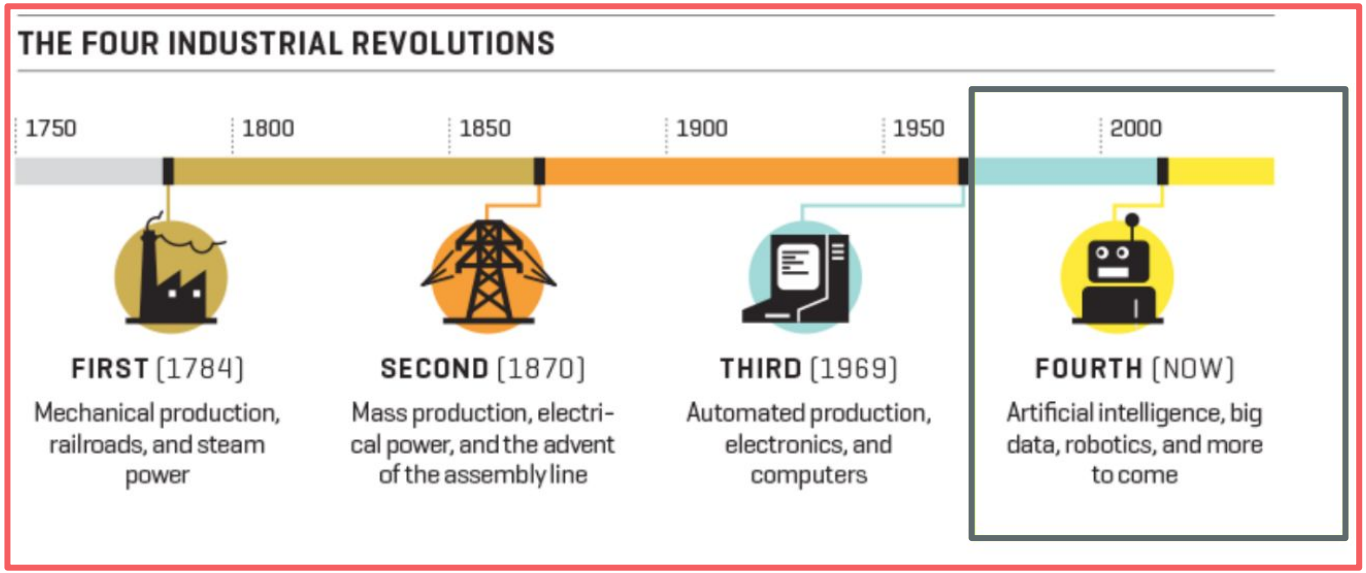
Supervised Learning:
makes use of examples where we know the underlying "truth" (label/output)

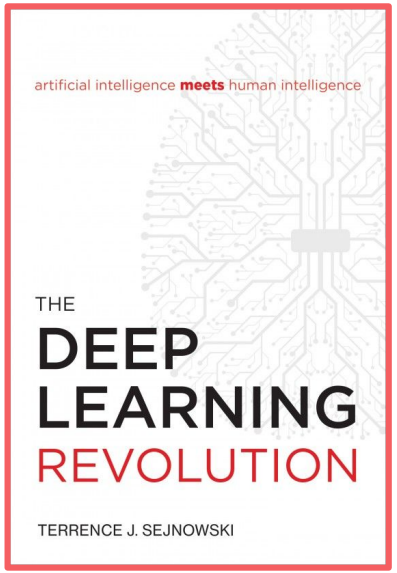
Unsupervised Learning:
Learn underlying structure or features ~~without~~ labeled training data



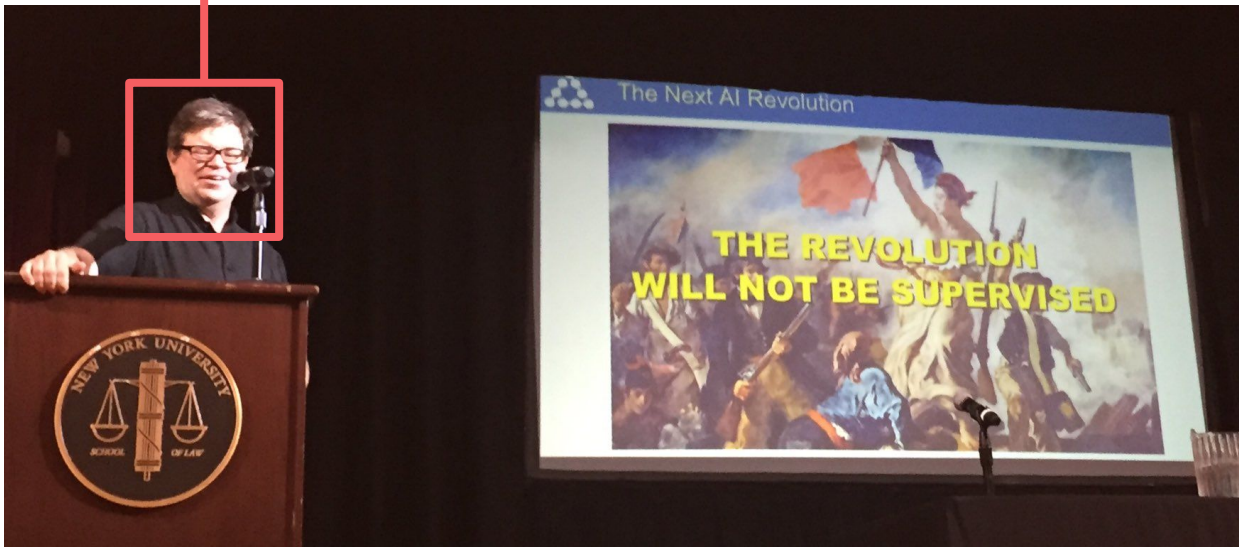
Supervised learning and Revolution

Driven by deep learning and big data, supervised learning changed the world of technology





Yet, the next AI Revolution...



Yann LeCun, recipient of the 2018 Turing Award (often referred to as "Nobel Prize of Computing"), and inventor of deep learning.

Unsupervised learning: 4 main areas

1. **Clustering:**

   Group data points into clusters based on features only.

2. **Dimensionality reduction:**

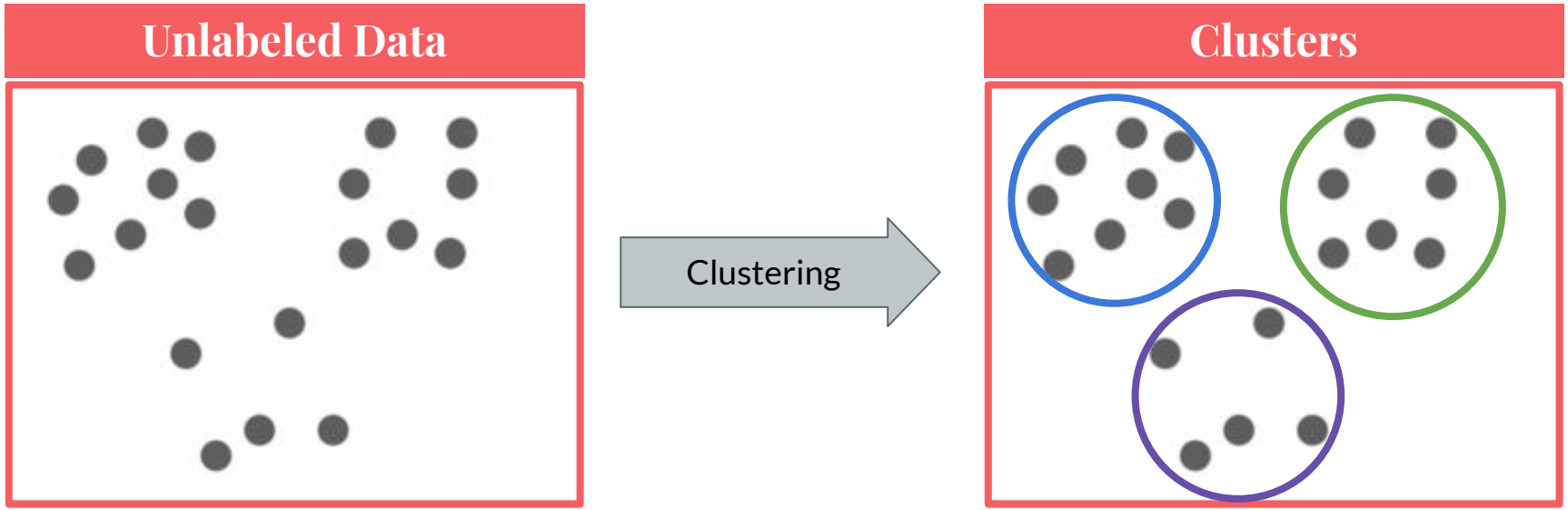
   Remove feature correlation, compress data, visualize data.

3. **Structured prediction:**

   Model latent variables (example: Hidden Markov Models).

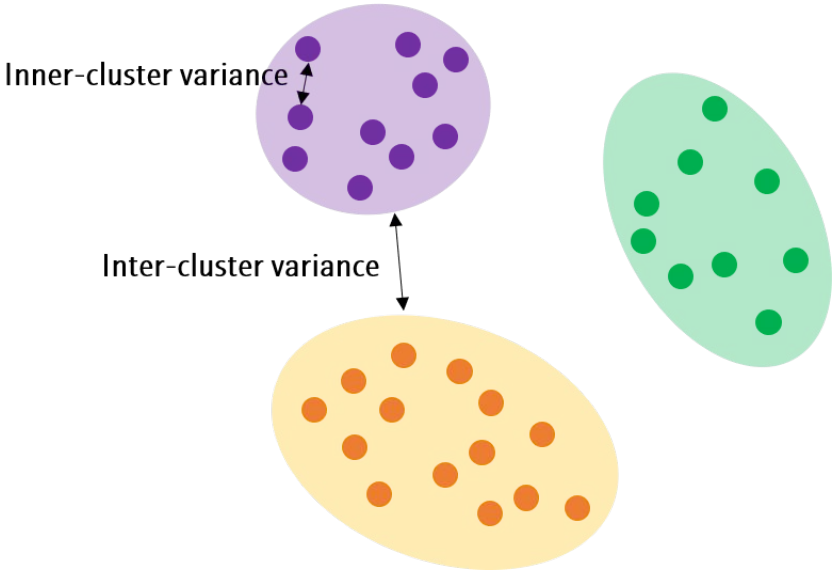
4. **Density Estimation.**

Clustering



Goal of clustering

- **Problem Statement**: Given a set of data points, group them into clusters so that:
  - Points within each cluster are similar to each other
  - Points from different clusters are dissimilar.
- In other words, we'd like to find clusters that:
  - **Minimize** the inner-cluster variance
  - **Maximize** the inter cluster variance



Categories of clustering algorithms

1. ***Partitional clustering***

   Divides data objects into nonoverlapping groups.

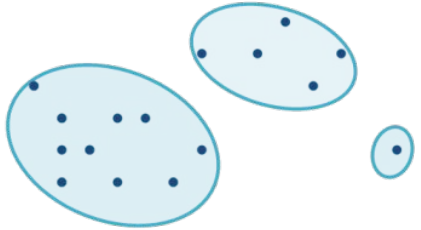
2. ***Hierarchical Clustering***

   Determines cluster assignments by building a hierarchy

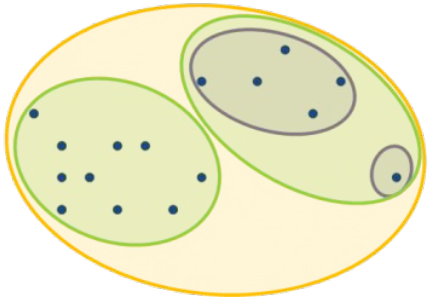
3. ***Density-Based Clustering***

   Determines cluster assignments based on the density of data points in a region.

Partitional

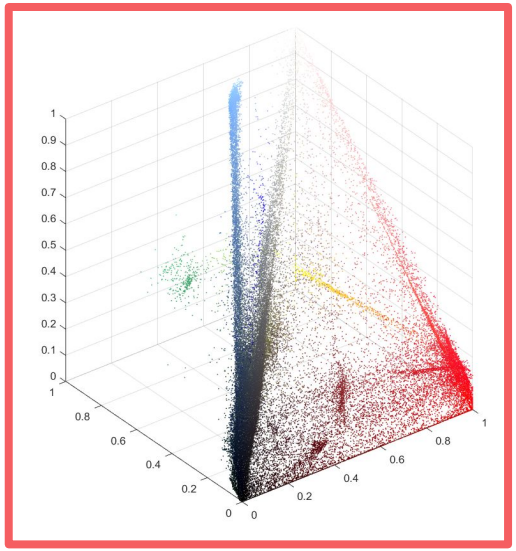


Hierarchical

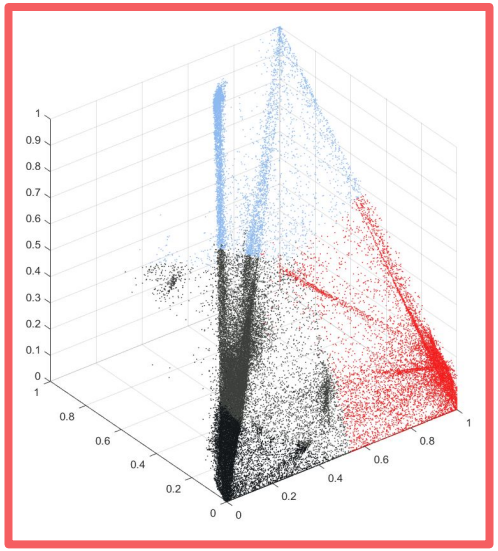


Application 1: Image Compression

The original data:



Clustering:



Application 1: Image Compression

Original: 458 KB and 57678 different colours

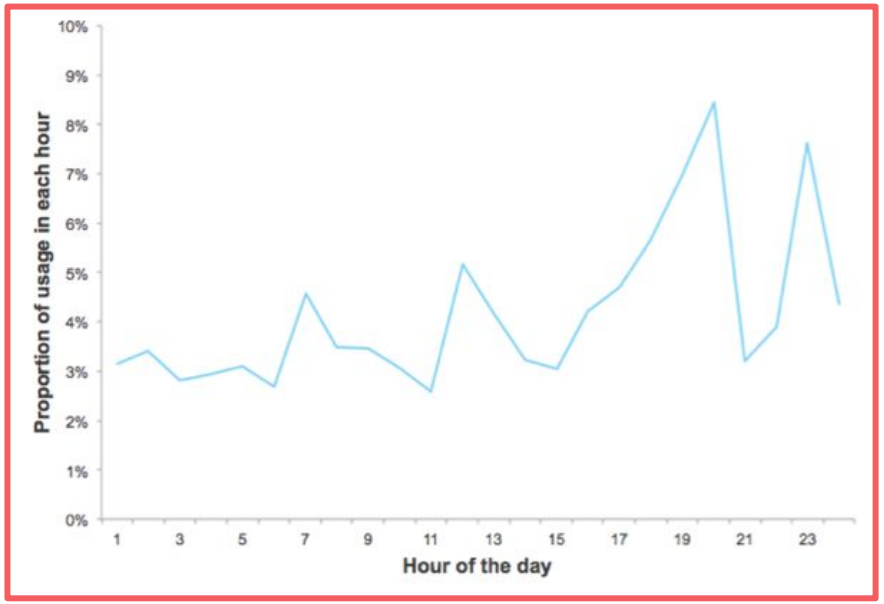
Compressed: 142 KB and 16 clusters

More Compressed: 37 KB and 4 clusters

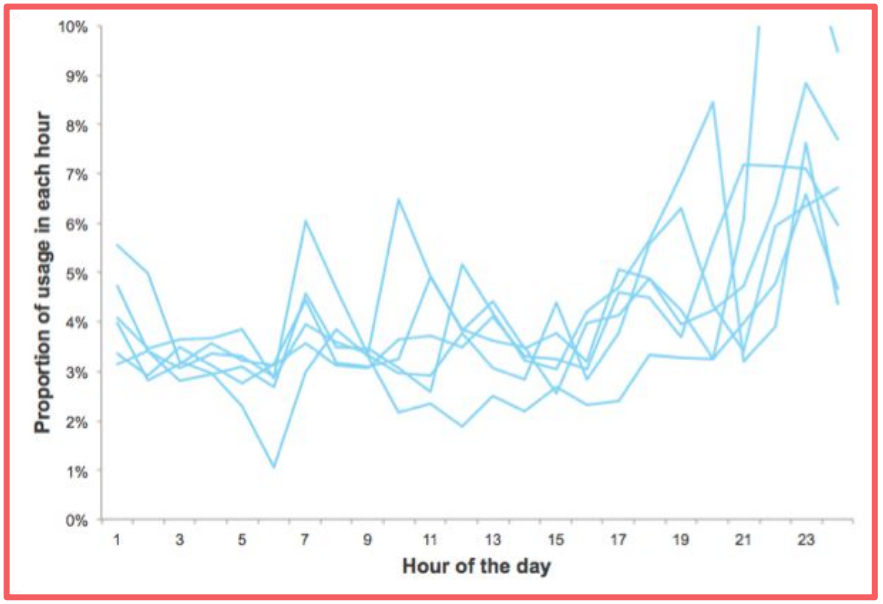
Image compression algorithms take advantage of visual perception of image data to provide superior results

Application 3: Electricity Usage Trends

We all know that not everyone uses energy the same way.



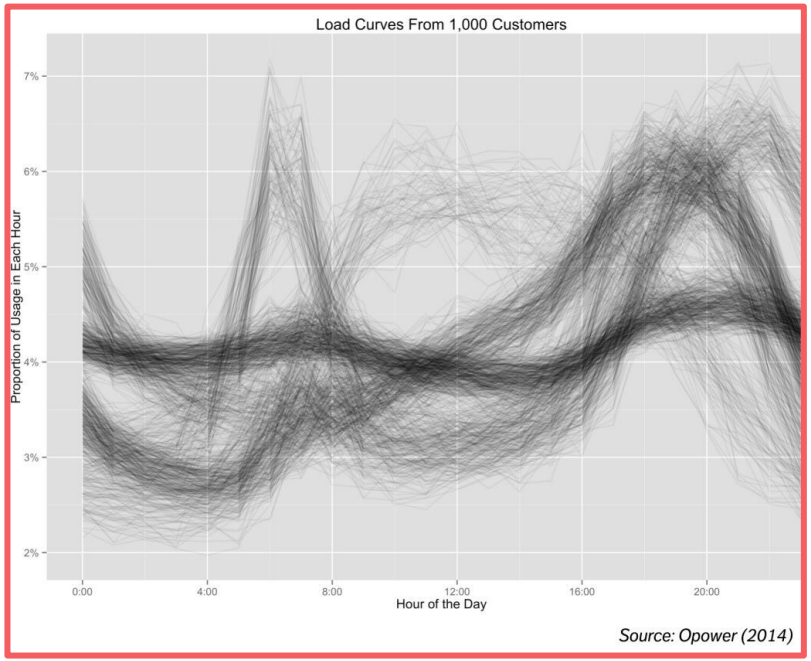
Hourly electricity usage from a single home for a single day



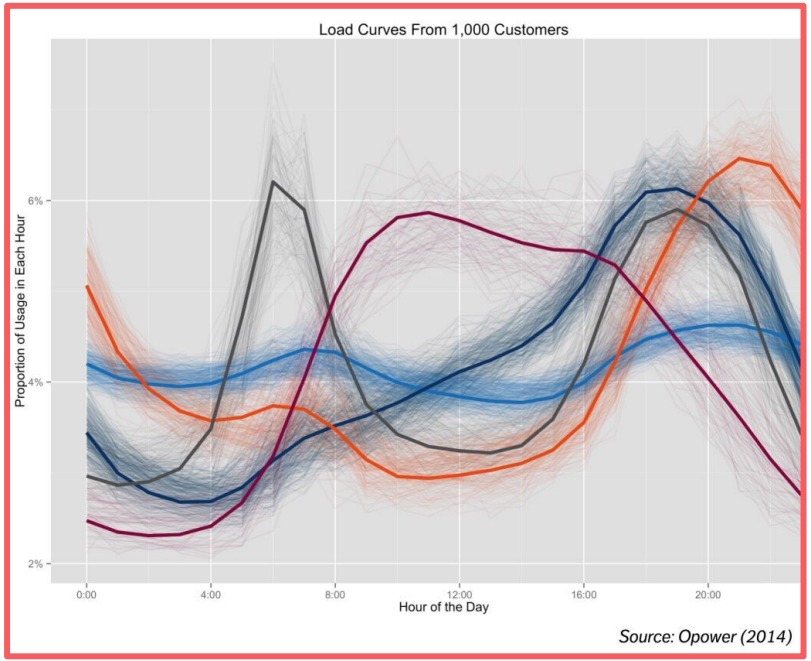
5 more days from that household plotted on top of each other

Application 2: Electricity Usage Trends

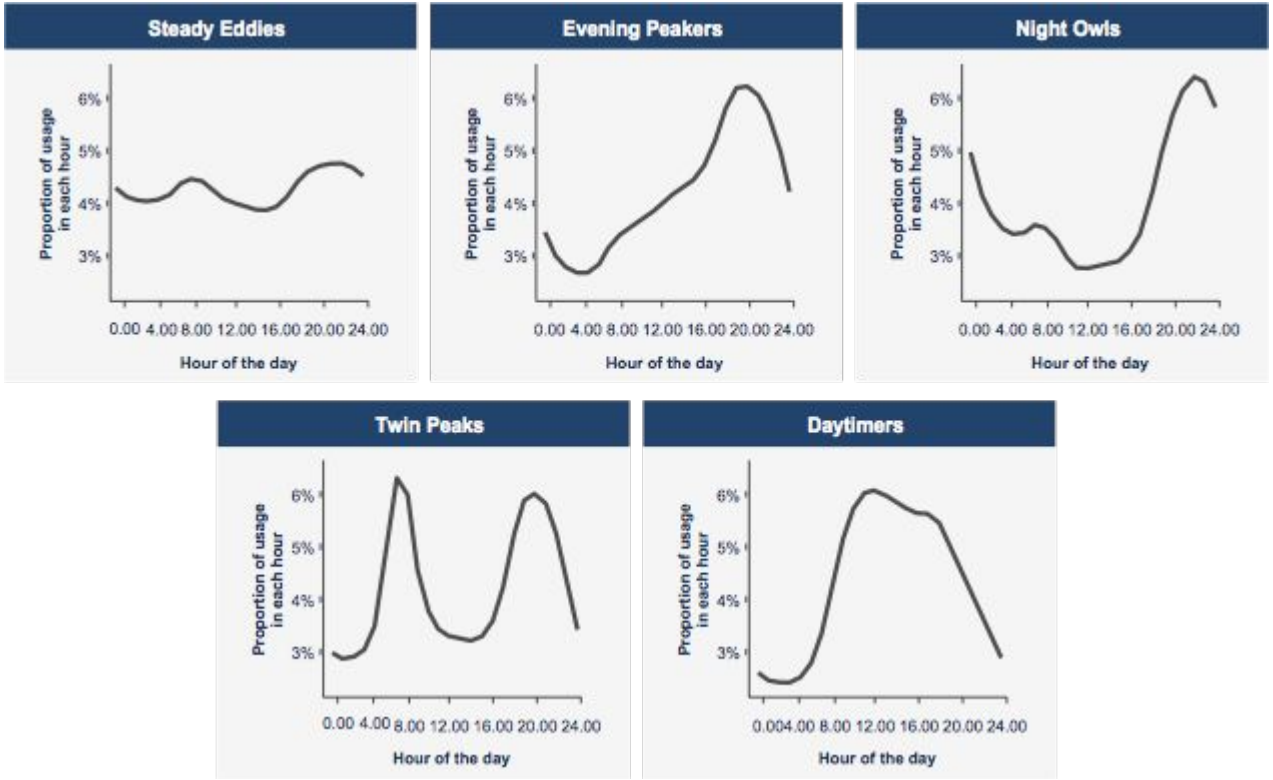
Average weekly electricity usage from 1000 households



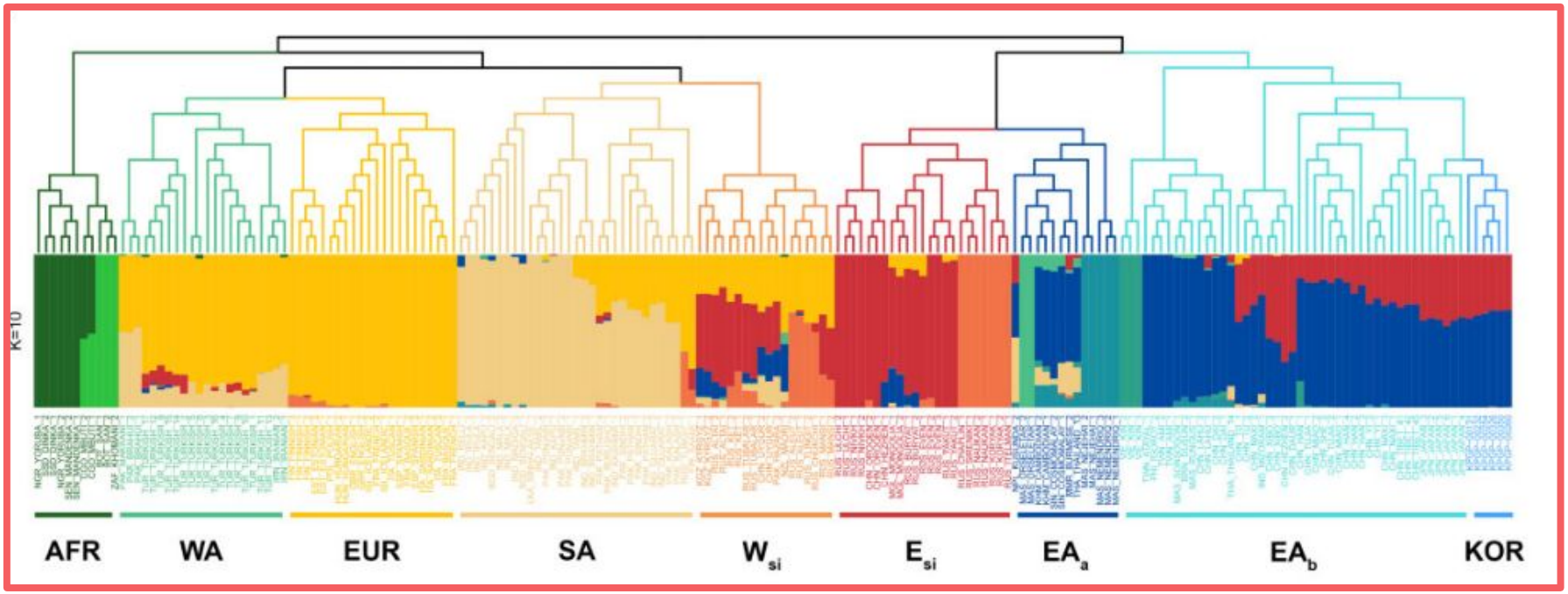
Clustered average weekly electricity usage (*K* = *5*)



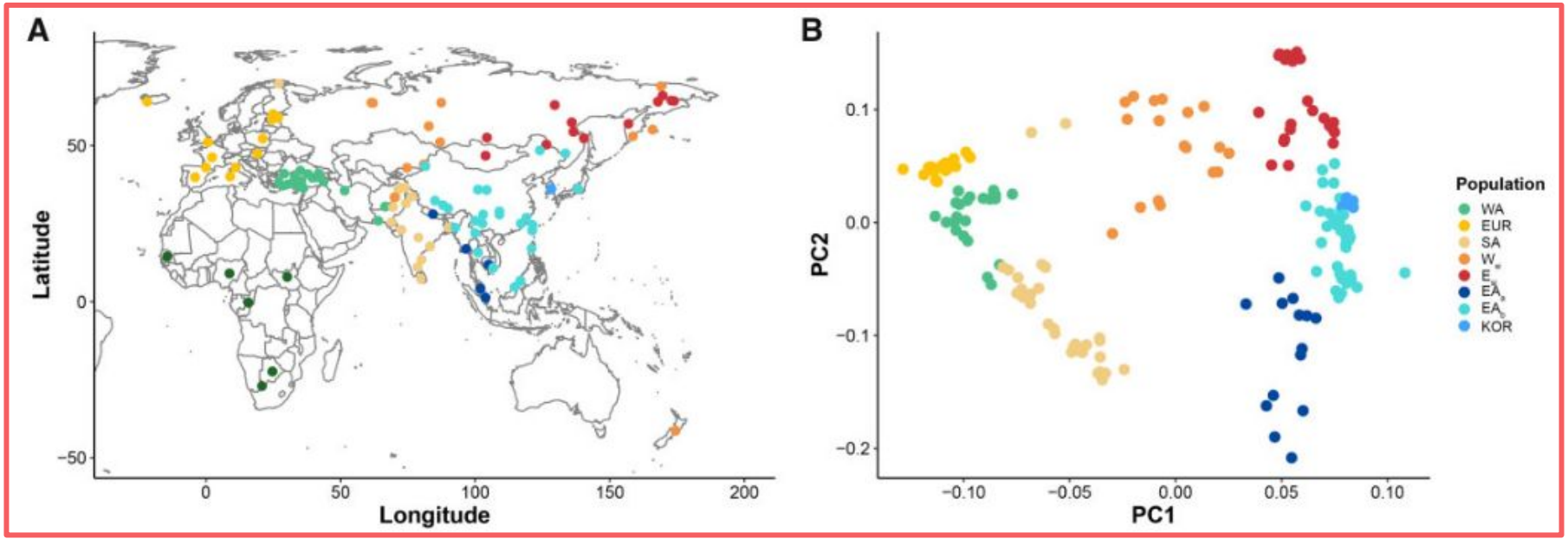
Application 2: Electricity Usage Trends



Application 2: Genomic Data

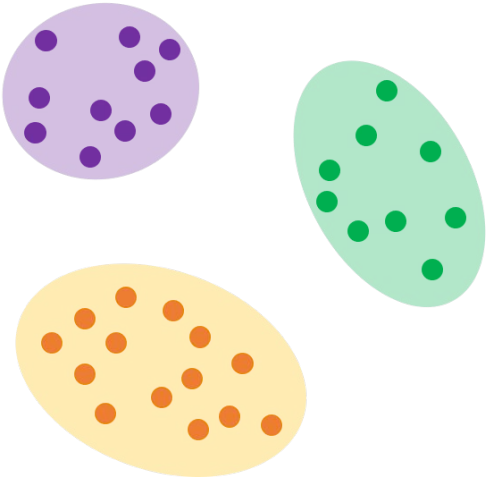


Application 3: Genomic Data



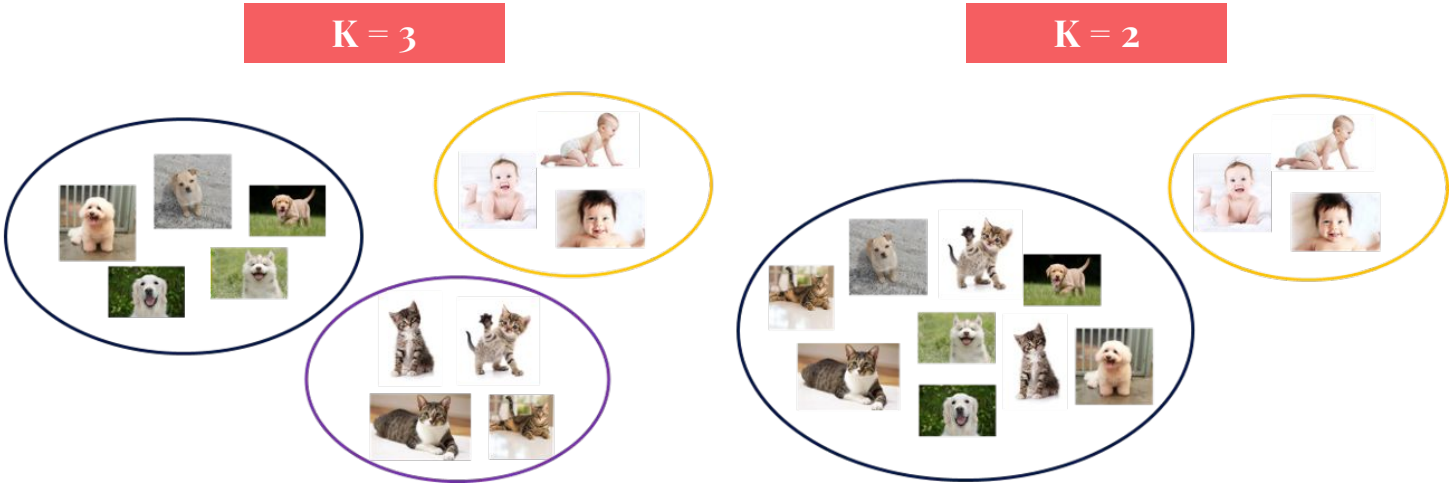
Partitional Clustering

- Nonoverlapping groups:
  - No object can be a member of more than one cluster,
  - Every cluster must have at least one object.
- Need to specify the number of clusters, K.
- Partitional clustering strengths:
  - Work well when clusters have a **spherical** or **ellipsoidal shape**.
  - They're **scalable** with respect to algorithm complexity.
- Partitional clustering weaknesses:
  - They're not well suited for clusters with **complex shapes**.
  - Required to define the number of clusters
- Most popular algorithm: **K-means**.

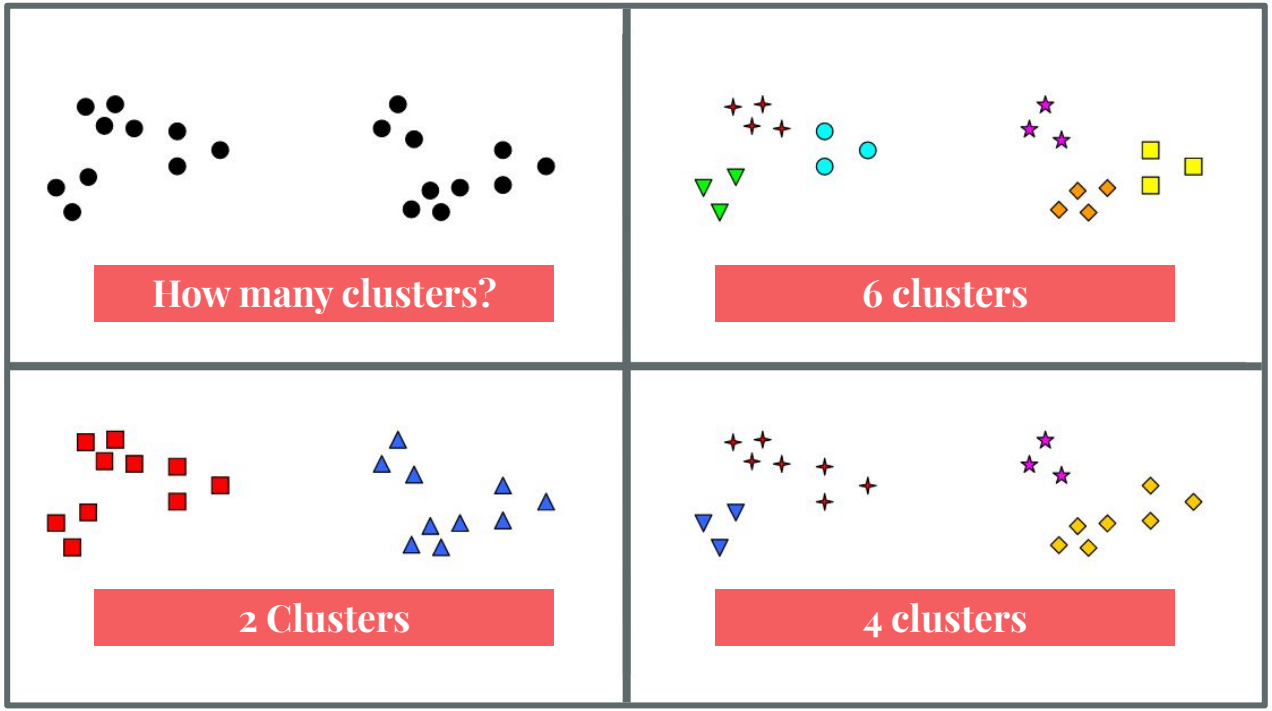


Number number of clusters

- A *huge* drawback: how to find the number of clusters?
- Typical solution:
  - Try a bunch of K's
  - Pick the best based on some criterion

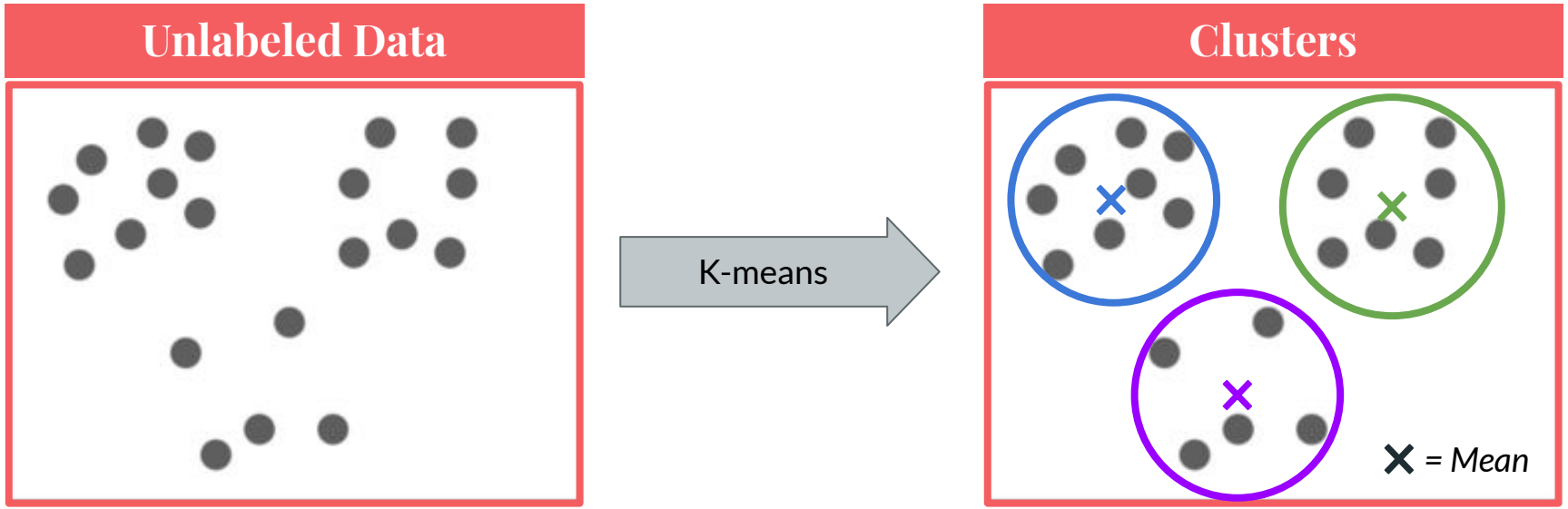


Number number of clusters



The K-means Problem

- In K-means, we want to:
  - Find K points (named **centers** or **means**)
  - Partition X by assigning each point to its nearest cluster mean.



The K-means Problem

- Say your dataset is

  $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$

- Assume that the number K is given, so need to find *K means*. Call them:

  $\mu_1, \mu_2, \ldots, \mu_K$

- Each mean represents a cluster. Call them

  $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$

- The goal is to find clusters that minimize the Within Cluster Sum of Squares:

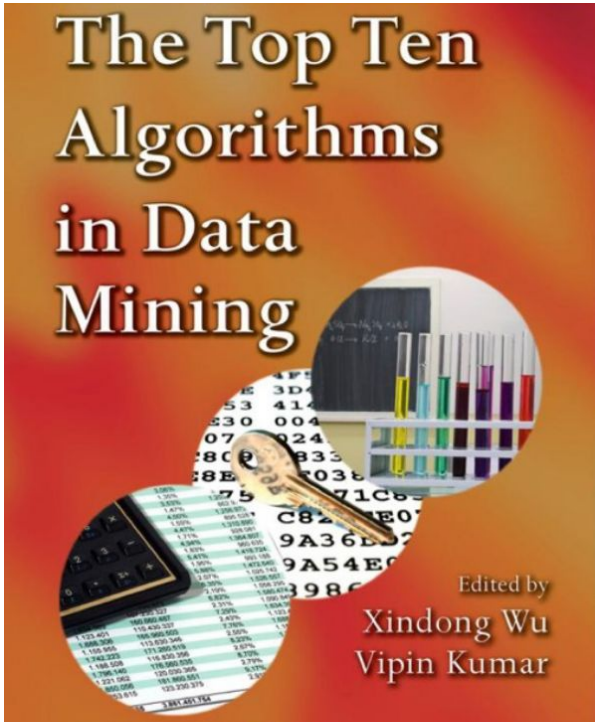
$$WCSS = \sum_{k=1}^{k} \sum_{x \in \mathcal{C}_k} \|x_i - \vec{\mu}_k\|^2$$

K-means Problem and K-means Algorithm

- K = 1 and K = *n* are easy special cases:
  - For K = 1: one mean, the dataset mean
  - For K = *n*: each point is its own mean
- For any other K, we can use simple iterative algorithm works quite well:

The K-means algorithm

- It was voted among the top-10 algorithms in data mining.
- It is easy to implement and runs very fast.



The K-means algorithm

1. Randomly pick K initial cluster means

   $\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_K^{(0)}$

2. The initial clusters $\mathcal{C}_1^{(0)}, \mathcal{C}_2^{(0)}, \ldots, \mathcal{C}_K^{(0)}$ are found via:

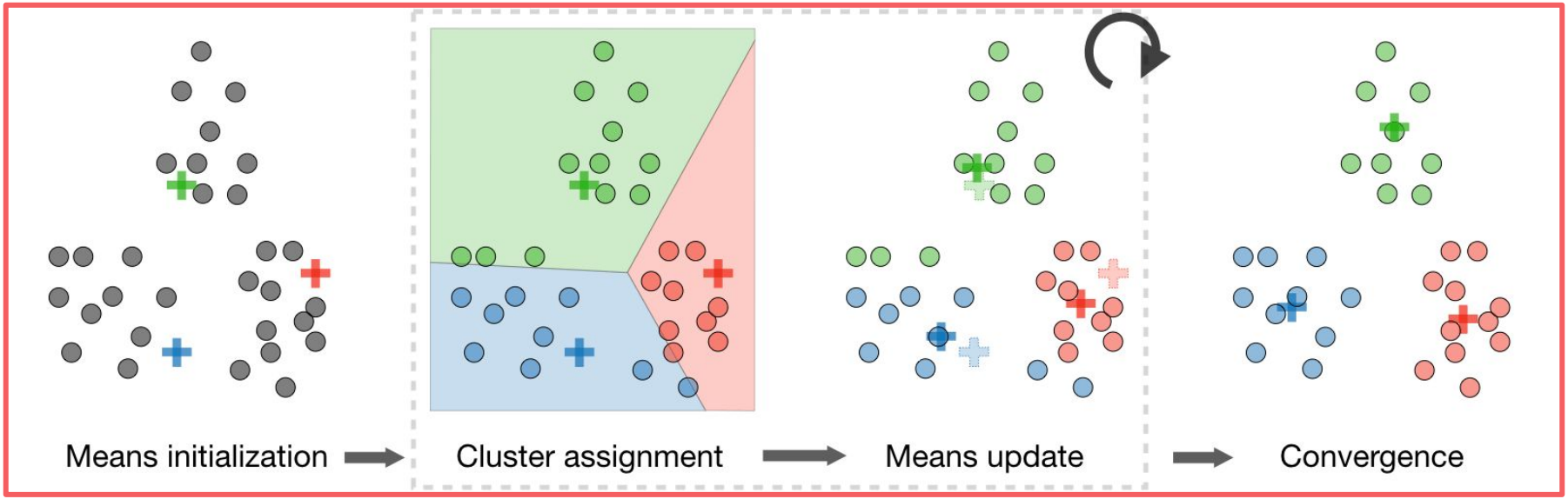
   $\mathcal{C}_k^{(0)} = \{x | \mu_k \text{ is the closest mean to } x\}$

3. The new means $\mu_1^{(1)}, \mu_2^{(1)}, \ldots, \mu_K^{(1)}$ are computed as the mean of each cluster:

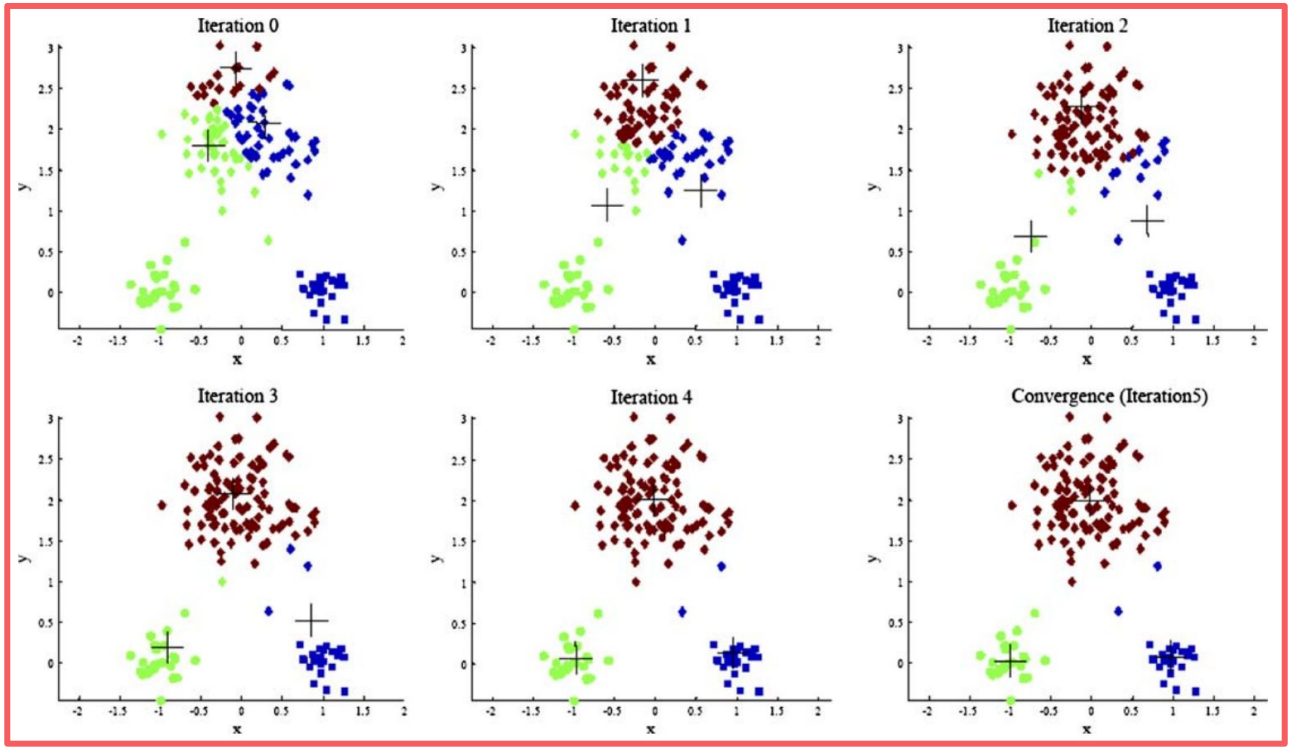
   $\mu_k^{(1)} = \frac{1}{|\mathcal{C}_k|} \sum_{x \in \mathcal{C}_k} x$

1. Repeat (go to step 2) until convergence.

The K-means algorithm visualized

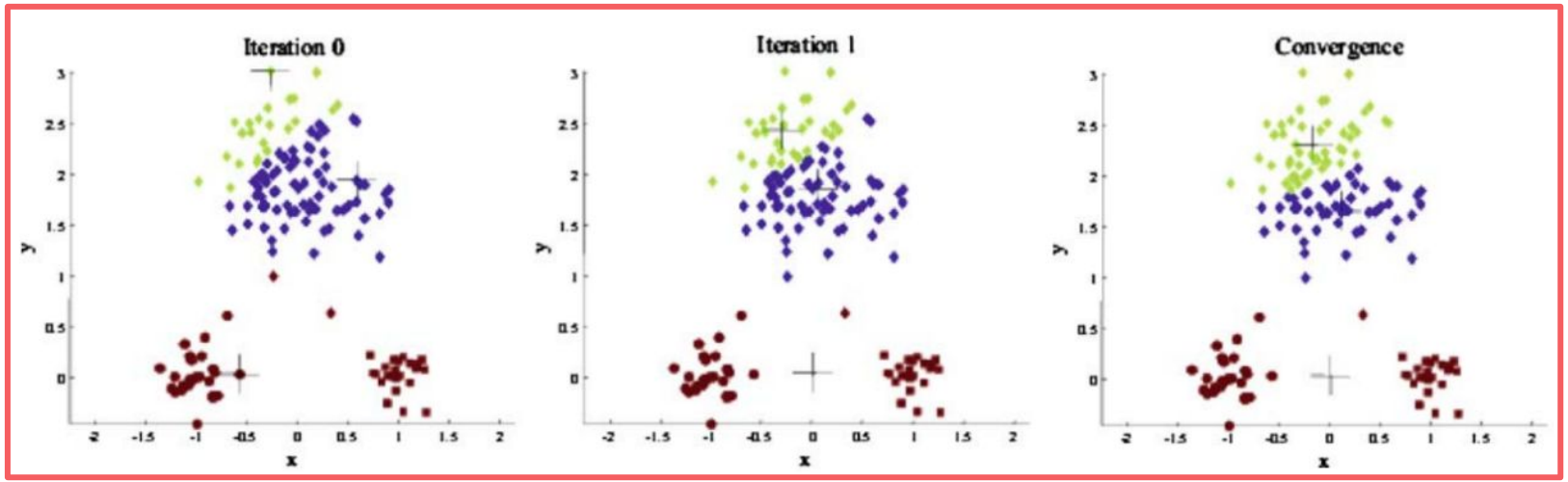


Sample K-means execution



Properties of the K-means algorithm

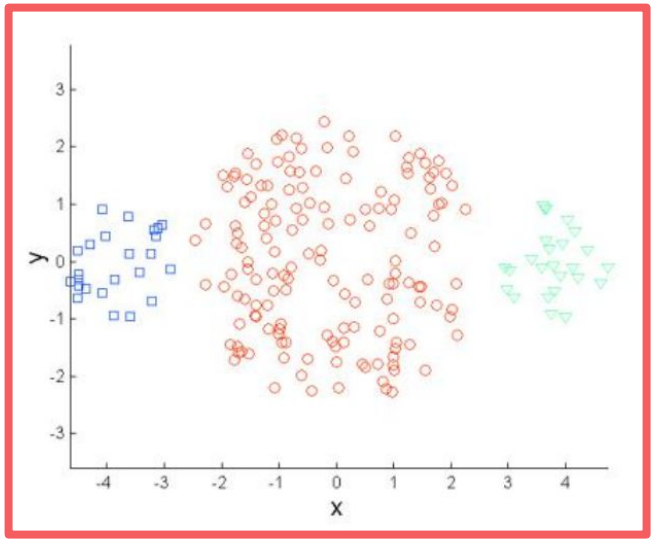
- Finds a **local** minimum of WCSS
- Often (not always) converges quickly
- The choice of initial means can have **large influence** in the result,



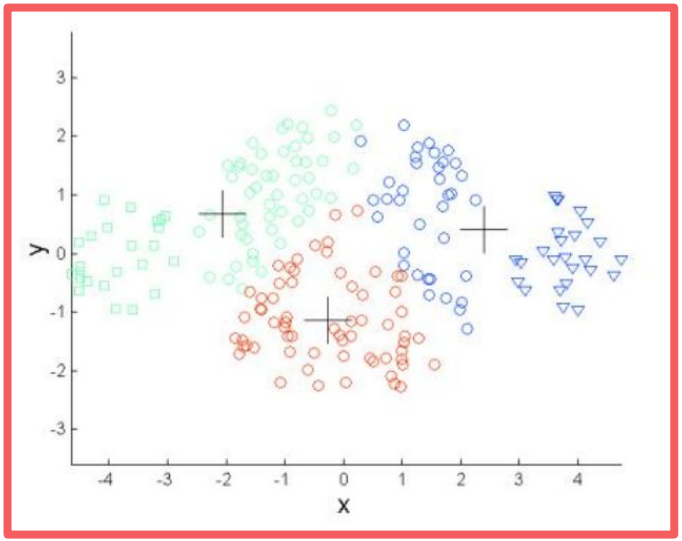
Limitations of the K-means algorithm

- Clusters with different sizes

Original Data



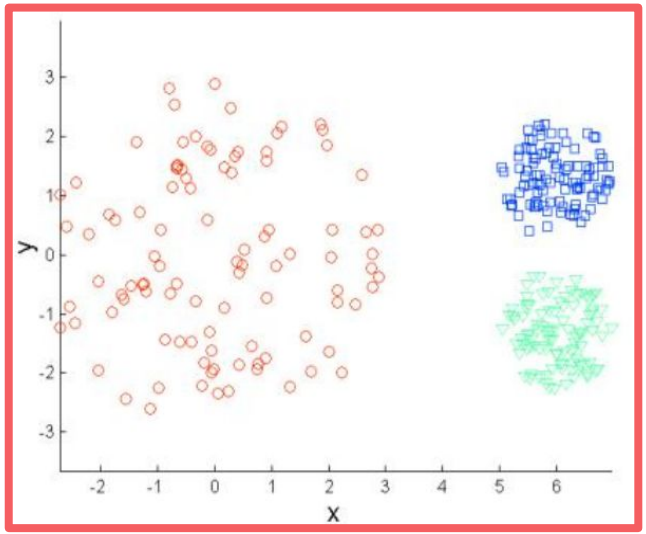
K-means (3 clusters)



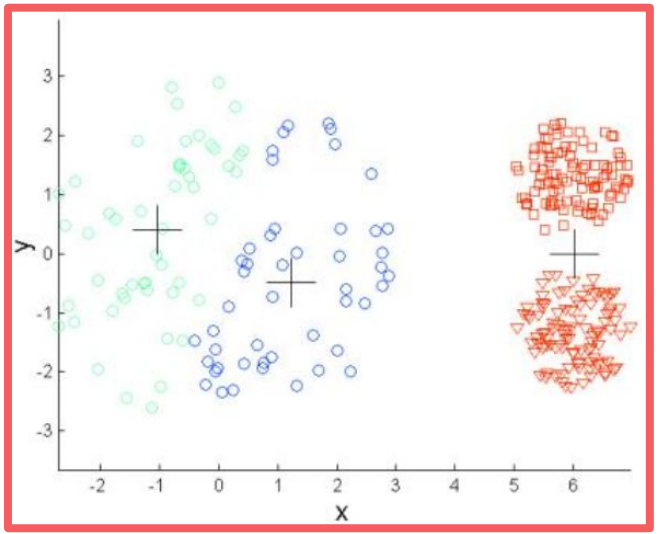
Limitations of the K-means algorithm

- Clusters with different densities

Original Data



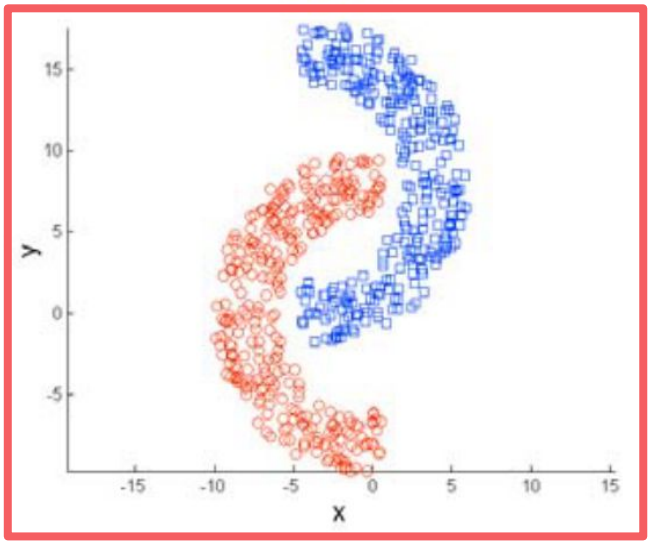
K-means (3 clusters)



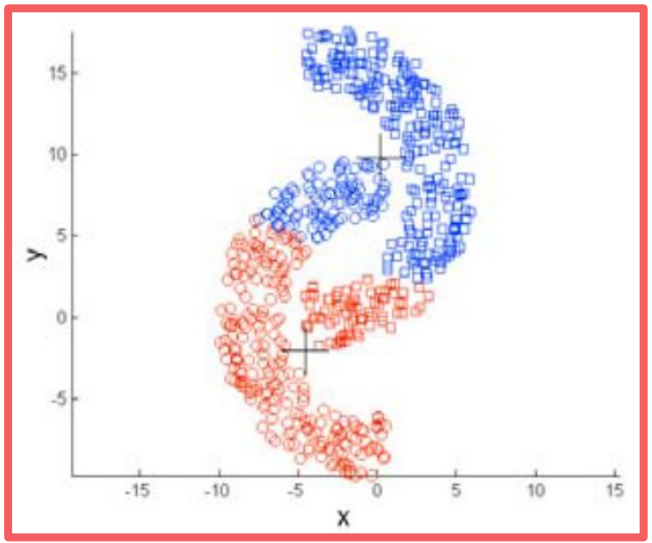
Limitations of the K-means algorithm

- Clusters with non-spherical shapes

Original Data

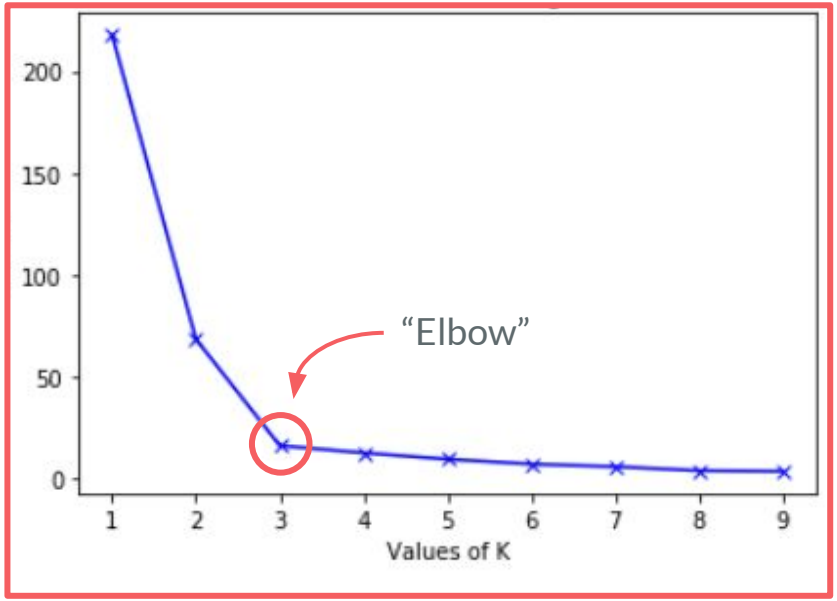


K-means (3 clusters)



The number of clusters

- How can we find the number of clusters K?
- We try various K and pick the best.
- We use the “elbow method”:
  - Plot the values of WCSS for each different K (looks like an arm)
  - Spot the “elbow” and the respective K.
- The graph before the “elbow” steeply declines, while the part after it is much smoother



Hands-on clustering

- First step: create a synthetic dataset, a blob dataset.

```
from sklearn.datasets import make_blobs
X, classes = make_blobs(n_samples=200, centers=4,
                        cluster_std=2, random_state=10)
```

- The dataset has
  - n = 200 points, all with p = 2 features.
  - 4 “classes” and each cluster has standard deviation of 2.
- Datapoints (real) classes:

```
print(classes)
```

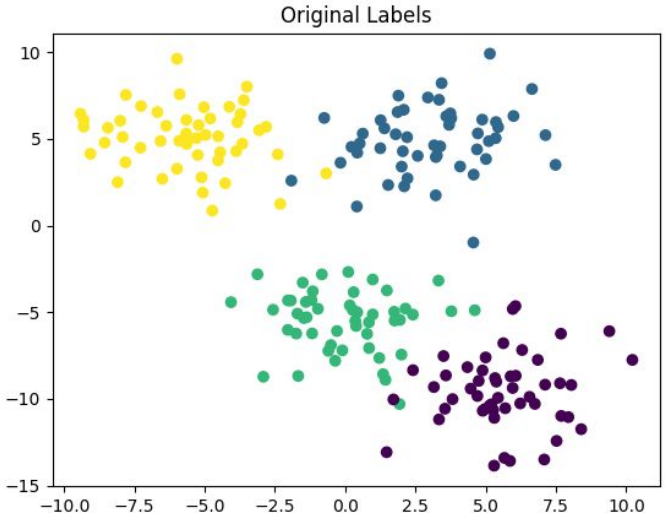
```
[1 2 3 3 0 2 3 1 3 3 2 0 1 2 2 3 1 1
 0 1 0 3 3 2 0 1 3 2 0 2 2 2 0 3 ...
```

Hands-on clustering

- To visualize it, we do:

```
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1],
            c=classes)
plt.title("Original Labels")
plt.show()
```

- It colors the points according to their classes.
- Here's the result:



Hands-on clustering

- Second step: perform K-means!

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
```

- The object "`kmeans`" comes with some info. Run

```
print(kmeans.n_iter_)
print(kmeans.cluster_centers_)
print(kmeans.labels_[:5])
print(kmeans.inertia_)
```

```
4
[[-0.02473046 -5.41927645]
 [ 3.07364878  4.89283159]
 [ 5.5474409  -9.60181246]
 [-5.65715189  5.03794162]]
[1 0 3 3 2]
1394.8755225665177
```

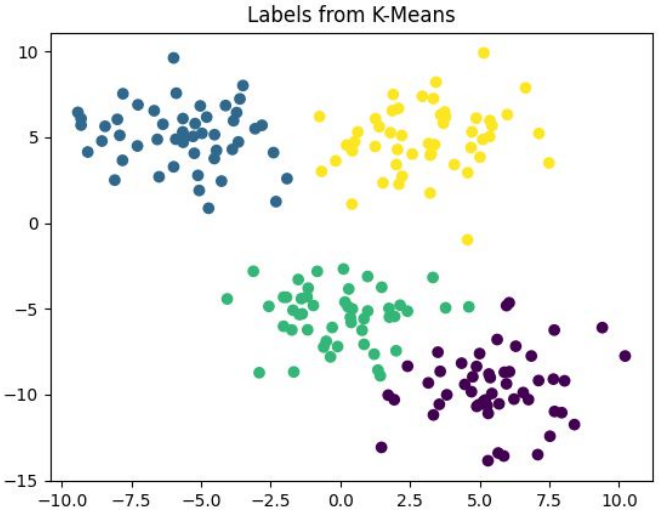
- Inertia is the same as we called WCSS (Within Cluster Sum of Squares)

Hands-on clustering

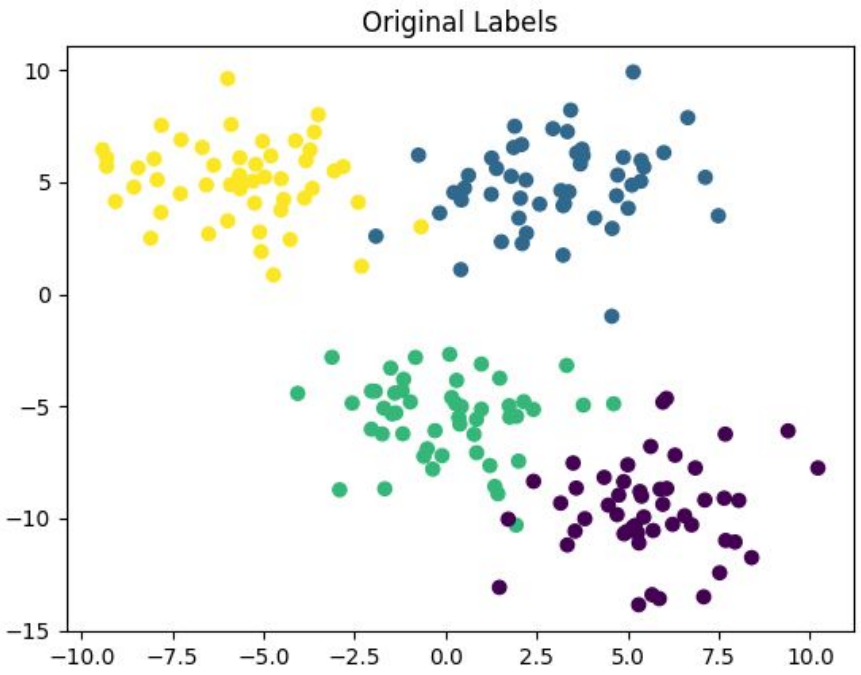
- To visualize it, we do:

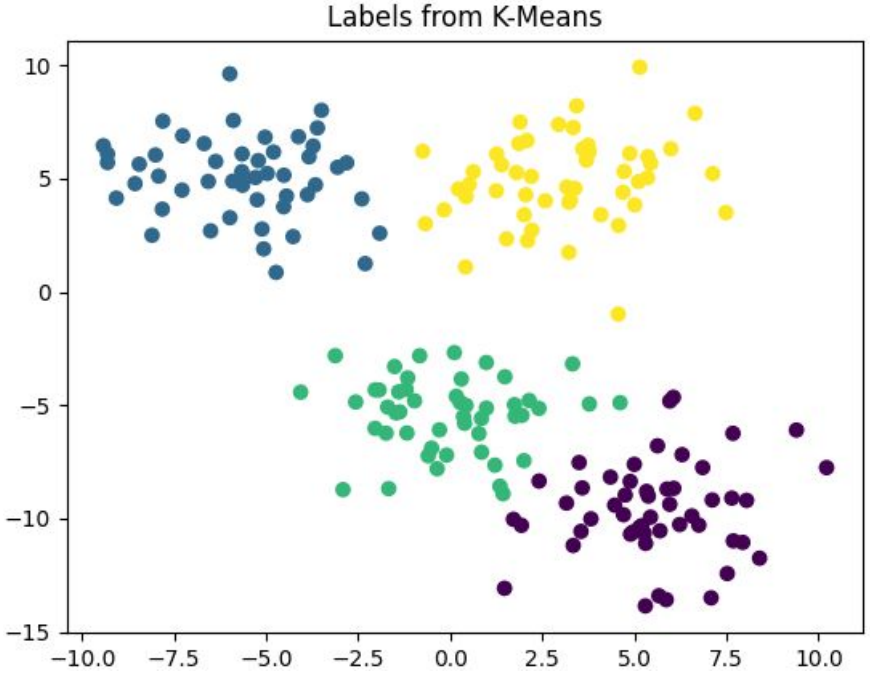
```
plt.scatter(X[:, 0], X[:, 1],
            c=kmeans.labels_)
plt.title("Labels from K-Means")
plt.show()
```

- Here's the result:



Hands-on clustering





Hands-on clustering

- The full code:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X, classes = make_blobs(n_samples=200, centers=4,  cluster_std=2, random_state=10)

plt.scatter(X[:, 0], X[:, 1], c=classes)
plt.title("Original Labels")
plt.show()

kmeans = KMeans(n_clusters=4)
kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
plt.title("Labels from K-Means")
plt.show()
```

Hands-on clustering: Your turn

1. Check the WCSS (“ `.inertia_` ”) of K-means for different values of “`n_clusters`”.
   a. Try all K from 1 to 11 and plot their respective WCSS.
   b. Try to localize the “elbow” of that graph.
2. Rerun the code you just wrote for different parameters in make_blobs
   a. Different number of samples
   b. Different number of clusters and their standard deviation.
   c. Different random state
3. Rerun the code you just wrote for the Moons dataset:

```
from sklearn.datasets import make_moons
X, classes = make_moons(n_samples=200, noise=0.05)
```

Hands-on Clustering: The Iris Dataset

- We'll work with the popular **Iris dataset** for clustering and visualization.
- The dataset describes 50 samples for each of three Iris flower species:

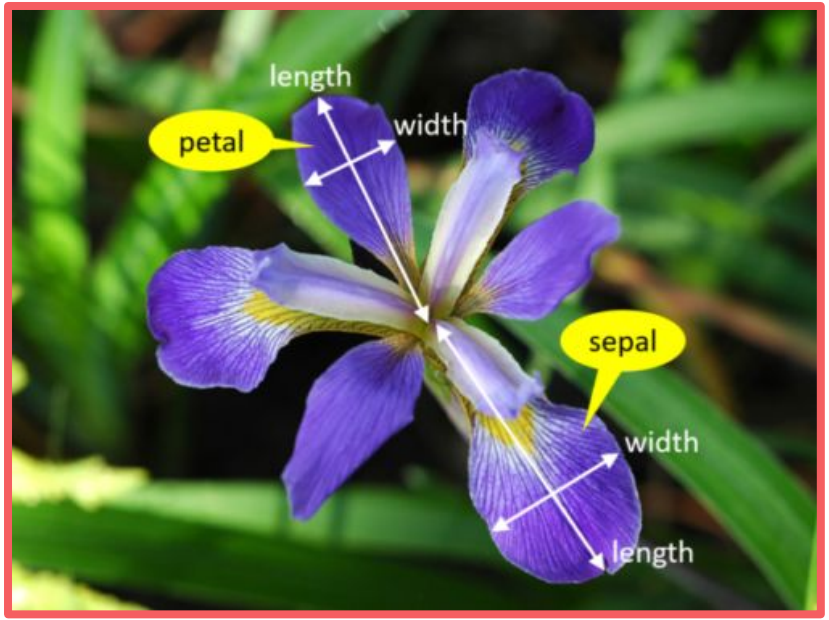
Iris setosa

Iris versicolor

Iris virginica

Hands-on Clustering: The Iris Dataset

- Each sample (flower) has p = 4 features (all in centimeters):
  - Sepal length,
  - Sepal width,
  - Petal length
  - Petal width.
- The **sepals** are the larger outer parts of each flower that protect the smaller inside **petals** before the flower buds bloom.



Hands-on Clustering: The Iris Dataset

- We'll use Python's `Seaborn` to download and visualize the Iris dataset.
- `Seaborn` is a library for making (beautiful) statistical graphics in Python.
  - Built on top of `matplotlib`
  - Integrated with `pandas` data structures.
- On your computers, run:

```
import seaborn as sns
iris = sns.load_dataset("iris")
print(iris)
```

Hands-on Clustering: The Iris Dataset

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
..	...	...	...	...	...
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Hands-on Clustering: The Iris Dataset

- Let's learn some things about our data. Run:

```
print(iris.values)
```

You'll have a Numpy's array in `iris.values`.

- Next run:

```
print(iris.values[:,0:3])
```

This is our **X**, our datapoints.

```
print(iris.values[:,-1])
```

These are the class of each point

- The classes are ordered: first 50 points are "`setosa`", then there are 50 "`versicolor`", and then 50 "`virginica`"

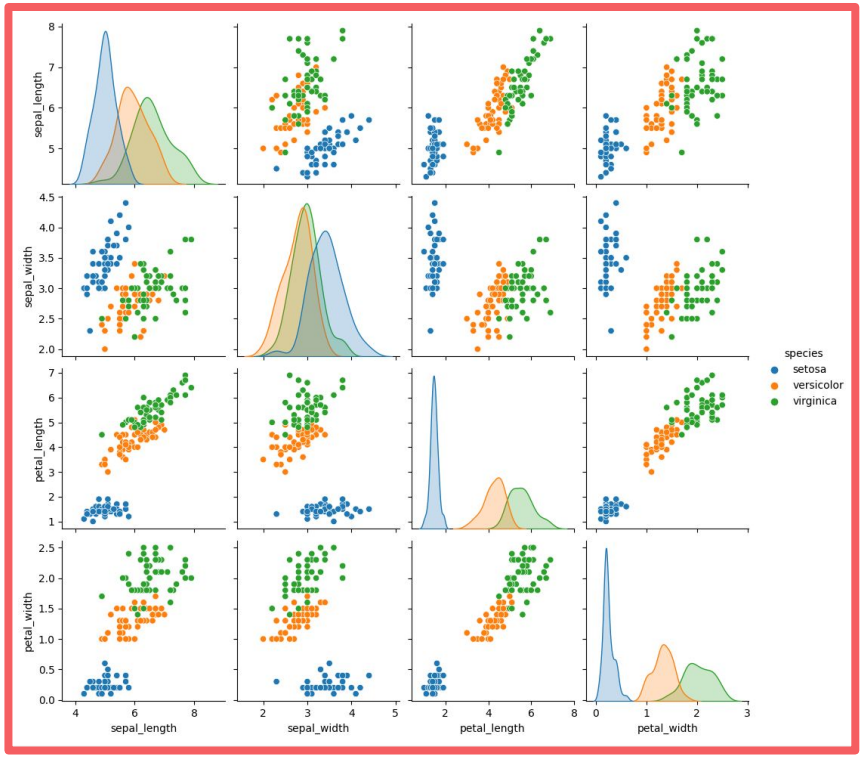
Hands-on Clustering: The Iris Dataset

- Our datapoints are in four dimensions! We can't visualize that.
- We then visualise pairs of features: `Seaborn`'s pairplot
- Append the following to the previous your code:

```
import matplotlib.pyplot as plt
sns.pairplot(data=iris, hue='species')
plt.show()
```

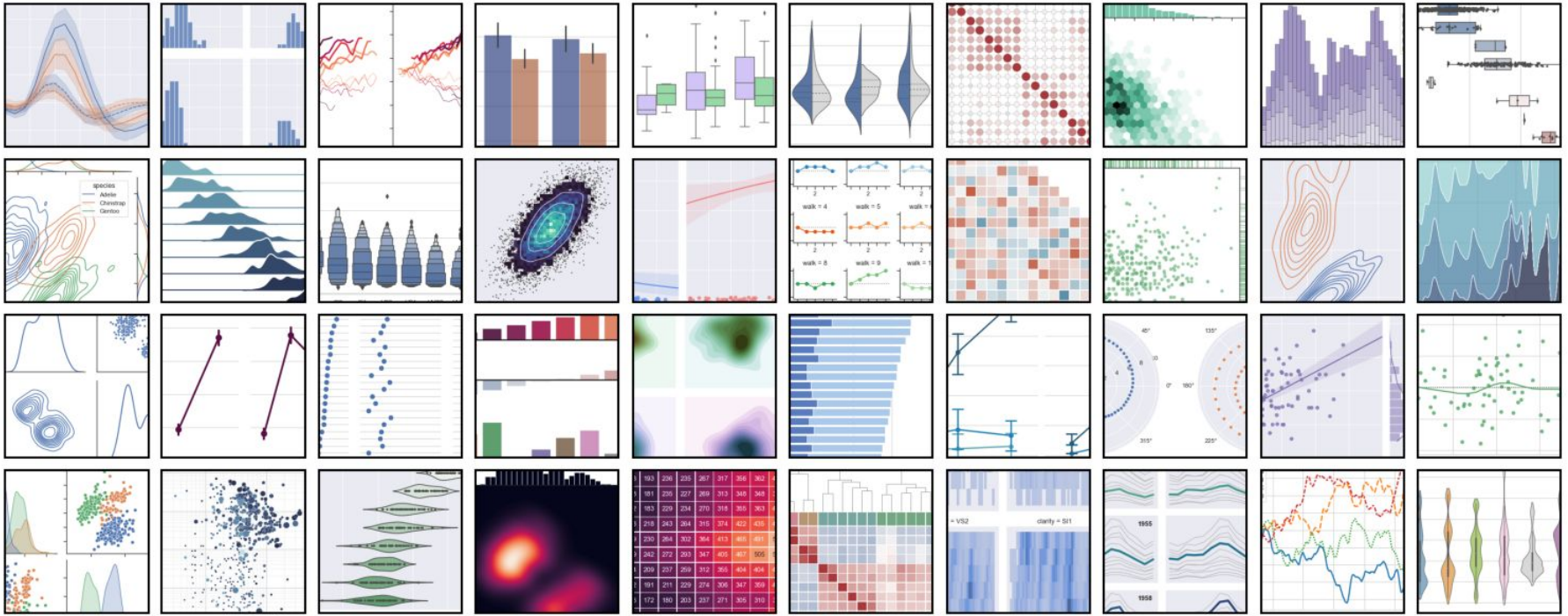
- In the above we have:
  - `data`: The a `Pandas`' DataFrame containing the data to plot.
  - `hue`: The DataFrame column that's used to determine colors of the plotted data (we'll color the data by Iris species)

Hands-on Clustering: The Iris Dataset



- A lot of information!
  - Feature distributions along the diagonal
  - Scatter plots considering pairs of features as (x, y) pairs.
  - Each color is a different species.
- Some info you can learn about Iris:
  - Ranges of each feature.
  - Setosa is easily classified using `petal_length`.
  - All the scatter plots separate blue dots from the orange and green dots.
  - Separating Virginica and Versicolor is harder.

Diversion: Seaborn awesomeness



Hands-on Clustering: The Iris Dataset

- Back to *K*-means! First define our data set:

```
X = iris.values[:, 0:3]
```

- Then run K-means from Sklearn on it:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

- Check the clustering results via `.labels_` and `.cluster_centers_`

```
print(kmeans.cluster_centers_)
print(kmeans.labels_)
```

Hands-on Clustering: The Iris Dataset

- Let's check some generated labels:
  - For the "`setosa`" class:

```
print(kmeans.labels_[0:50])
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

  - For the "`virginica`" class:

```
print(kmeans.labels_[100:150])
```

```
[2 0 2 2 2 2 0 2 2 2 2 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 2 2 2
2
```

Hands-on Clustering: The Iris Dataset

- We'll use PCA to perform dimensionality reduction so we can plot the results.
- Compute the PCA of X with 2 PCs:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

- Then, run

```
X_pca = pca.transform(X)
X_centers = pca.transform(kmeans.cluster_centers_)
```

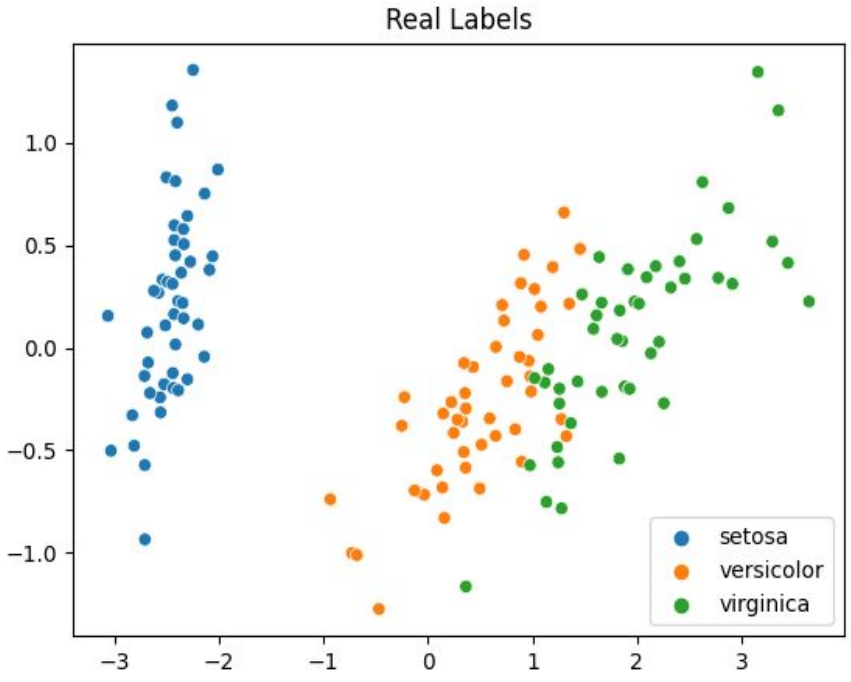
Hands-on Clustering: The Iris Dataset

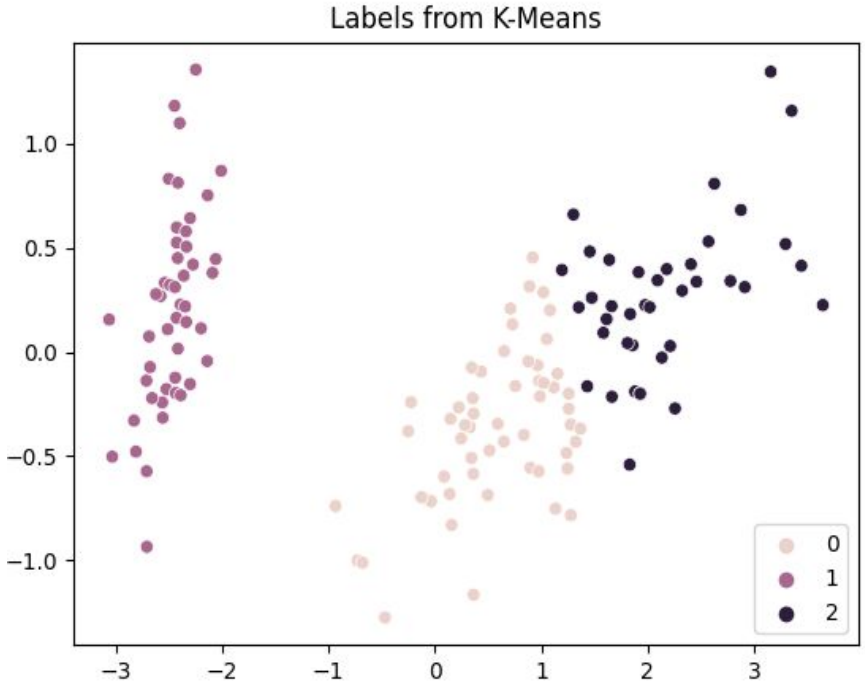
- Now compare the real labels to the K-means labels:

```
classes = iris.values[:, 4]
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=classes)
plt.title("Real Labels")
plt.show()
```

```
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1],
                hue=kmeans.labels_)
plt.title("Labels from K-Means")
plt.show()
```

Hands-on Clustering: The Iris Dataset

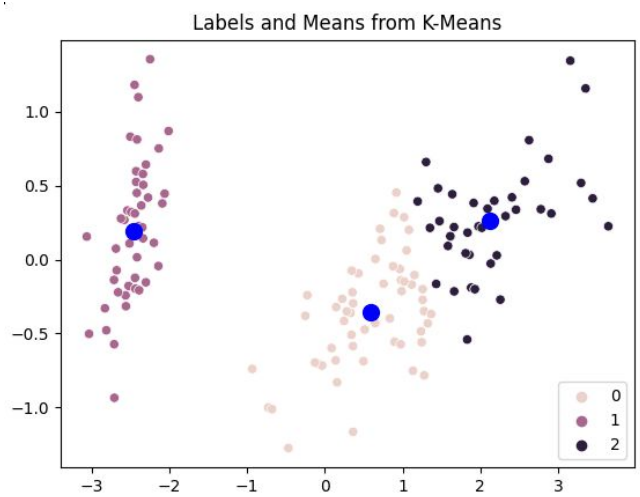




Hands-on Clustering: The Iris Dataset

- Now, we plot the labels and the means:

```
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1],
                hue=kmeans.labels_)
plt.scatter(X_centers[:, 0], X_centers[:, 1],
            s=100, c='b')
plt.title("Labels and Means from K-Means")
plt.show()
```



Obs: you could use "`sns.scatterplot`" for scatter plotting the means too

Hands-on Clustering: The Iris Dataset

- The full code:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

iris = sns.load_dataset("iris")

sns.pairplot(data=iris, hue="species")
plt.show()

X = iris.values[:, 0:3]
Classes = iris.values[:, 4]

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

```
pca = PCA(n_components=2)
pca.fit(X)

X_pca = pca.transform(X)
X_centers = pca.transform(kmeans.cluster_centers_)

sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1],
                hue=classes)
plt.title("Real Labels")
plt.show()

sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1],
                hue=kmeans.labels_)
plt.scatter(X_centers[:, 0], X_centers[:, 1],
            s=100, c='b')
plt.title("Labels and Means from K-Means")
plt.show()
```

Hands-on Clustering: The Penguins Dataset

- Now, it's your turn: do a similar analysis with the **penguins dataset**.

```
import seaborn as sns
penguins = sns.load_dataset("penguins")
```

- This dataset has information about 344 penguins of three species:
  - The island where they are found,
  - Their bill's length and depth,
  - Their flipper length and body mass,
  - Their sex.