

A diverse group of people are gathered around a large white table in a bright, modern office setting. They are all smiling and looking at various items on the table, including several laptops and numerous colorful charts, graphs, and sticky notes. The atmosphere is collaborative and positive. A semi-transparent blue banner is overlaid across the middle of the image, containing the text "Intro to ML".

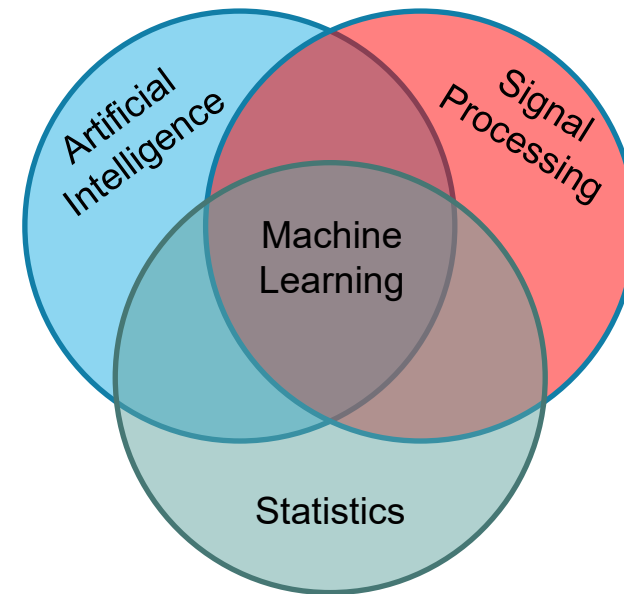
# Intro to ML

# What is ML?

- A field of study concerned with making quantitative inferences and predictions from data
- Uses statistical techniques and computational power to “learn” these inferences and predictions without being explicitly programmed
- A little more formally: an algorithm is said to learn from data if its performance improves as the amount of data increases

# Roots of machine learning

- ML theory and methodology emerged from three main areas: artificial intelligence, signal processing, and statistics
- Examples of contributions:
  - AI: Support vector machines, neural networks
  - SP: information theory, detection and estimation theory
  - Stats: regression, classification, a probabilistic framework for handling uncertainty and randomness in the data



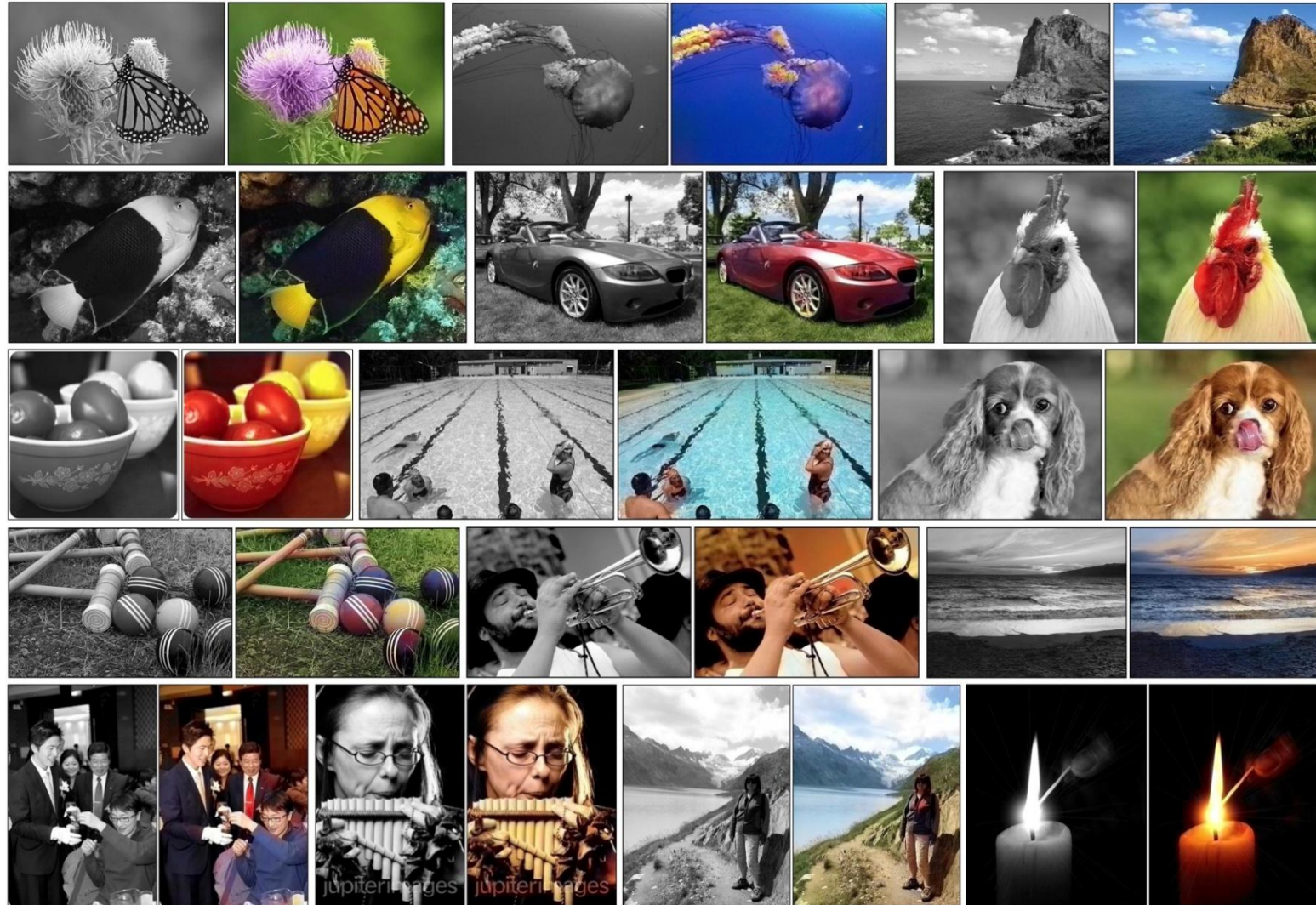


# Classical Statistics vs ML

- What are the differences between classical statistics and ML?
  - There is an overlap of methods
- Somewhat broadly, classical statistics focuses heavily on data modeling
- ML typically focuses on the endgame/output (e.g., prediction)
  - A model is useless if it doesn't give a good prediction
- Many ML methods are based on statistical models
- See Breiman (2001) for one viewpoint on the two fields: “Statistical Modeling: The Two Cultures”

# Success of Machine Learning/AI

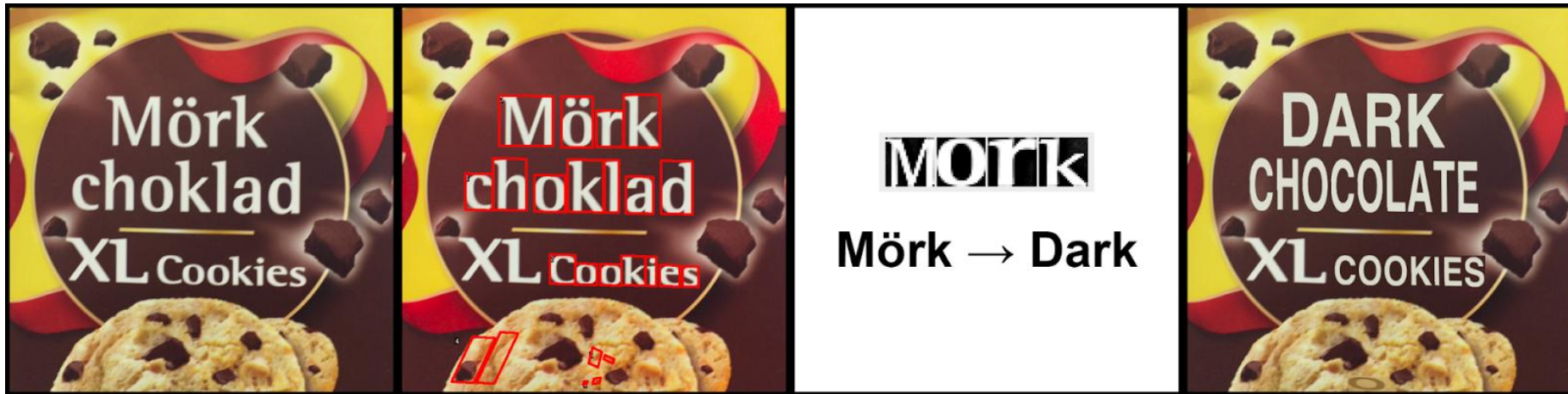
Image colorization (Zhang et al., 2016)



# Success of Machine Learning/AI

Real-time visual translation on smartphones

1. Find the letters
2. Recognize the letters
3. Translate
4. Render the translation in the same style



Google blog, 2015

# Success of Machine Learning

Automated programming with ChatGPT



Find the bug with this code:

```
...  
for (var i = 0; i < 5; i++) {  
  setTimeout(() => console.log(i), 1000)  
}  
...
```



The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.



To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

```
Copy code  
  
for (var i = 0; i < 5; i++) {  
  // Create a new variable and pass its value to setTimeout  
  const value = i;  
  setTimeout(() => console.log(value), 1000)  
}
```

This code will print the numbers 0 to 4 to the console, as expected. Alternatively, the `let` keyword can be used in place of `var` to declare the loop variable, which will automatically create a new variable for each iteration of the loop and avoid the need to create a new variable manually. This is a common pattern for avoiding closure-related bugs in JavaScript.



# Success of Machine Learning

AI generated art



Best Dalle2 Pics  
@Dalle2Pics

...

A raccoon playing tennis at Wimbledon in the 1990s  
[#dalle2](#) [#dalle](#)



3:33 AM · May 14, 2022



# Success of Machine Learning

## Automatic game playing

Alpha Go Zero

 **nature**  
International journal of science

Access provided by Yale University

  Altmetric: 2188 Citations: 1 [More detail >>](#)

Article

### Mastering the game of Go without human knowledge

David Silver , Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

*Nature* **550**, 354–359 (19 October 2017)  
doi:10.1038/nature24270  
[Download Citation](#)

Received: 07 April 2017  
Accepted: 13 September 2017  
Published online: 18 October 2017

[Computational science](#)  
[Computer science](#) [Reward](#)

Alpha Zero

arXiv.org > cs > arXiv:1712.01815 [Search or](#)  
(Help | Advanced Search)

Computer Science > Artificial Intelligence

### Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis

(Submitted on 5 Dec 2017)

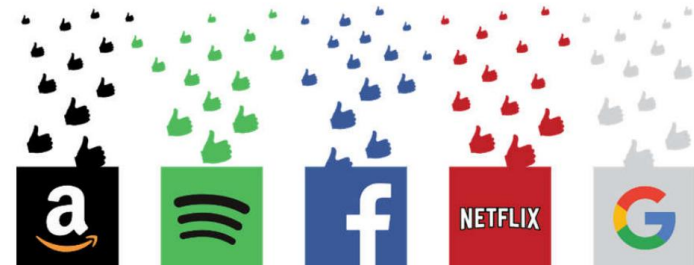
The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go, by tabula rasa reinforcement learning from games of self-play. In this paper, we generalise this approach into a single AlphaZero algorithm that can achieve, tabula rasa, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

Subjects: **Artificial Intelligence (cs.AI)**; Learning (cs.LG)  
Cite as: [arXiv:1712.01815 \[cs.AI\]](#)  
(or [arXiv:1712.01815v1 \[cs.AI\]](#) for this version)

**Submission history**  
From: David Silver [[view email](#)]  
[v1] Tue, 5 Dec 2017 18:45:38 GMT (272kb,D)

# Other Machine Learning Applications

- Chatbots (NLP)
- Fraud detection
- Route planning
- Search engine refinement
- Face recognition/detection
- Credit risk assessment
- Financial market prediction
- Medical diagnosis
- Personalized medicine
- Electricity demand forecasting
- Spam filtering
- Collision avoidance systems
- Speech synthesis/analysis (e.g. Siri, Alexa)
- Recommender systems (e.g. Netflix, Spotify)



# MACHINE LEARNING TYPES

## ■ Supervised

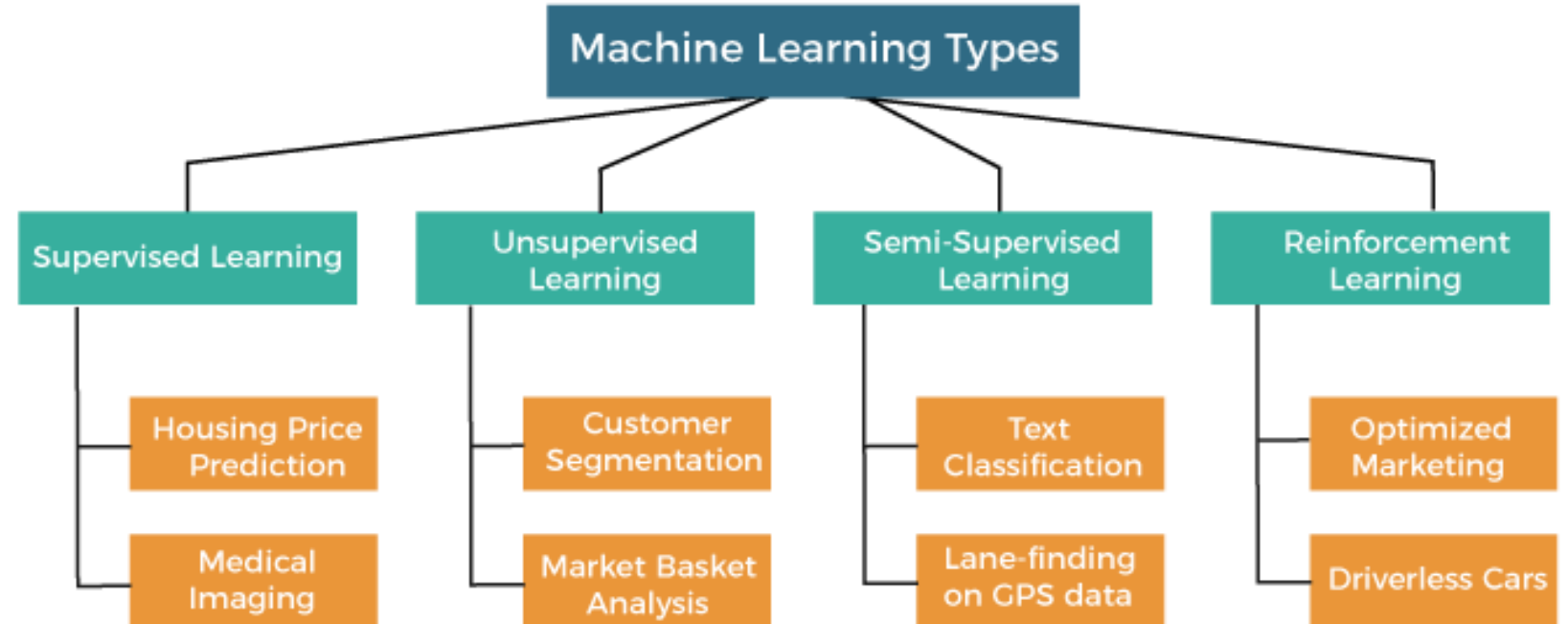
- Classification
- Regression

## ■ Unsupervised

- Clustering
- Dimensionality Reduction

## ■ Semi-supervised

## ■ Reinforcement



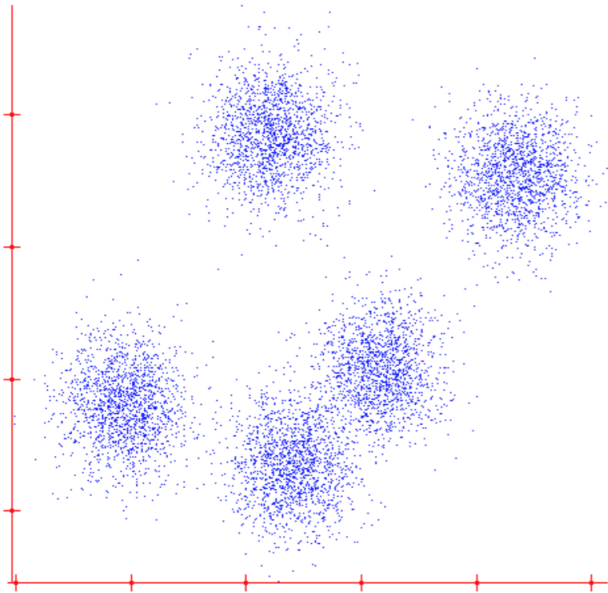


# QUIZ

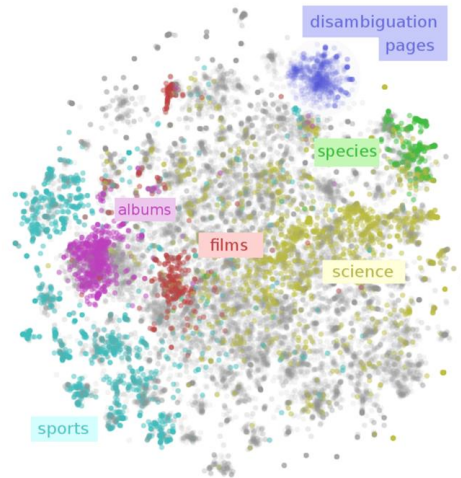
Of the following examples, which would you address using an unsupervised learning algorithm? (Check all that apply.)

- ☐ Given email labeled as spam/not spam, learn a spam filter.
- ☒ Given a set of news articles found on the web, group them into set of articles about the same story.
- ☒ Given a database of customer data, automatically discover market segments and group customers into different market segments.
- ☐ Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.

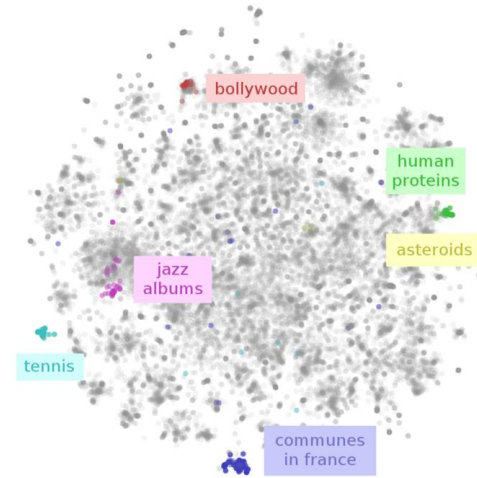
# CLUSTERING DATA: GROUP SIMILAR THINGS



**Large Clusters**



**Small Clusters**



wikipedia  
articles



[Goldberger et al.]

# APPLICATIONS OF CLUSTER ANALYSIS


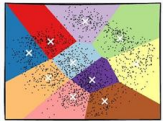
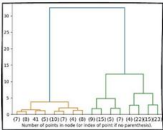
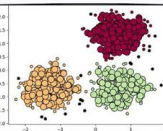
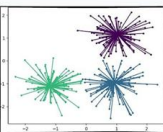
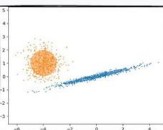
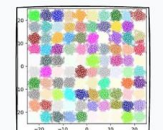
- **Customer Segmentation:** In marketing, clustering helps to group customers based on behavior, preferences, or demographics.
  - This segmentation enables personalized marketing, targeted promotions, and better customer service.
- **Document Clustering:** often used in natural language processing to organize large text corpora.
  - It groups similar documents, which is useful for topic modeling, or summarizing news articles.
- **Anomaly Detection:** Can identify outliers in data.
  - Ex: in network security, clustering can detect unusual patterns of behavior that may indicate a cyberattack or fraud.



## MORE APPLICATIONS OF CLUSTER ANALYSIS

- **Social Network Analysis:** Clustering helps to find communities within social networks
  - by grouping users with similar connections, interests, or behaviors.
- **Genomics:** clustering is used to classify genes with similar expression patterns,
  - helping in the discovery of gene functions and understanding of diseases.
- **Urban Planning and Geographic Analysis:** Helps in analyzing geographic data
  - Ex: identifying areas with similar land use, demographic profiles, or crime rates, which aids in urban planning.

# MAJOR TYPES OF CLUSTERING ALGORITHMS

| <div> <div>6 Types of Clustering Algorithms in Machine Learning</div> <div>  <a href="http://blog.DailyDoseofDS.com">blog.DailyDoseofDS.com</a> </div> </div> |                        |   |  |
|--|------------------------|---|--|
| Clustering Algorithm Type  | Clustering Methodology | Algorithm(s)  |  |
|   | Centroid-based         | Cluster points based on proximity to centroid                                   | KMeans<br>KMeans++<br>KMedoids                       |
|   | Connectivity-based     | Cluster points based on proximity between clusters                              | Hierarchical Clustering (Agglomerative and Divisive) |
|   | Density-based          | Cluster points based on their density instead of proximity                      | DBSCAN<br>OPTICS<br>HDBSCAN                          |
|    | Graph-based            | Cluster points based on graph distance  | Affinity Propagation<br>Spectral Clustering          |
|   | Distribution-based     | Cluster points based on their likelihood of belonging to the same distribution. | Gaussian Mixture Models (GMMs)                       |
|   | Compression-based      | Transform data to a lower dimensional space and then perform clustering         | BIRCH  |

Our focus:

**1. Centroid-based:** Cluster data points based on proximity to centroids.

**2. Connectivity-based:** Cluster points based on proximity between clusters.

**3. Density-based:** Cluster points based on their density. It is more robust to clusters with varying densities and shapes than centroid-based clustering.

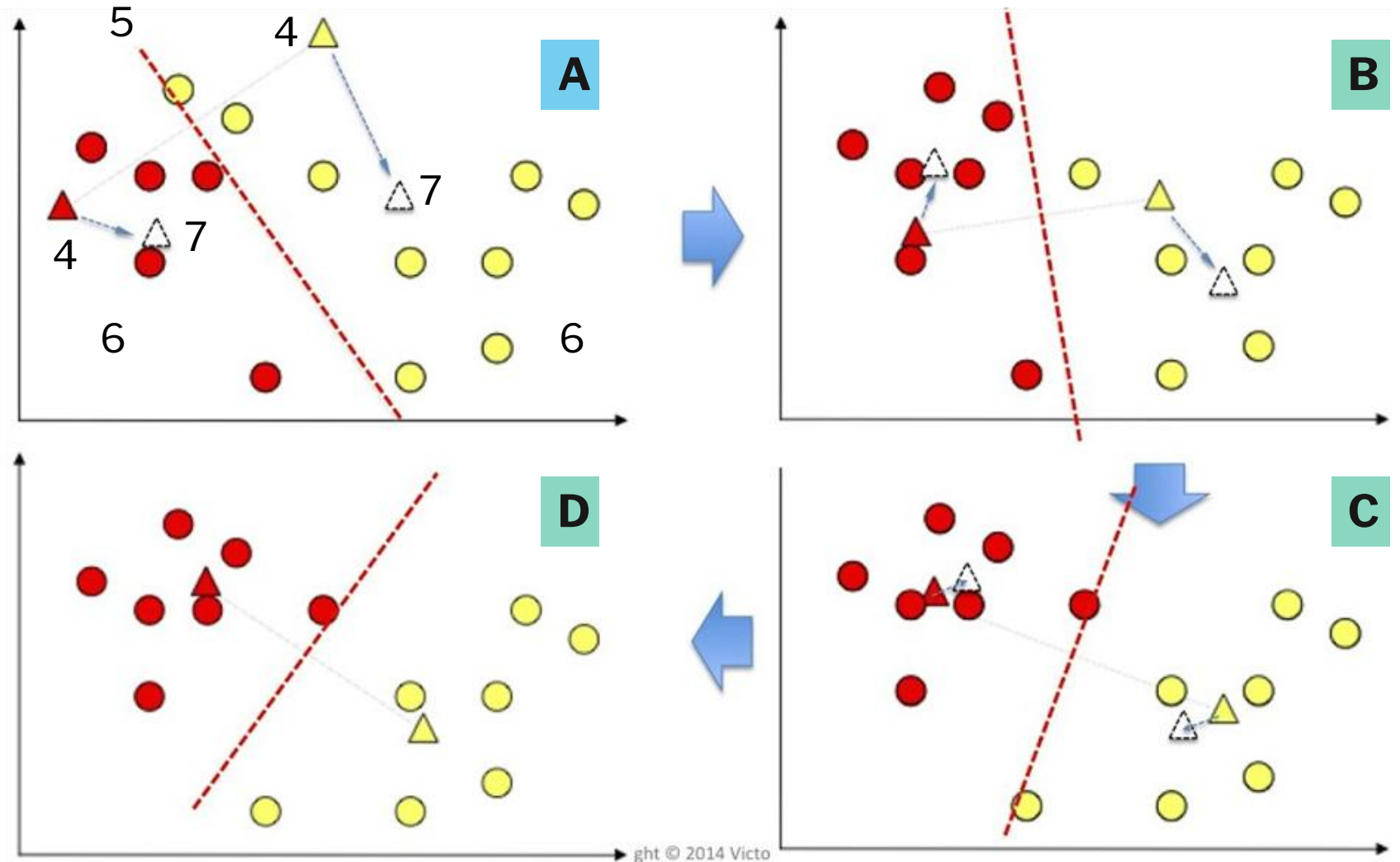
**4. Graph-based:** Cluster points based on graph distance.

**5. Distribution-based:** Cluster points based on their likelihood of belonging to the same distribution. Gaussian Mixture Model in one example.

**6. Compression-based:** Transform data to a lower dimensional space and then perform clustering

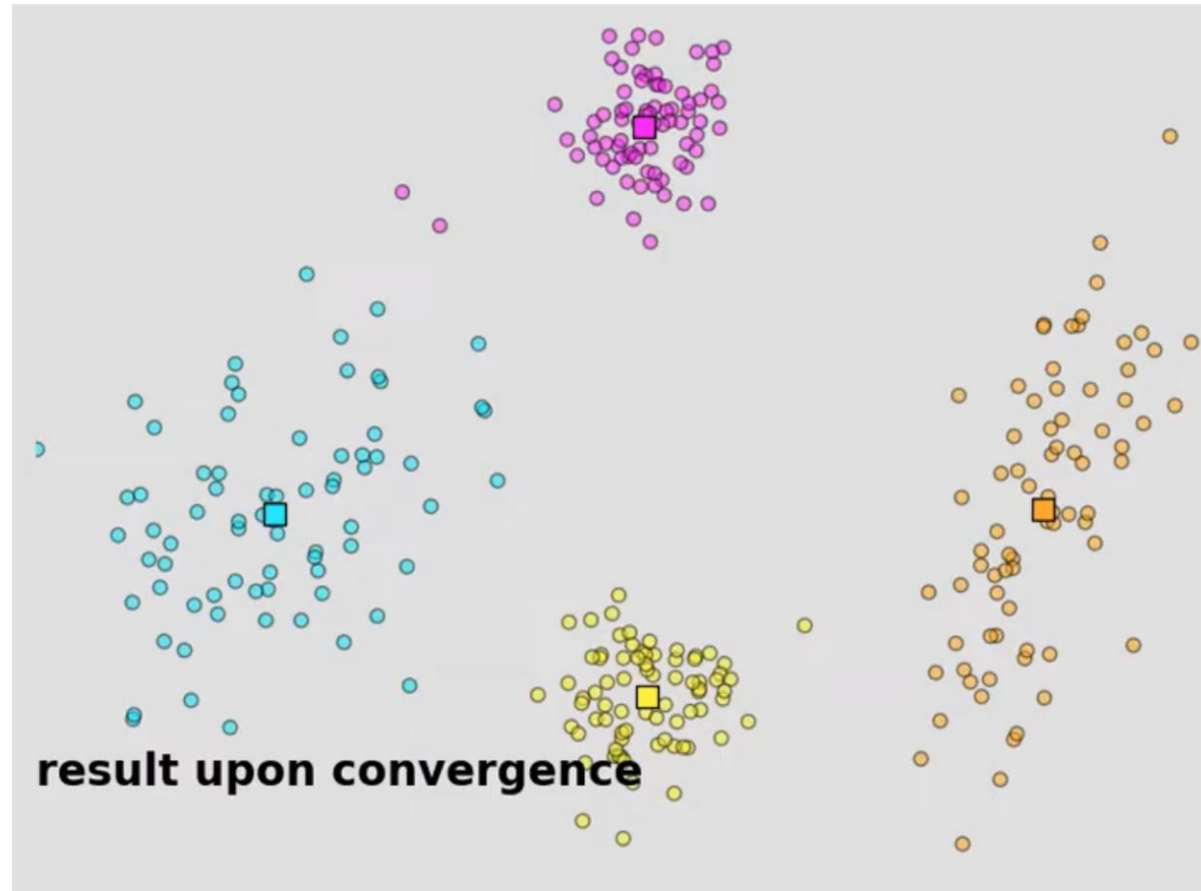
# K-MEANS CLUSTERING

1. Standardize samples
2. Choose # clusters (K)
3. Choose # iterations
4. Randomly place initial centroids (Fig. A)
5. Calculate distance to centroid (Fig. A)
6. Assign samples with the nearest centroid color (Fig. A)
7. Move centroids to the mean of cluster (Fig. A)
8. Repeat 5-7 (Figs. B, C, D) until convergence





# K-MEANS CLUSTERING ANIMATION



<https://www.youtube.com/watch?v=5l3Ei69l40s&t=59s>

## K-MEANS DETAILS

- For a very good detailed explanation of how K-Means works under the hood:

<https://www.youtube.com/watch?v=IX-3nGHDhQg>

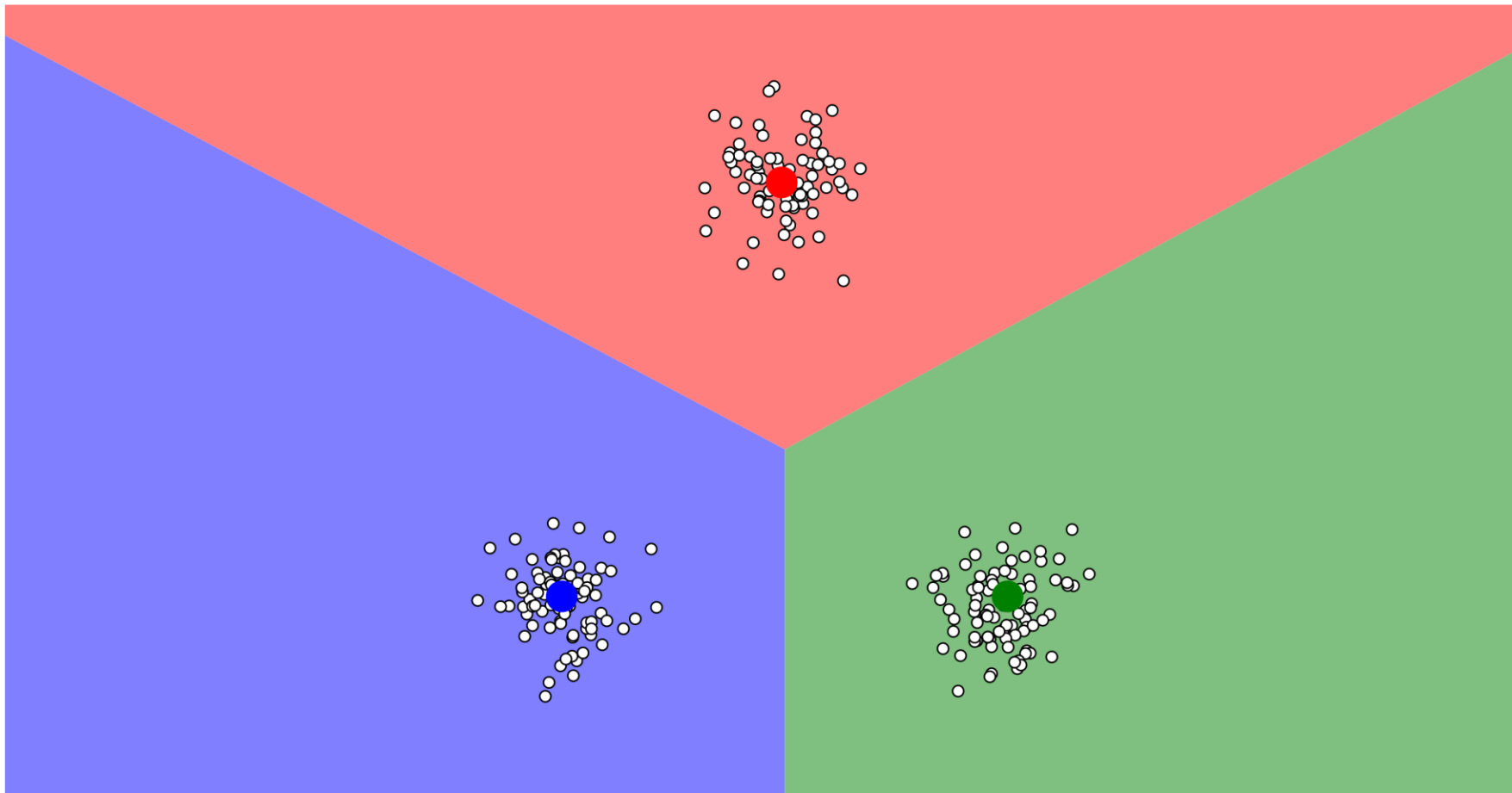
- Another very good video, if you can understand his accent, is:

<https://www.youtube.com/watch?v=iNIZ3IU5Ffw&t=300s>

- What are some of the advantages/disadvantages of the K-Means algorithm for clustering? When does it work well? When does it fail?

# Visualizing K-Means

<https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>





# EVALUATION METHOD: WITHIN CLUSTER SUM OF SQUARES (WCSS)

- The within-cluster sum of squares (WCSS) measures the compactness of a clustering result by calculating the sum of the squared distances between each data point and the centroid of the cluster it belongs to.
- Here's how to calculate the WCSS:
  1. **Identify the clusters:** For each cluster, determine the data points assigned to it and calculate the centroid.
  2. **Calculate the squared distance:** For each data point in a cluster, calculate the Euclidean distance to the cluster's centroid. Then, square this distance.
  3. **Sum the squared distances:** Sum up the squared distances for all points in the cluster to get the WCSS for that cluster.
  4. **Sum across all clusters:** If you have multiple clusters, repeat the above steps for each cluster and sum the results to get the overall WCSS.

```
import numpy as np

def calculate_wcss(data, labels, centroids):
    wcss = 0
    for i, centroid in enumerate(centroids):
        # Get points in cluster i
        cluster_points = data[labels == i]
        # Calculate the sum of squared distances to the centroid
        wcss += np.sum((cluster_points - centroid) ** 2)
    return wcss
```

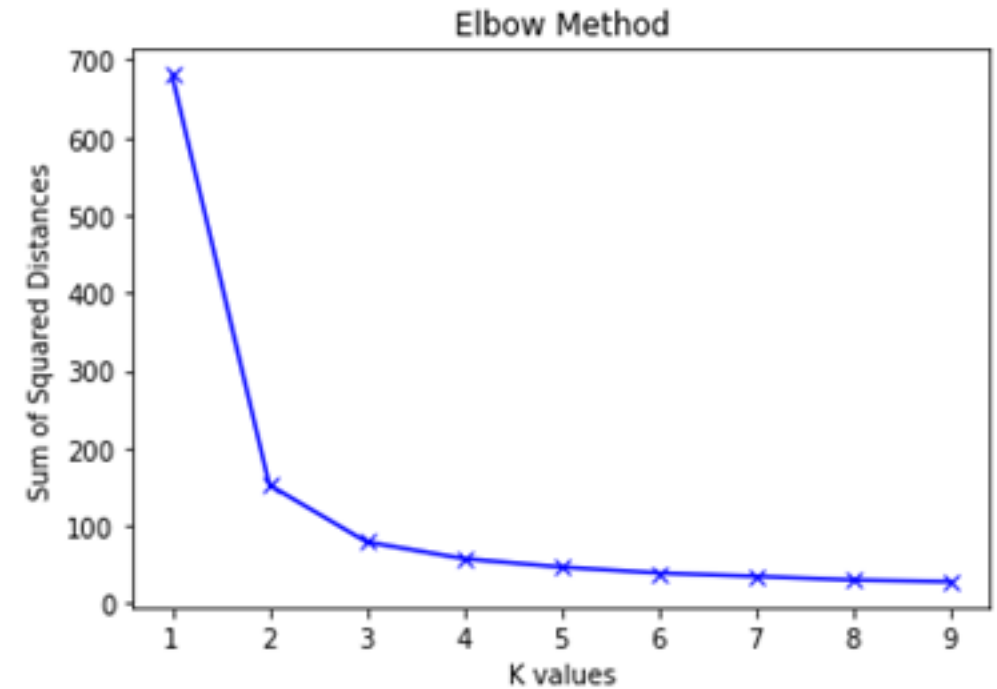
In this example:

- `data` is your dataset as a NumPy array,
- `labels` are the cluster labels for each data point (indicating cluster assignment),
- `centroids` are the coordinates of each cluster centroid.

# FINDING THE OPTIMAL K VALUE USING ELBOW METHOD

## Steps:

- 1. For different values of K, execute the following steps:
- 2. For each cluster, calculate the sum of the squared distance of every point to its centroid.
- 3. Add the sum of squared distances of each cluster to get the total sum of squared distances for that value of K.
- 4. Keep adding the total sum of squared distances for each K to a list.
- 5. Plot the sum of squared distances (using the list created in the previous step) and their K values.
- 6. Select the K at which a sharp change occurs (looks like an elbow of the curve).



# ELBOW PLOT CODE

**1. Data Generation:** Generate sample data with `make_blobs` for demonstration. In practice, you'd replace `X` with your dataset.

**2. WCSS Calculation:** Initialize an empty list `wcss` and use a loop to run `KMeans` with different values of `k` (from 1 to 10). The `inertia_` attribute of the `KMeans` object gives the WCSS for the fitted clusters.

**3. Plotting:** The plot of WCSS vs. `k` shows how the WCSS decreases as we increase the number of clusters. The “elbow” in this plot indicates the optimal number of clusters, where increasing `k` further has diminishing returns on reducing WCSS.

## 4. Interpreting the Elbow Plot

The elbow point, where the WCSS curve starts to flatten, suggests the optimal number of clusters. Choosing a `k` beyond this point provides minimal additional clustering improvement.

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

# Generate sample data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=1.0, random_state=42)

# List to store WCSS values for each k
wcss = []

# Run k-means with k values from 1 to 10 and calculate WCSS
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_) # inertia_ is the WCSS for the current k

# Plot the WCSS values for each k to find the "elbow" point
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.show()
```

# SILHOUETTE METHOD FOR EVALUATION

- The silhouette method is a technique for evaluating the quality of clustering, which can help determine the optimal number of clusters for a given dataset.
  - It quantifies how well each data point fits into its assigned cluster compared to other clusters.
  - **Silhouette Coefficient:** The silhouette coefficient is a measure of this fit, ranging from -1 to +1.
  - A higher score indicates better clustering.
  - The silhouette score combines the concepts of **cohesion** (how similar a point is to other points in its cluster) and **separation** (how dissimilar a point is to points in other clusters). [1]
- **Here's how the silhouette method works:**
  - Calculate the silhouette coefficient **for each data point**. The silhouette coefficient for a data point  $i$  is calculated as:
    - $(b(i) - a(i)) / \max(a(i), b(i))$
  - where:
    - $a(i)$  is the average distance between point  $i$  and all other points in the same cluster.
    - $b(i)$  is the average distance between point  $i$  and all points in the nearest neighboring cluster.
  - **Calculate the average silhouette coefficient for all data points.** This average score represents the overall quality of the clustering.
  - **Repeat steps 1 and 2 for different numbers of clusters.** By comparing the average silhouette coefficients for different  $k$  values, you can identify the number of clusters that produce the highest score, indicating the most optimal clustering structure.



# SILHOUETTE METHOD EXAMPLE

Suppose you have a dataset and want to determine the optimal number of clusters using the silhouette method. You apply a clustering algorithm, such as K-Means, with different  $k$  values (e.g.,  $k = 2, 3, 4, 5$ ). For each  $k$ , you calculate the silhouette coefficient for each data point and then average those coefficients. Let's say you obtain the following average silhouette scores:

- $k = 2$ : 0.45
- $k = 3$ : 0.62
- $k = 4$ : 0.55
- $k = 5$ : 0.48

Observing the silhouette scores, the highest score occurs at  $k = 3$ . This suggests that 3 is the most suitable number of clusters for this dataset.

## ■ Advantages:

- The silhouette method quantifies clustering quality, offering greater objectivity than visual methods like the elbow method.
- It considers both cohesion and separation, giving a more comprehensive evaluation of cluster structure.

## ■ Limitations:

- The calculation of silhouette coefficients **can be computationally expensive**, particularly for large datasets.
- As with any clustering evaluation metric, the silhouette method is not a foolproof solution and should be used in conjunction with other methods and domain knowledge.

# SILHOUETTE METHOD PYTHON CODE

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

# Generate synthetic dataset with 4 centers
X, _ = make_blobs(n_samples=500, centers=4, cluster_std=1.0, random_state=42)

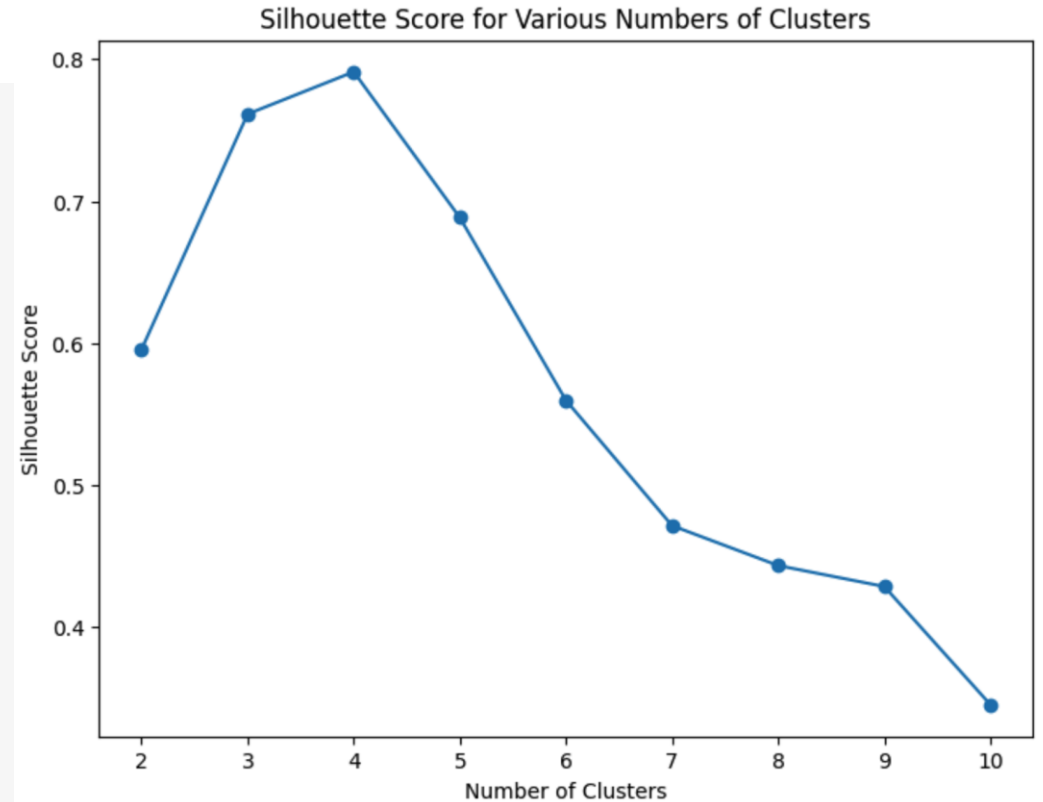
# List to store silhouette scores
silhouette_scores = []

# Range of clusters to test
range_n_clusters = range(2, 11)

# Compute silhouette scores for each number of clusters
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = kmeans.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the average silhouette score is: {silhouette_avg:.3f}")

# Plot silhouette scores
plt.figure(figsize=(8, 6))
plt.plot(range_n_clusters, silhouette_scores, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score for Various Numbers of Clusters")
plt.show()
```

Screenshot



## WSSC VS. SILHOUETTE

### When to Use Each?

| Metric           | Measures                  | Best For  | Limitations                                |
|------------------|---------------------------|---|--|
| WCSS             | Intra-cluster compactness | Elbow Method, K-Means                                 | Does not consider inter-cluster separation |
| Silhouette Score | Compactness + Separation  | Comparing cluster quality across different algorithms | Computationally expensive                  |