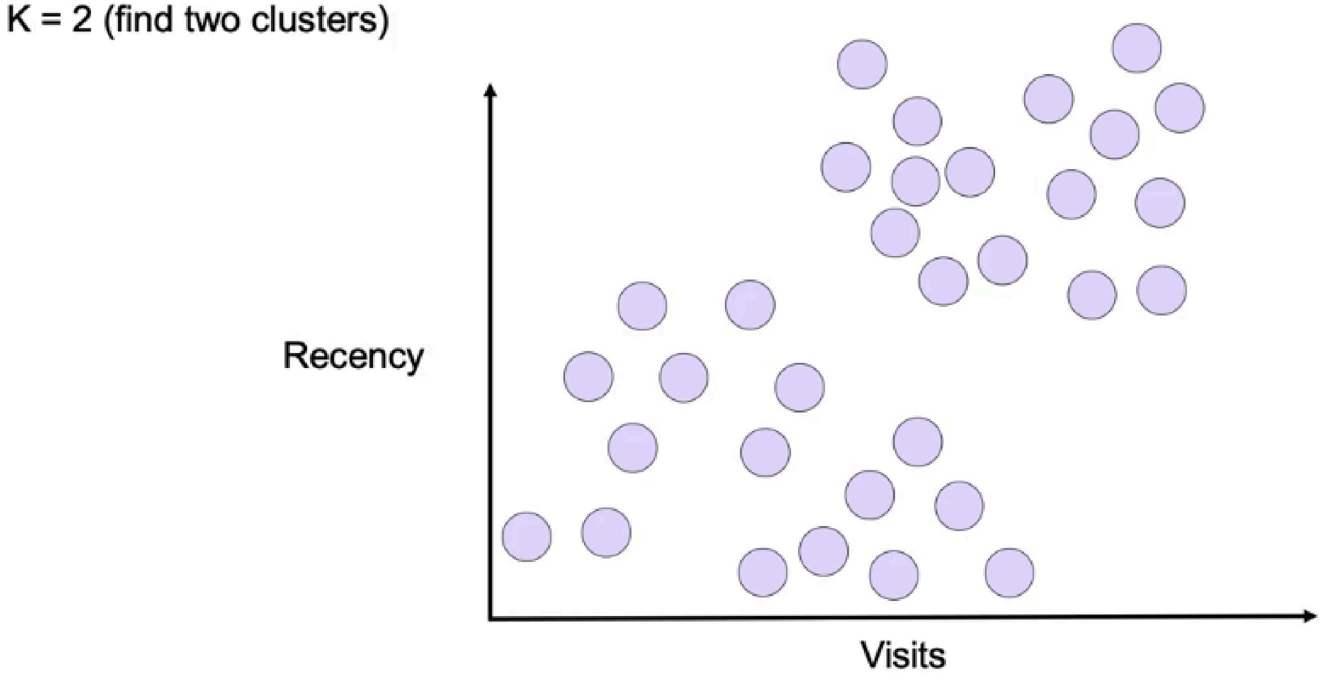
9) K-Means Clustering

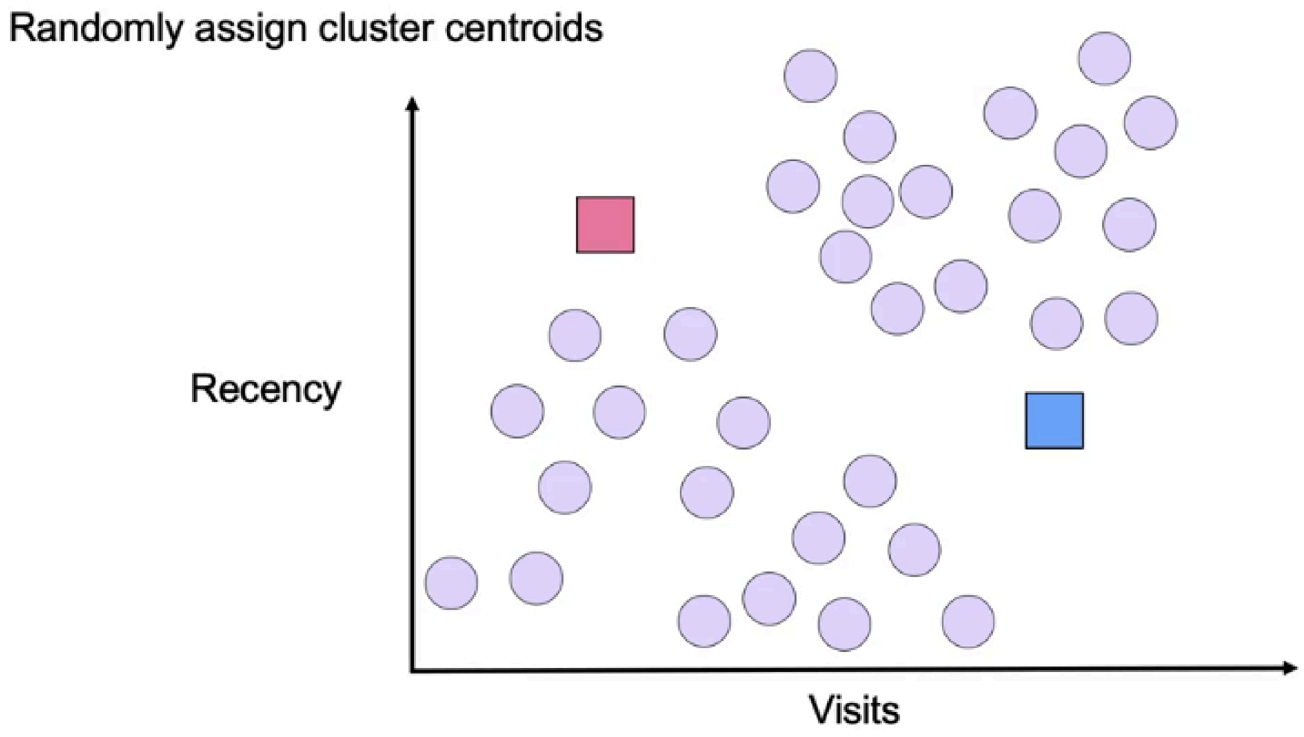
- Depending on the starting positions of the centroids, their end positions can be different

How it works

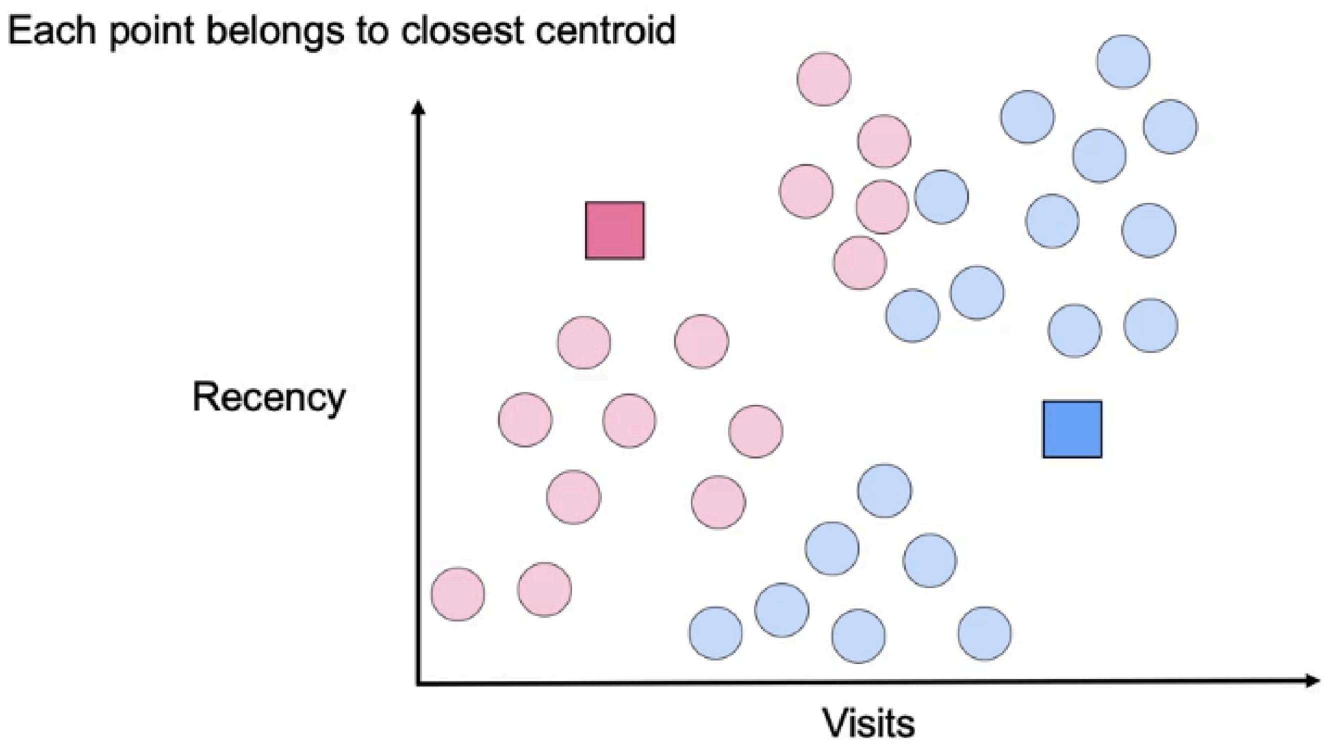
Suppose we have the following data:



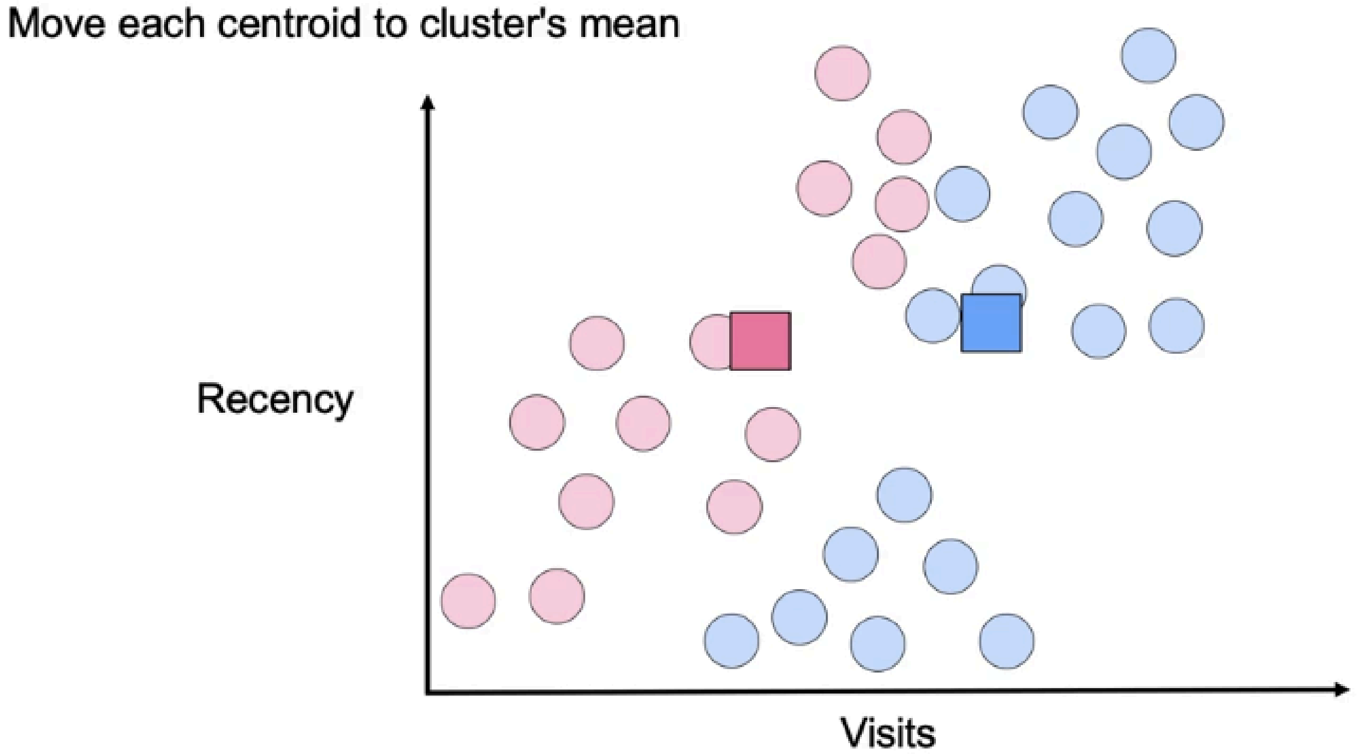
What we will do is we will first pick two random points (ie "centroids"), for example:



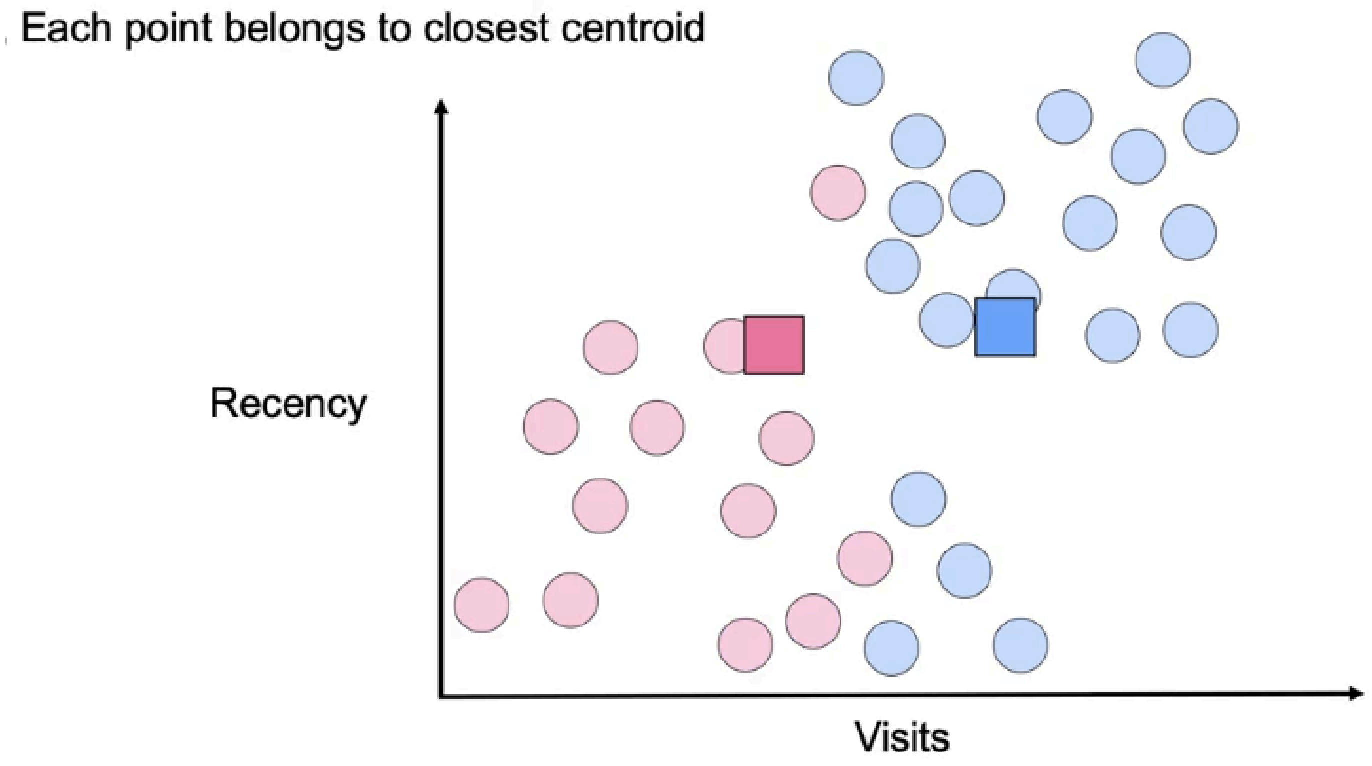
Then we try to predict what cluster each point belongs to (based on distance to the centroids), like so:



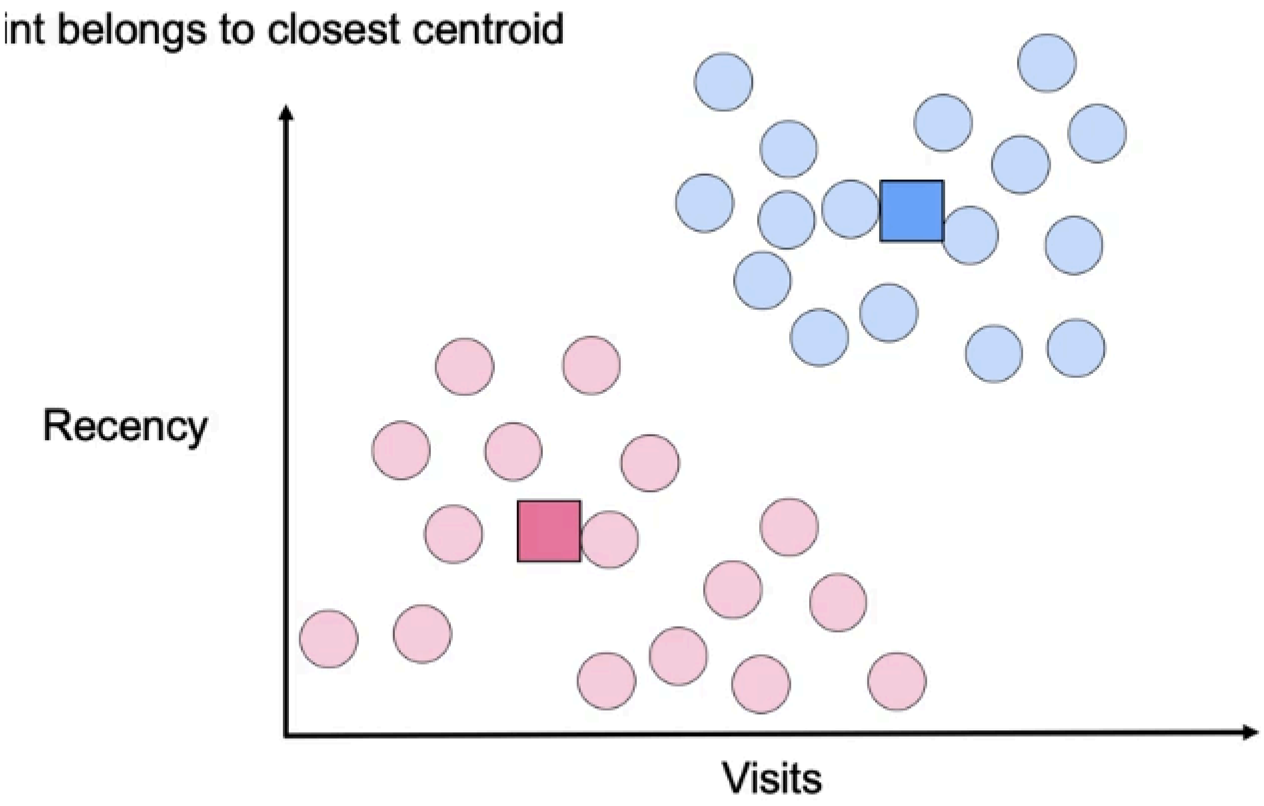
Then we move each centroid to its cluster's mean/center, like so:



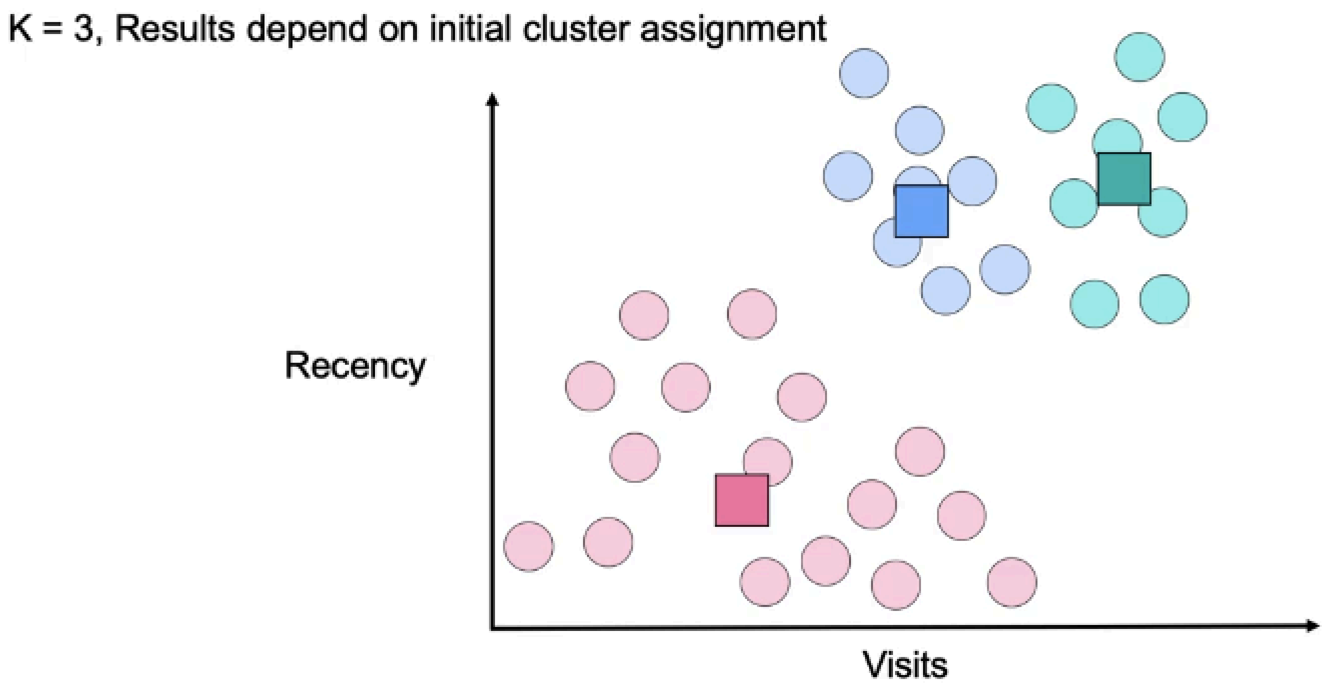
So lets do the prediction step again with the new centroids



We then move the centroids to the center/mean of their cluster and do the prediction again, and we keep doing that until the centroids dont move anymore, ie they are in the center/mean of their cluster. So we will end up with something like:



If there were 3 centroids, the result might have been:



Performance metric(s)

- These helps us select the right number of clusters
- To find the best model, initiate the model multiple times (with diff configs) and take the model with the best score

Inertia

- Sum of squared distances from each point (x_i) to its cluster's centroid (C_k)
- Smaller value corresponds to tighter cluster
- Value sensitive to the number of points in the cluster
  - **Inertia:** sum of squared distance from each point (x_i) to its cluster (C_k).

$$\sum_{i=1}^{n}(x_i - C_k)^2$$

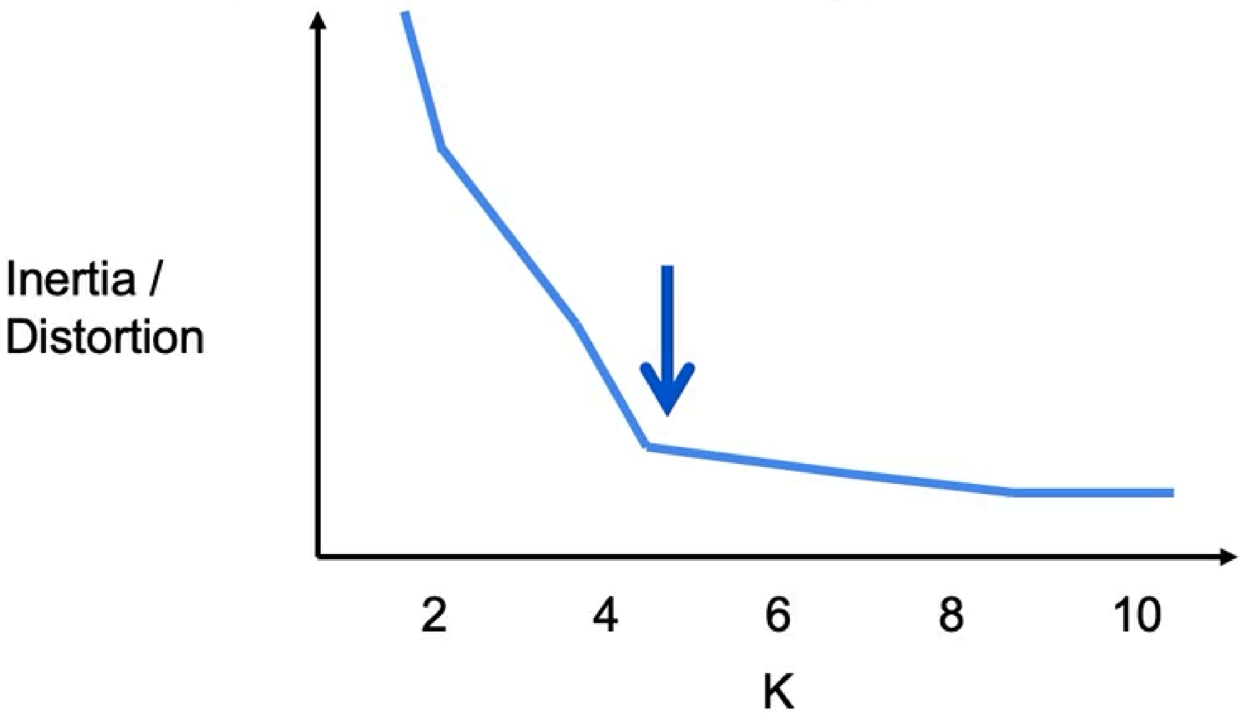
Distortion

- Same as inertia, but takes average instead of sum
- Average of the squared distances from each point (x_i) to its cluster's centroid (C_k)
- Smaller value corresponds to tighter cluster
- Doesn't generally increase when more points are added (relative to inertia)
  - **Distortion:** average of squared distance from each point (x_i) to its cluster (C_k).

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - C_k)^2$$

Elbow method

- Value decreases with increasing K. Just think like if we will have a cluster for every single data point, the inertia/distortion would be 0



So in the above graph, the best value for `k` will be 4, since there will be diminishing returns after that

Code

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3, init='k-means++')
model.fit(X1)
y_predict = model.predict(X2)

# Can also be used in batch mode with `MiniBatchKMeans`
from sklearn.cluster import MiniBatchKMeans
```

The elbow method can be done like:

```
perfs = {}
Ks = list(range(2,11))

for k in Ks:
    model = KMeans(n_clusters=k)
    model.fit(X)
    perfs[k] = model.inertia_
```