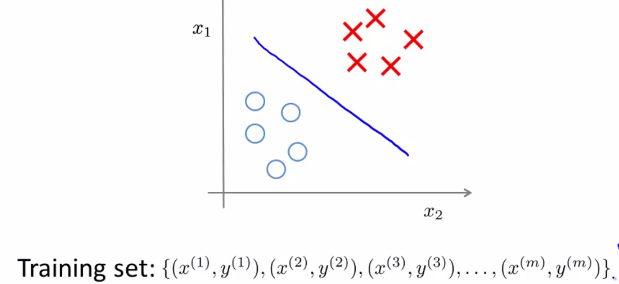
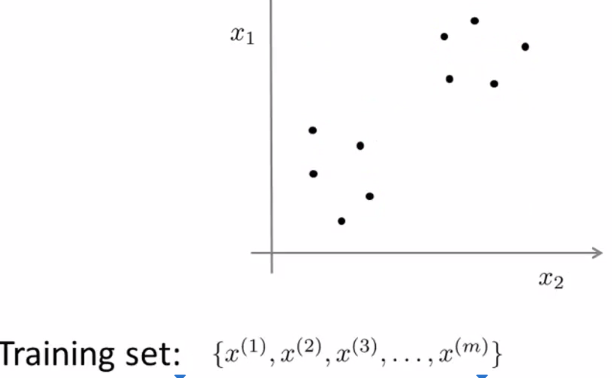
***Clustering***

**I. UNSUPERVISED LEARNING INTRO**

* Clustering will be our first unsupervised learning algorithm
* A typical supervised learning problem = *given* a LABELED training set w/ the goal of finding the decision boundary that separates  positive label + negative label examples.
* Given a set of labels to fit a hypothesis to it.



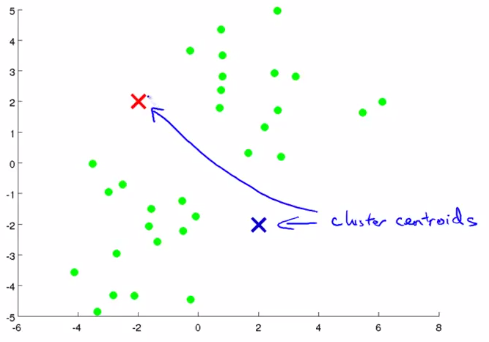
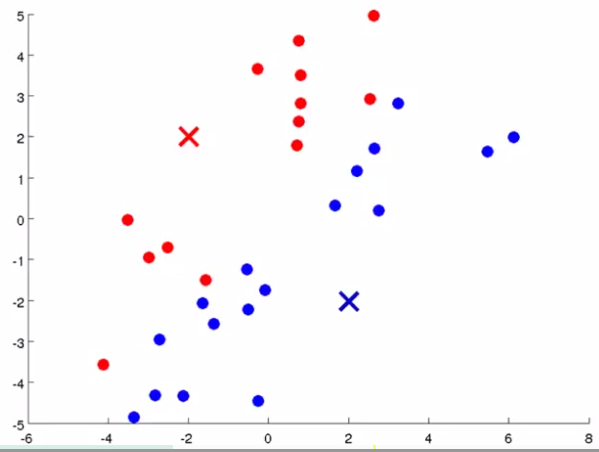
* In contrast, in an unsupervised learning problem, we're given data that does NOT have any labels associated w/ it. So, we're given data that looks like this.



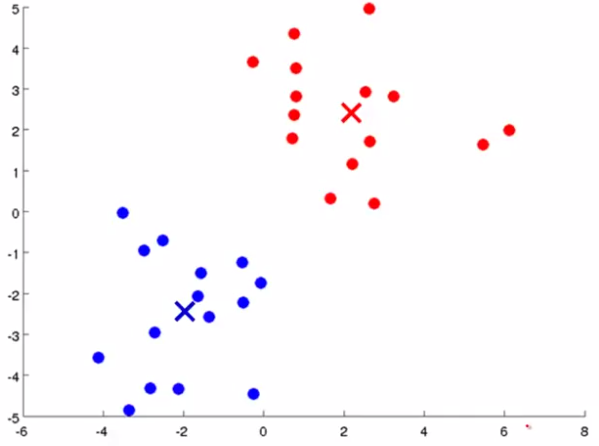
* See a set of points x w/ no labels y (no colors or shapes)
* In unsupervised learning, we give this unlabeled training set to an algorithm + ask it to find some structure in the data for us.
* 1 type of structure we might have an algorithm find for this data set a grouping into 2 separate clusters + so an algorithm that finds clusters like the ones I've just circled is called a clustering algorithm
* **Clustering** is good for:
* **Market segmentation** 🡪 may have a database of customers + want to group them into different segments so you can sell to them separately or serve different segments better.
* **Social network analysis** 🡪 things like Facebook, Google+, or info about people you email the most frequently to find coherence in groups of people
* Who are the coherent groups of friends in the social network?
* **To organize computing clusters/data centers better** 🡪 If you know which CPUs in a data center tend to work together, you can use that to reorganize resources + how we lay out the network + design the data center communications more efficiently
* **To understand galaxy formation** + using that to understand astronomical data.

**II. K-MEANS ALGORITHM**

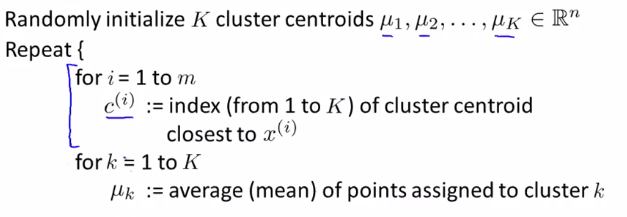
* **The K Means algorithm** is by far the most widely used clustering algorithm + is best illustrated in pictures
* Let's say I want to take an unlabeled data set + group the data into 2 clusters.
* 1st first step is to randomly initialize 2 points, the cluster **centroids,** b/c we want to group my data into 2 clusters.
* K Means is an **iterative algorithm** + it has 2 steps🡪 **Cluster assignment** + move centroid.
* In cluster assignment, the algorithm goes through each of the examples + depending on whether it's closer to the 1 cluster centroid or the other, it assigns the data point to 1 of the 2 centroids

* In move centroid step, we take the 2 centroids + move them to *the average of the points* w/in the cluster (move centroids to mean of the location of all red points + mean location of all blue points)
* Then we go back + do *another cluster assignment step* + then do *another* move centroid step.
* If you keep running additional iterations of K-means, the cluster centroids will eventually not change any further
* At this point, *K means has* ***converged*** + it's done a pretty good job finding the 2 clusters in this data



* Let's write out the K means algorithm more formally.

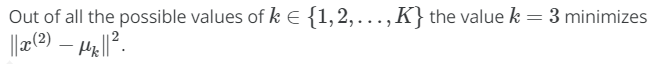


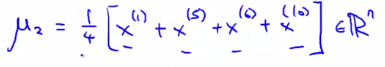
* It takes 2 inputs: parameter **K**, the number of clusters you want to find in the data, + an unlabeled training set of just X’s 🡪 {x1, …. x(m)}
* X(i) is an R(n) dimensional vector = training examples are n-dimensional rather n+1 dimensional



* The 1st step in K means is to randomly initialize k cluster centroids which we call μ1, μ2, up to μ(k)
* Then, the “inner loop” 🡪 For each training examples, set a variable c(i) to be the index (1-K) of the cluster centroid closest to x(i) 🡺 cluster assignment
* c(i) is going to be a number from 1 to K that tells us which cluster a point is closes to
* Another way of writing this: Compute c(i) by taking the ith example x(i) + measure it's distance to each of cluster centroid = *norm between x and μ* = **||x(i) – μ(k)||**



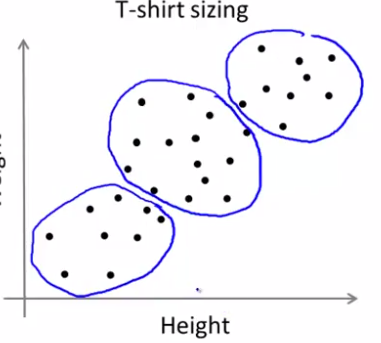
* capital K = the total number centroids lowercase k = index of different centroids.
* C(i) is going to find the value of k that minimizes this distance between X(i) + the cluster centroid
* By convention people tend to write this as the squared distance, so we can think of c(i) as picking the cluster centroid w/ the smallest squared distance to a training example x(i).
* But, of course, minimizing squared distance + minimizing distance should give the same value
* 
* 
* 
* 
* The other “inner loop” of K means does the move centroid step.
* For each cluster centroid (k = {1, …., K}), it sets μ(k) to the average of the points assigned to that cluster k
* As a concrete example, let's say cluster centroid 2 has training examples (x1, x5, x6, x10) assigned to it
* This also means c(1) = 2, c(5) = 2, c(6) = 2, c(10) = 2,
* The average of these points in cluster 2, μ2, is an n-dimensional vector (b/c each example was an n-dimensional vector



* This has the effect of moving μ2 to the average of those
* But what if there’s a centroid w/ 0 points assigned to it?
* In that case, the more common thing to do is to just eliminate that centroid + end up w/ K-1 clusters
* Sometimes if you *really need* k clusters, the other thing you can do is randomly reinitialize that centroid
* But it's more common to just eliminate a cluster if, somewhere during K-means, it has no points assigned to its centroid (that can happen, but not that often)
* 1 other common application of K Means is to the problems w/ **non-separated clusters**.
* So far we've been picturing K Means + applying it to data sets w/ pretty well-separated clusters
* It turns out that very often K Means is also applied to data sets where there may not be several very well-separated clusters.
* Example application: Let's say you are a t-shirt manufacturer + you've gone to the population you want to sell shirts to + collected several examples of height + weight of these people



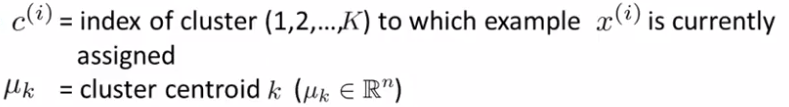
* Now you want to size your t shirts + design shirts of 3sizes, small, medium + large.
* So how big should I make small? How big should I make medium? How big should I make large?
* 1 way to do that would to be to run my K-means clustering on this data set
* Maybe what K Means will do is group these points into 3 clusters



* So, even though beforehand it didn't seem like we had 3 well-separated clusters, K-Means will separate out the data into multiple clusters for you
* What you can do is then look at this 1st population of people + their height + weight + try to design a small t-shirt so that it fits this population of people well
* Then design a medium t-shirt + design a large t-shirt
* This is kind of an example of market segmentation, where you're using K-Means to separate your market into 3 different segments to design a product separately that tries to suit the needs of each of your 3 separate sub-populations well.

**III. OPTIMIZATION OBJECTIVE**

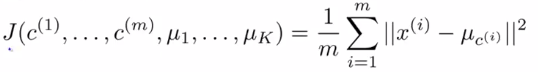
* Most of the supervised learning algorithms we've seen have an **optimization objective** or some **cost function** the algorithm was trying to *minimize*.
* It turns out K-means also has an optimization objective/cost function it's trying to minimize.
* Knowing what this is will be useful for 2 purposes:
* Knowing what the optimization objective of K-means is help one to debug the learning algorithm to make sure K-means is running correctly.
* We can use it to help K-means find better clusters + avoid the local optima.
* Reminder: While K-means is running, we keep track of 2 sets of variables:

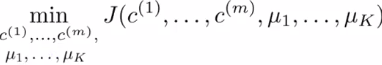


* First is the c(i)'s and that keeps track of the index or the number of the cluster, to which an example x(i) is currently assigned. And then the other set of variables we use is μ subscript k, which is the location of cluster centroid k. Again, for k-means we use capital K to denote the total number of clusters. And here lower case k is going to be an index into the cluster centroids and so, lower case k is going to be a number between one and capital K.
* 1:29
* We use **μ(c(i))** to denote the centroid of the cluster to which example x(i) has been assigned
* Let’s say that x(i) has been assigned to cluster 5, therefore c(i), the *index* of x(i), is equal to 5.
* So μ(c(i)) is = μ5 b/c c(i) = 5

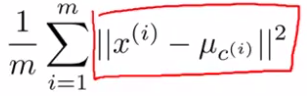


* The optimization objective of the k-means clustering algorithm:

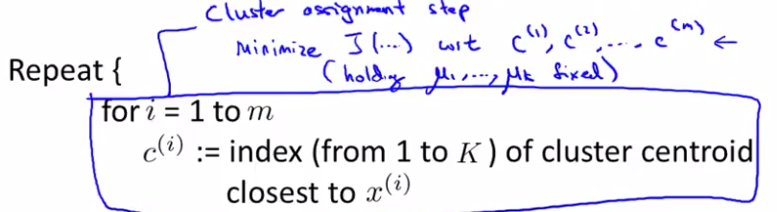


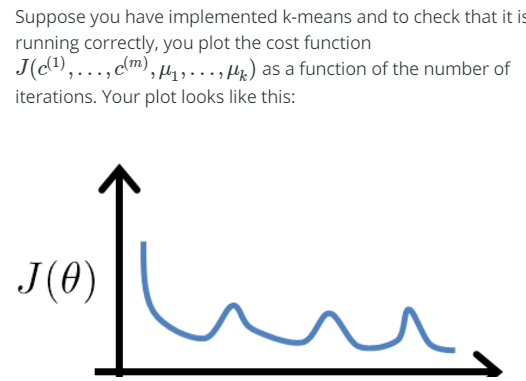


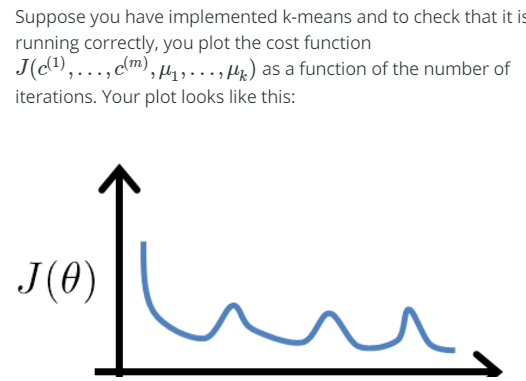
* The cost function K-means is minimizing is a function, J, of all the parameters [c1-c(m) and μ1 through μ(K)] that K-means is varying as the algorithm runs.
* The actual optimization objective is to the right 🡪 1/m \* sum of the squared distances between each example x(i) and the location of the cluster centroid to which x(i) has been assigned.



* What K-means is doing is trying to define parameters c(i) and μ(i) to try to minimize this cost function J, sometimes called the **distortion** **cost function**, or **distortion of the K-means algorithm**
* It's possible to show mathematically that the cluster assignment step is doing is minimizing J w/ respect to the variables c1, c2 and so on, up to c(m) *while holding the centroids μ1 up to μ(K) fixed*.
* So, the cluster assignment step doesn't change the centroids, but is picking the values of c1-c(m) that minimizes the cost/distortion function J.
* Intuitive meaning: “Assign each point to a centroid closest to it, b/c that's what minimizes the squared distance between the points + the centroid”

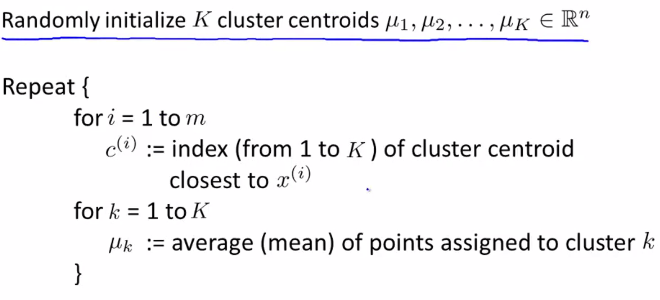
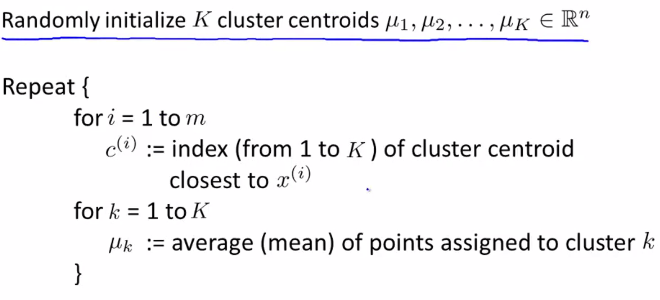


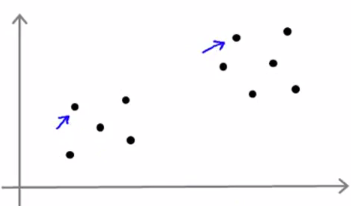
* It can also be shown mathematically that the move centroid step chooses the values of μ that minimizes J (w/respect to the locations of the centroids μ1-μ(K).
* So, K-means is this taking the 2 sets of variables + partitioning them into 2 halves, the set of c variables + the set of μ variables.
* It first minimizes J w/ respect to c + *then* minimizes J w/ respect to μ
* *Then it keeps on.*
* We can also use this optimization objective/cost function to try to debug other any algorithm + make sure our implementation of K-means is converging + running correctly
* 



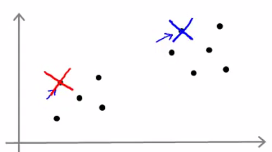
* 

**IV. RANDOM INITIALIZATION**

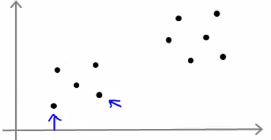
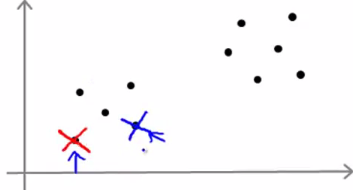
* Here's the K-means clustering algorithm that we talked about earlier.
* 
* 1 step we never really talked much about was HOW you *randomly initialize* the cluster centroids.
* There are few different ways one can imagine using to randomly initialize cluster centroids.
* It turns out that there’s 1 method that’s much more recommended than most other options
* When running K-means, you should have the number of centroids, K, set to be less than the number of training examples m.
* It would be really weird to run K-means with a number of centroids equal or greater than the number of examples you have
* Randomly pick K training examples + set μ1-μK equal to these K examples.
* Let’s say that K = 2 + we want to find 2 clusters
* In order to initialize centroids, randomly pick a couple examples



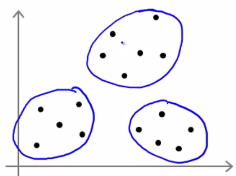
* Initialize the centroids right on top of those examples + that's 1 random initialization of K-means



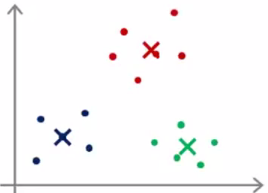
* Sometimes I might get less lucky when picking random initial examples

 🡺 

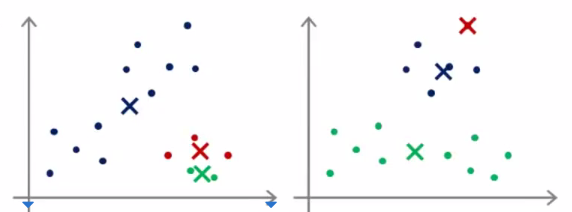
* At initialization, the 1st cluster centroid μ1 will = x(i) for some randomly chosen value of i + μ2 will = x(j) for some different randomly chosen value of j + so on for more clusters + more centroids
* K-means can end up converging to different solutions depending on how centroids were initialized,
* In particular, K-means can actually end up at local optima.
* If you're given the data w/ 3 possible clusters:



* If you run K-means + it ends up at a good local optima (maybe the global optima), you might end up w/ good clustering.



* But if you had a particularly unlucky, random initialization, K-means can also get stuck at different local optima.

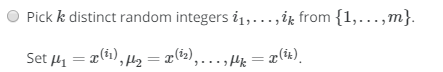


* On the left, the blue cluster has captured a lot of points on the left, while green captured only a relatively small number of points
* This corresponds to a bad local optima b/c it has basically taken the red + blue clusters + fused them into 1 blue one + has split the green cluster into 2 separate sub-clusters, red and green.
* Both of these examples correspond to different local optima of K-means
* On the right, the red cluster captured only a single unlabeled example.
* Remember **local optima** refers to local optima of this **distortion function** J



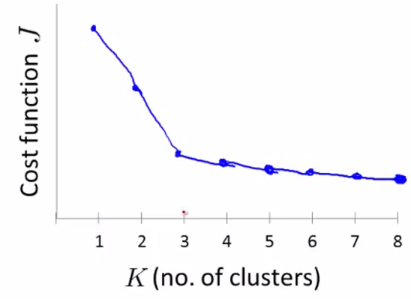
* What these local optima really correspond to are solutions where K-means got stuck to the local optima + is not doing a very good job minimizing this distortion function J.
* So, if you're worried about K-means getting stuck in local optima + you want to increase the odds of K-means finding the best possible clustering, we can try *multiple* random initializations.
* Run K-means lots of times to try to make sure we get as good a solution/as good a local/global optima as possible.
* Let's say, I decide to run K-means 100 times (50-1k is typical)
* For each of the 100 random initializations, we’d run K-means + that would give us a set of cluster centroids, + then we’d then compute the distortion/cost J on the set of cluster assignments + cluster centroids that we got.
* Having done this whole procedure 100 times, you’ll have 100 different ways of clustering the data
* Finally, just pick 1 clustering that gives us the lowest cost/distortion



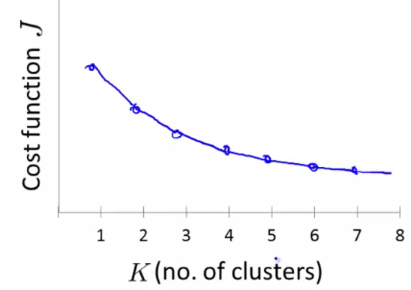
* It turns out that if running K-means w/ a fairly small number of clusters (2-10), doing multiple random initialization can sometimes ensure you find a better local optima/better clustering of the data
* But if K is very large (much greater than 10), multiple random initializations is LESS likely to make a HUGE difference
* There is a much higher chance your 1st random initialization will give you a pretty decent solution
* Doing multiple random initializations afterward will probably give you a *slightly better* solution but, but maybe not that much.
* But it’s really only when you have a relatively small number of clusters (2-4) that random initialization could make a huge difference in terms minimizing distortion + giving good clustering
* So, if you're trying to learn a clustering w/ a relatively small number of clusters (2-67), using multiple random initializations can sometimes help you find much better clustering of the data.
* But, even if learning a large number of clusters, the random initialization method should give K-means a reasonable starting point to start from for finding a good set of clusters.
* 
* 

**V. CHOOSING THE NUMBER OF CLUSTERS**

* To be honest, there actually isn't a great way choosing the number of clusters or doing it automatically
* By fa, the most common way of choosing the number of clusters is still doing it manually by looking at visualizations or at the output of the clustering algorithm, or something else.
* A large part of why it might not always be easy to choose the number of clusters it’s often generally ambiguous as to how many clusters there are in the data.
* For a data set, some may see 4 clusters, some see 2, some maybe 3
* This is part of unsupervised learning 🡪 We are aren't given labels, + so there isn't always a clear cut answer
* This is 1 of the things that makes it more difficult to have an automatic algorithm for choosing how many clusters to have.
* When people talk about ways of choosing the number of clusters, 1 method they sometimes talk about is something called the **Elbow Method**
* In which we vary K, the total number of clusters
* We run K-means w/ 1 cluster (everything gets grouped into a single cluster) + compute the cost/ distortion function J + plot it
* Then run K means w/ 2 clusters (maybe w/ multiple random initialization, maybe not) + this should hopefully result in a smaller distortion seen in its plot
* Then run K-means w/ 3 clusters, 4, 5 + so on
* We end up w/ a curve showing how the distortion should go down as we increase the number of clusters



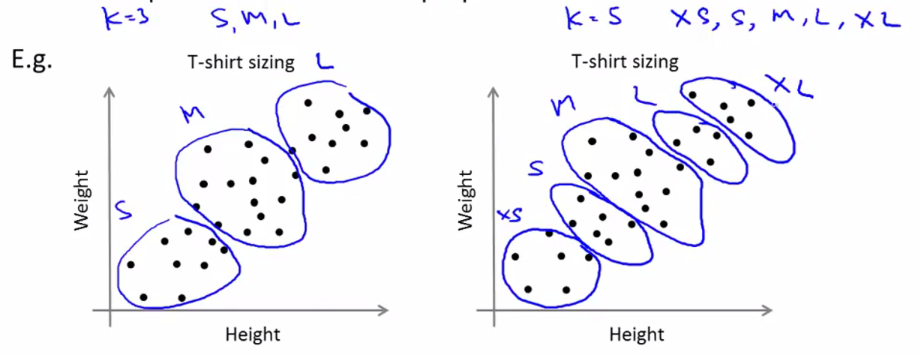
* Looking at this plot, it looks like there's a clear elbow there at K = 3
* We see a pattern where distortion goes down rapidly from 1 to 2, + 2 to 3, then we reach an elbow at 3, + then the distortion goes down very minimally/slowly after that
* It looks like 3 clusters is the right number of clusters, b/c that's the elbow of this curve
* Distortion goes down rapidly until K = 3, + goes down very slowly after that.
* If you apply the Elbow Method, + if you get a plot that actually looks like this, that's pretty good + would be a reasonable way of choosing the number of clusters.
* But the Elbow Method isn't used that often + 1 reason is that, if you actually use this on a clustering problem, you often end up w/ a curve that looks much more ambiguous:



* There's no clear elbow, but it looks like distortion continuously goes down, so maybe 3 is a good number, maybe 4, maybe 5
* It makes it harder to choose a number of clusters using this method.
* So maybe the Elbow Method is worth a shot, but don’t have high expectations of it working for any particular problem.
* 



* 1 other way of thinking about how you choose the value of K
* Very often people are running K-means in order to get clusters for some *later/downstream* purpose
* Maybe you want to use K-means in order to do market segmentation or to organize a CPU cluster better, or for some different purpose
* If that downstream purpose gives you an evaluation metric, then often, a better way to determine number of clusters is to see how well different numbers of clusters serve that downstream purpose
* Ex: T-shirt size
* Trying to decide, do I want K = 3 T-shirt sizes (S, M, L)?
* Or maybe I want to choose K = 5 (XS, S, M, L, XL)
* If I run K-means w/ K = 3 and w/ K = 5 clusters:



* The nice thing about this example is it gives us another way to choose whether we want 3 or 4 or 5 clusters,
* In particular, what you can do is ask: "If I have 5 segments, how well will my T-shirts fit my customers? How many T-shirts can I sell? How happy will my customers be?"
* What really makes sense?
* From the perspective of my T-shirt business, do I want more T-shirt sizes so my T-shirts fit my customers better, or do I want fewer T-shirt sizes so I can sell them to customers more cheaply?
* The business itself might give you a way to decide between 3 vs 5 clusters.
* If using K-means for image compression + trying to choose how many clusters to use, you could again use an evaluation metric of image compression to choose the number of clusters, K
* So, how good you want an image to look VS how much you want to compress the image file size
* For the most part, the number of clusters K is still chosen by hand/human input/human insight.
* 1 way to try to do so automatically to use the Elbow Method, but don’t always expect that to work well
* A better way to think about how to choose the number of clusters is to ask “For what purpose are you running K-means? What is the number of clusters K that serves whatever later purpose you are actually running the K-means for?”