# Clustering

Unsupervised learning is contrasted from supervised learning because it uses an **unlabeled** training set rather than a labeled one.

In other words, we don't have the vector y of expected results, we only have a dataset of features where we can find structure.

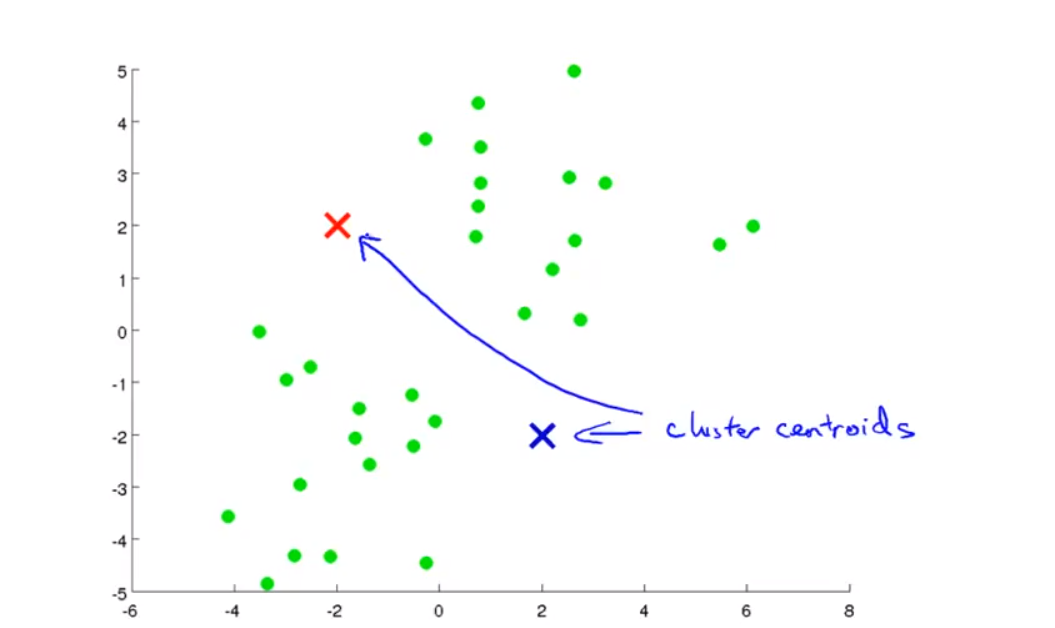
Clustering is good for:

* Market segmentation
* Social network analysis
* Organizing computer clusters
* Astronomical data analysis

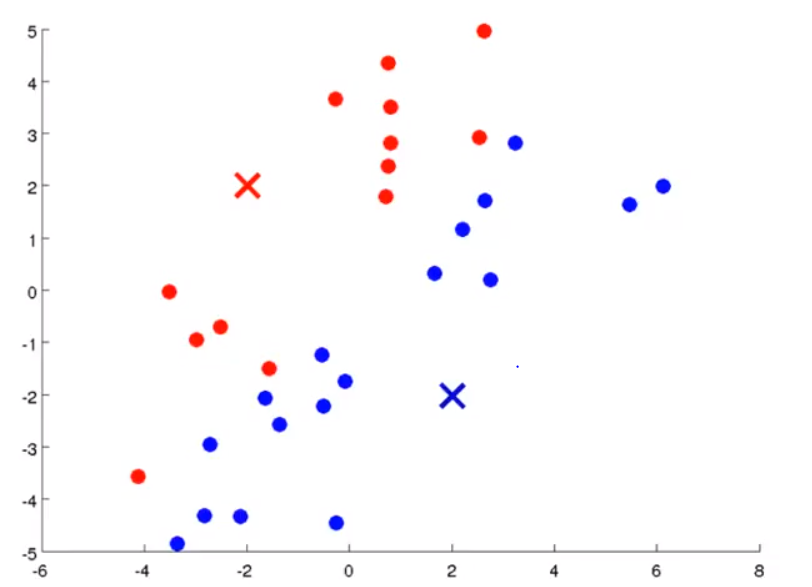
## K-means Algorithm

The K-Means Algorithm is the most popular and widely used algorithm for automatically grouping data into coherent subsets.

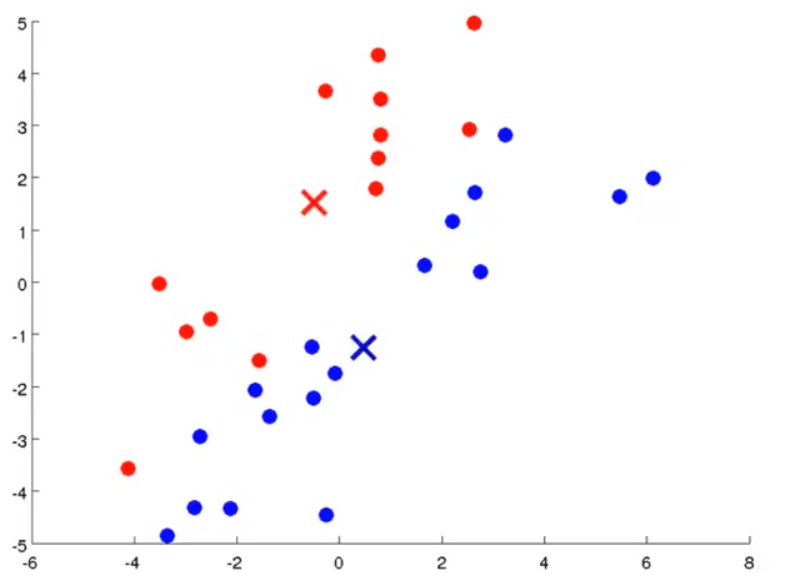
1. Randomly initialize two points in the dataset called the cluster centroids.



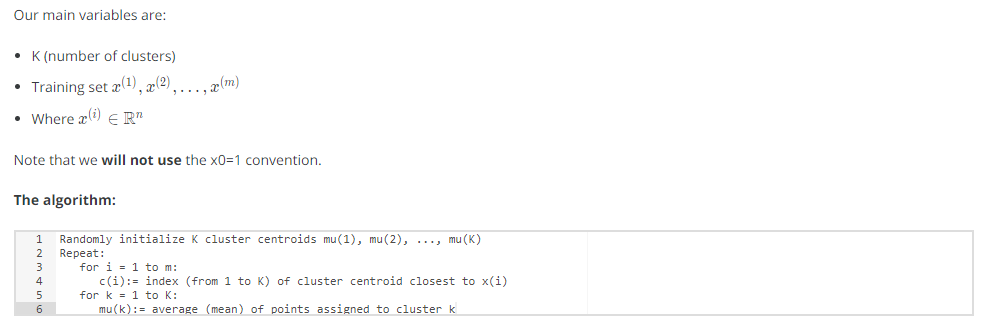
1. Cluster assignment: assign all examples into one of two groups based on which cluster centroid the example is closest to.



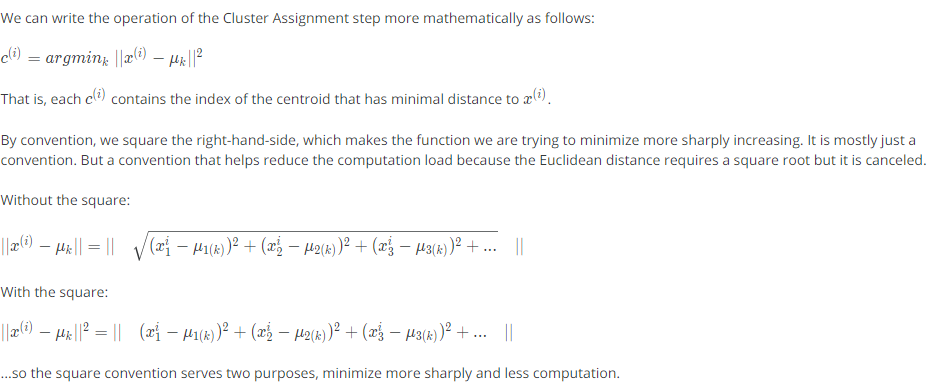
1. Move centroid: compute the averages for all the points inside each of the two cluster centroid groups, then move the cluster centroid points to those averages.

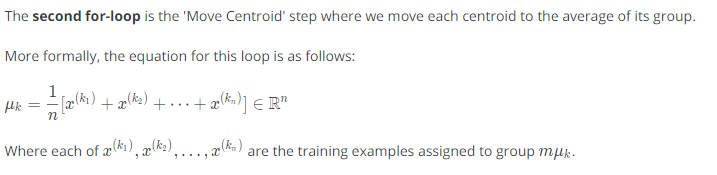


1. Re-run (2) and (3) until we have found our clusters.



The **first for-loop** is the 'Cluster Assignment' step. We make a vector c where c(i) represents the centroid assigned to example x(i).



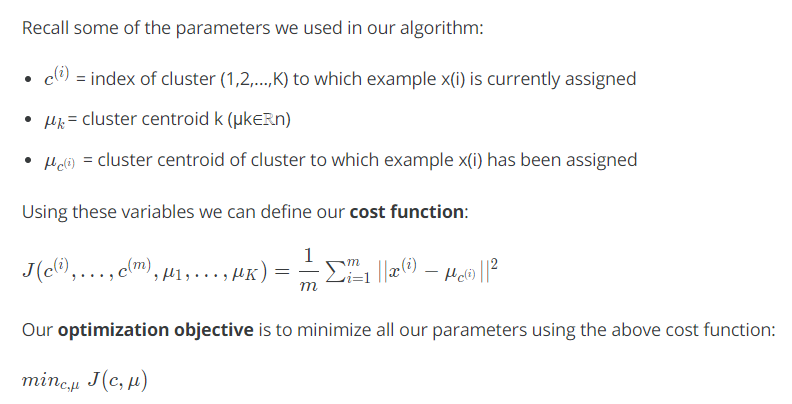


If you have a cluster centroid with **0 points** assigned to it, you can randomly **re-initialize** that centroid to a new point. You can also simply **eliminate** that cluster group.

After a number of iterations the algorithm will **converge**, where new iterations do not affect the clusters.

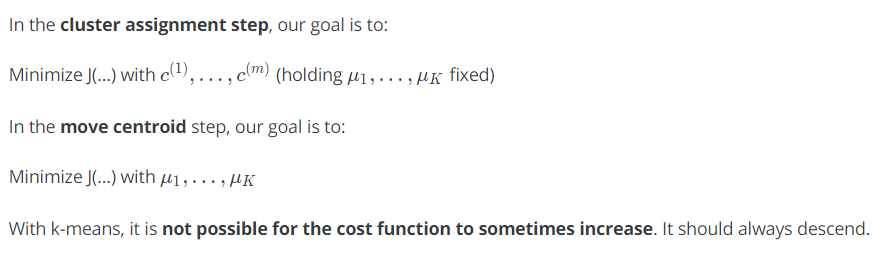
**Note on non-separated clusters**: Some datasets have no real inner separation or natural structure. K-means can still evenly segment your data into K subsets, so can still be useful in this case.

## Optimization Objective

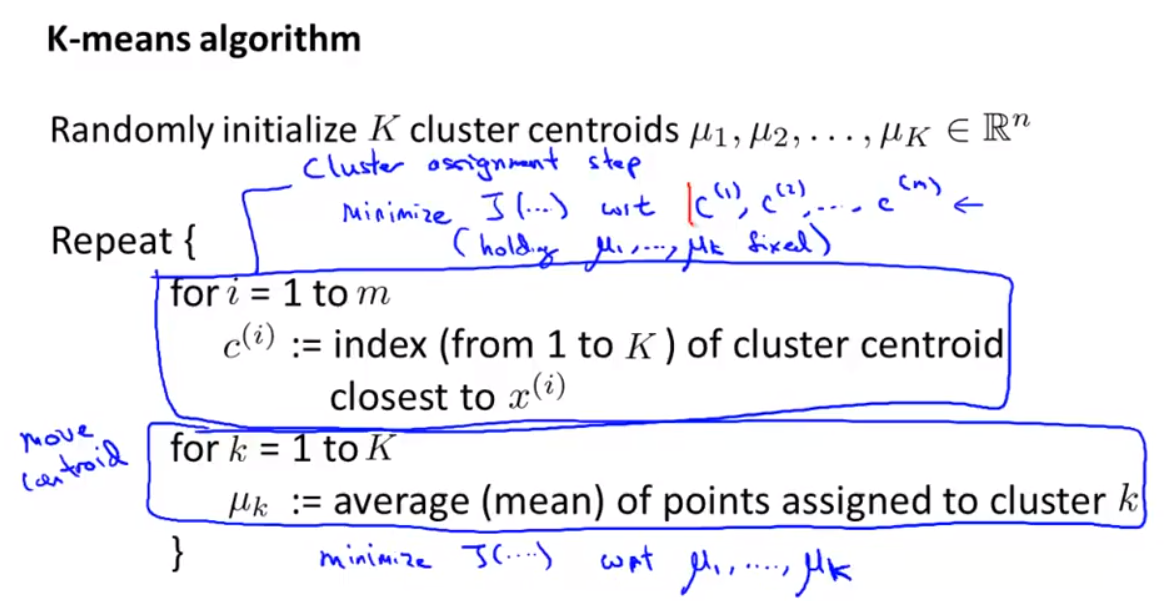


That is, we are finding all the values in sets c, representing all our clusters, and μ, representing all our centroids, that will minimize **the average of the distances** of every training example to its corresponding cluster centroid.

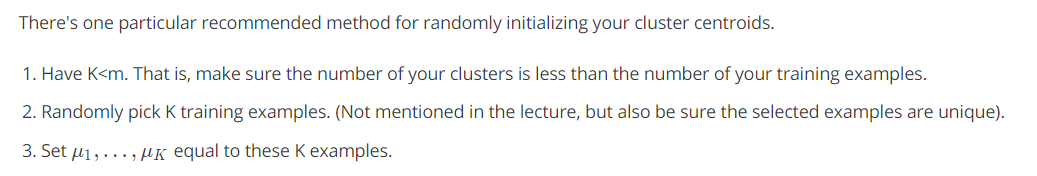
The above cost function is often called the **distortion** of the training examples.



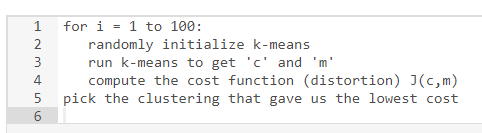
If it is increasing, it means there is some bug in the code.



## Random Initialization



K-means **can get stuck in local optima**. To decrease the chance of this happening, you can run the algorithm on many different random initializations. In cases where K<10 it is strongly recommended to run a loop of random initializations.



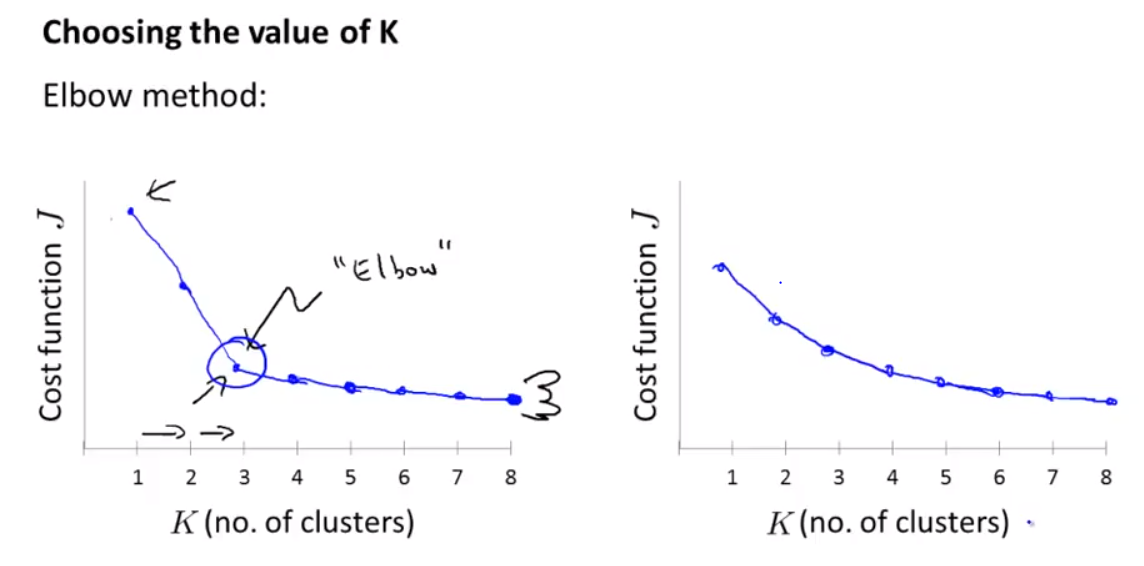
## Choosing the number of clusters

Choosing K can be quite arbitrary and ambiguous.

**The elbow method**: plot the cost J and the number of clusters K. The cost function should reduce as we increase the number of clusters, and then flatten out. Choose K at the point where the cost function starts to flatten out.

However, fairly often, the curve is **very gradual**, so there's no clear elbow.

**Note:** J will **always** decrease as K is increased. The one exception where it can increase is if k-means gets stuck at a bad local optimum.

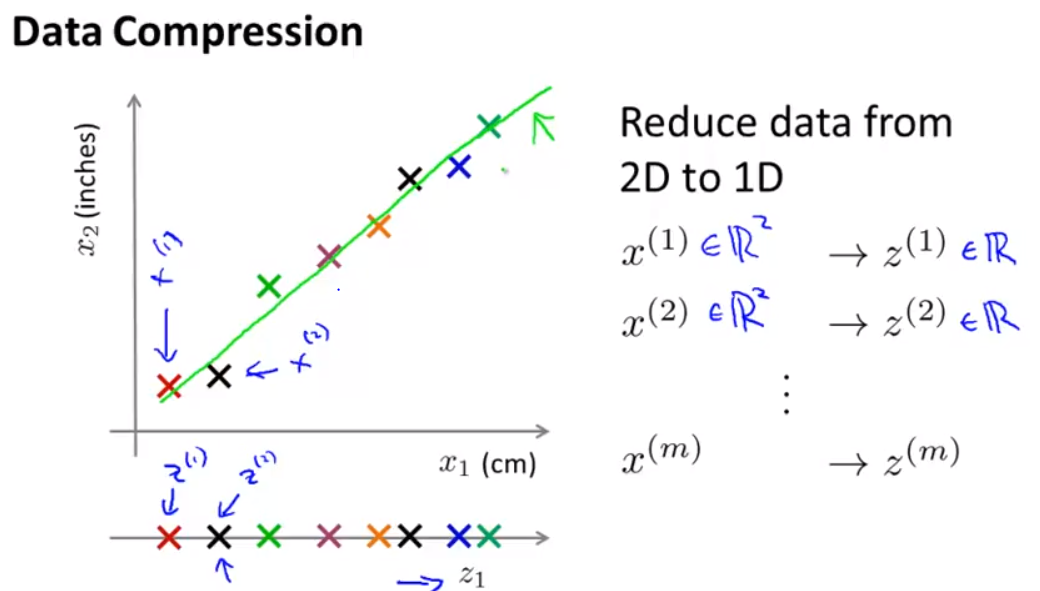


Another way to choose K is to observe how well k-means performs on a **downstream purpose**. In other words, you choose K that proves to be most useful for some goal you're trying to achieve from using these clusters.

# Dimensionality Reduction

## Motivation I – Data compression

* We may want to reduce the dimension of our features if we have a lot of redundant data.
* To do this, we find two highly correlated features, plot them, and make a new line that seems to describe both features accurately. We place all the new features on this single line.



Doing dimensionality reduction will reduce the total data we have to store in computer memory and will speed up our learning algorithm.

**Note: In dimensionality reduction, we are reducing our features rather than our number of examples.** Our variable m will stay the same size; n, the number of features each example from x(1) to x(m) carries, will be reduced.

## Motivation II – Data Visualization

It is not easy to visualize data that is more than three dimensions. We can reduce the dimensions of our data to 3 or less in order to plot it.

We need to find new features, z1, z2 ​(and perhaps z3) that can effectively **summarize** all the other features.

Example: hundreds of features related to a country's economic system may all be combined into one feature that you call "Economic Activity."



# Principal Component Analysis (PCA)

## PCA problem formulation

The most popular dimensionality reduction algorithm is *Principal Component Analysis* (PCA).

**Problem formulation**

Given two features, *x*1​ and *x*2​, we want to find a single line that effectively describes both features at once. We then map our old features onto this new line to get a new single feature.

The same can be done with three features, where we map them to a plane.

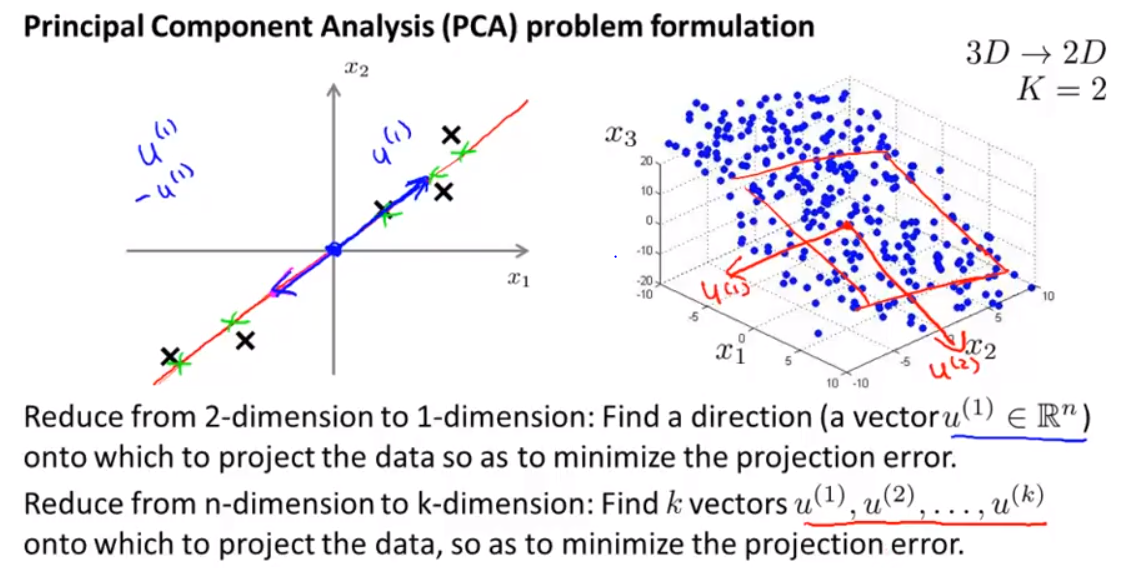
The **goal of PCA** is to **reduce** the average of all the distances of every feature to the projection line. This is the **projection error**.

**Reduce from 2d to 1d**: find a direction (a vector u(1) ∈R*n*) onto which to project the data so as to minimize the projection error.

The more general case is as follows:

**Reduce from n-dimension to k-dimension**: Find k vectors *u*(1), *u*(2),…, *u*(k) onto which to project the data so as to minimize the projection error.

If we are converting from 3d to 2d, we will project our data onto two directions (a plane), so k will be 2.

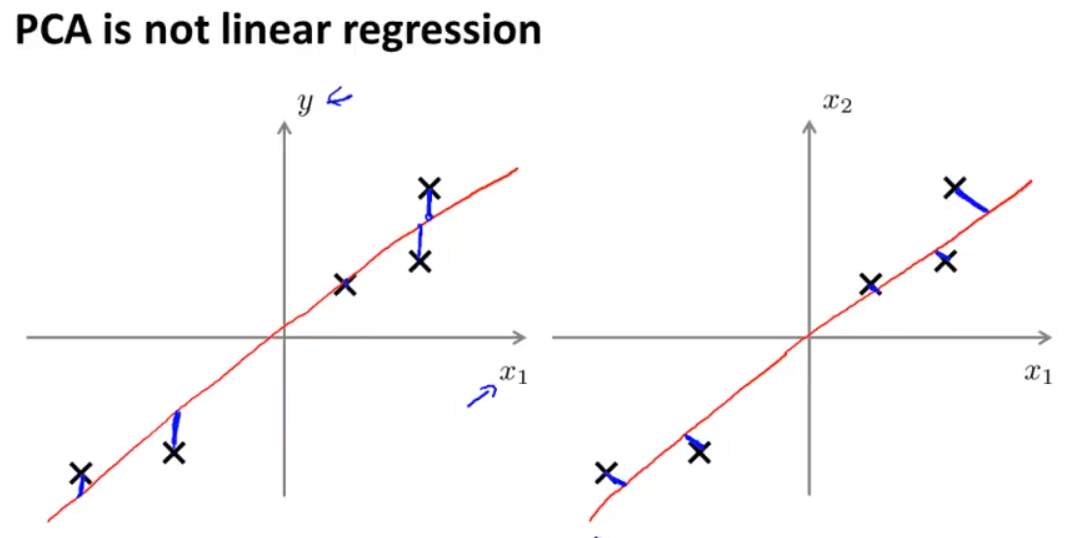


**PCA is not linear regression**

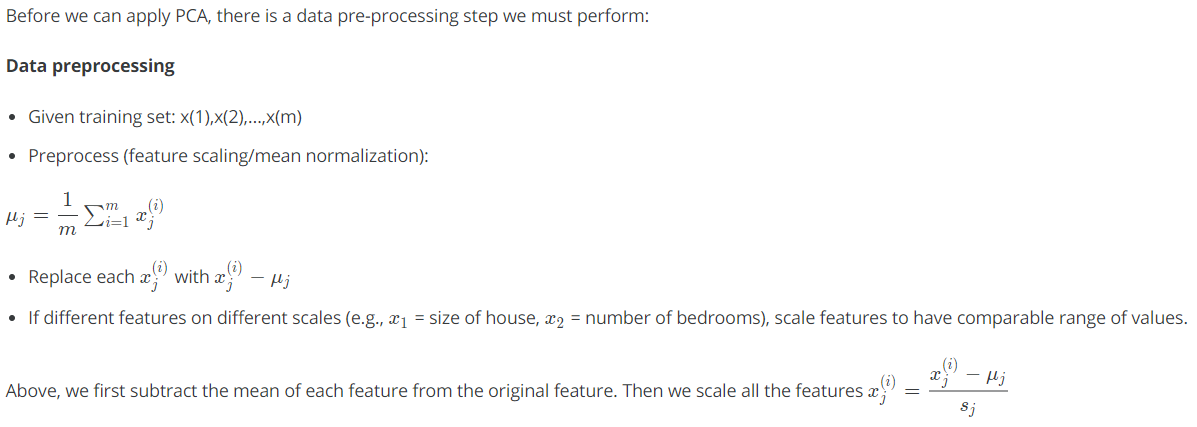
* In linear regression, we are minimizing the **squared error** from every point to our predictor line. These are vertical distances.
* In PCA, we are minimizing the **shortest distance**, or shortest *orthogonal* distances, to our data points.

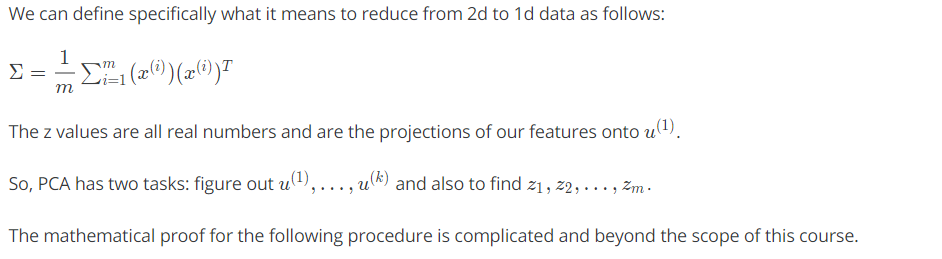
More generally, in linear regression we are taking all our examples in x and applying the parameters in Θ to predict y.

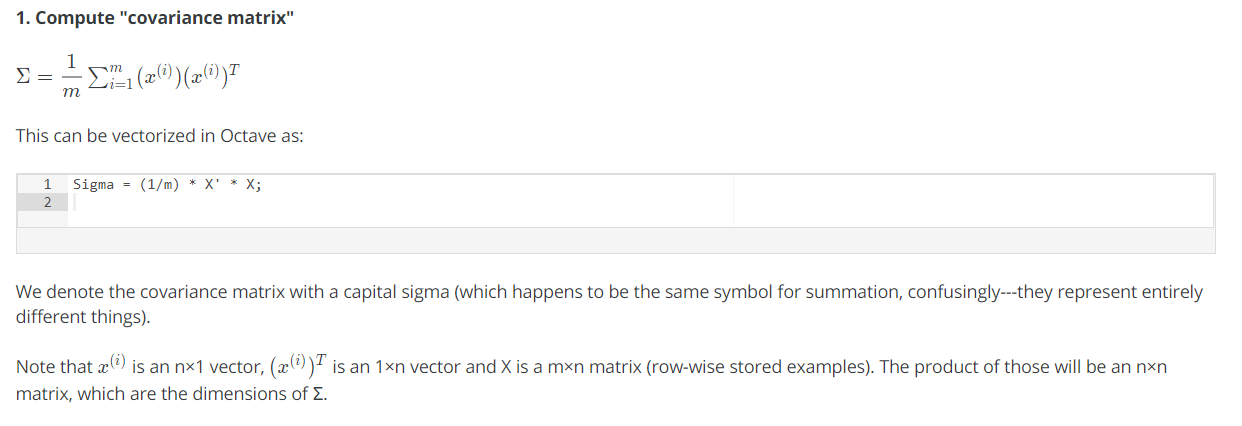
In PCA, we are taking a number of features *x*1​,*x*2​,…,*xn*​, and finding a closest common dataset among them. We aren't trying to predict any result and we aren't applying any theta weights to the features.

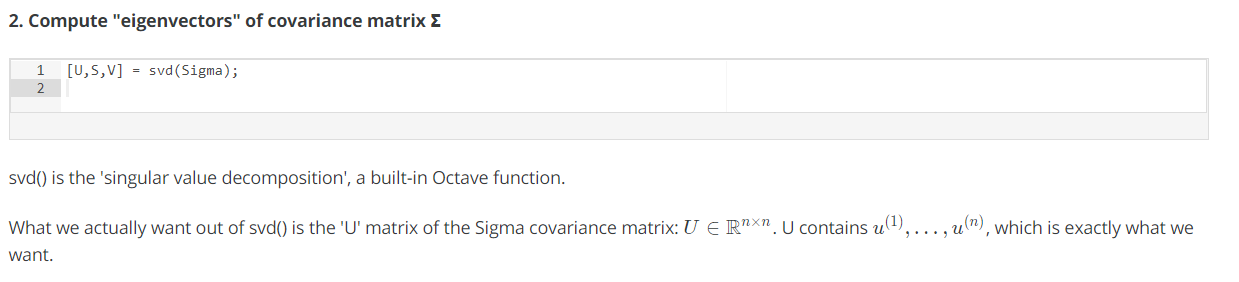


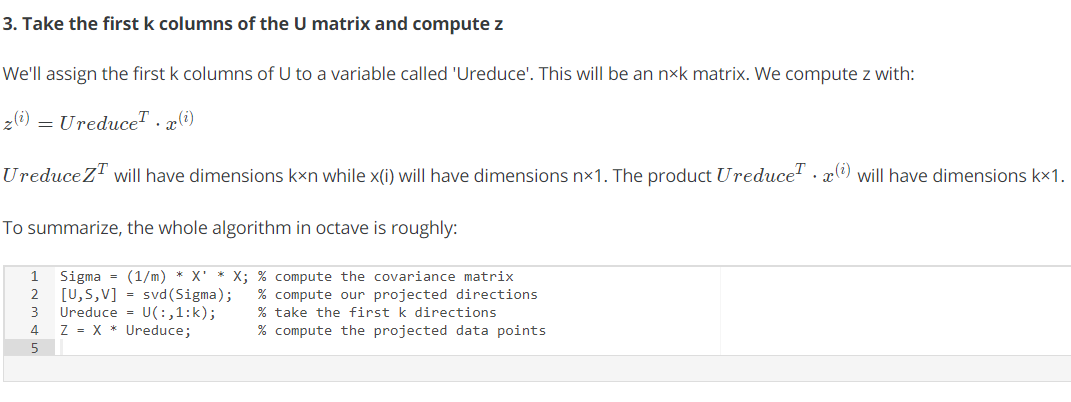
## PCA algorithm





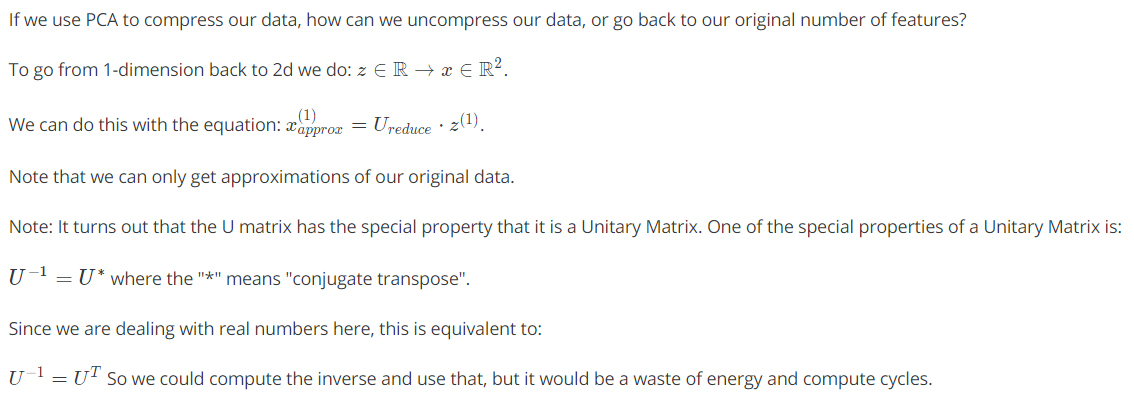


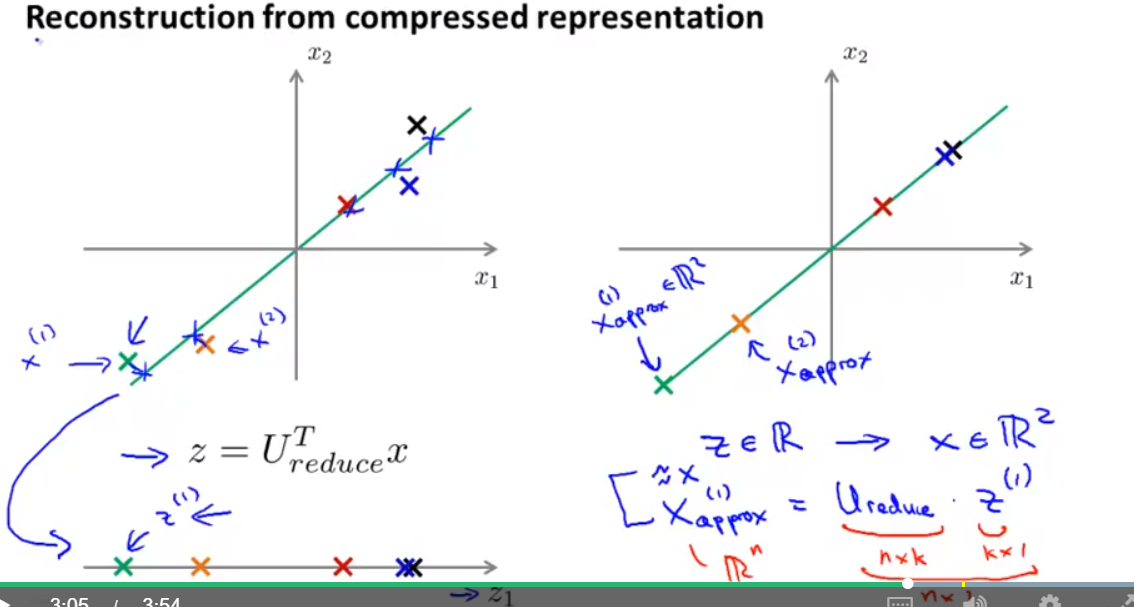




# Applying PCA

## Reconstruction from compressed representation

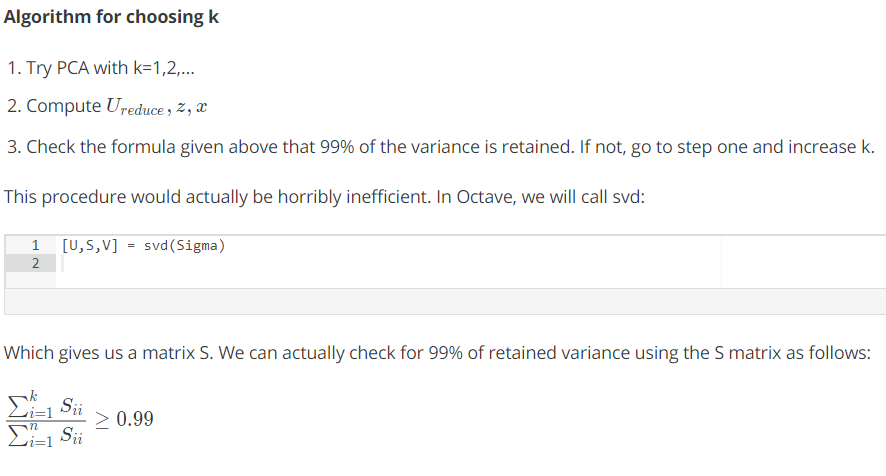


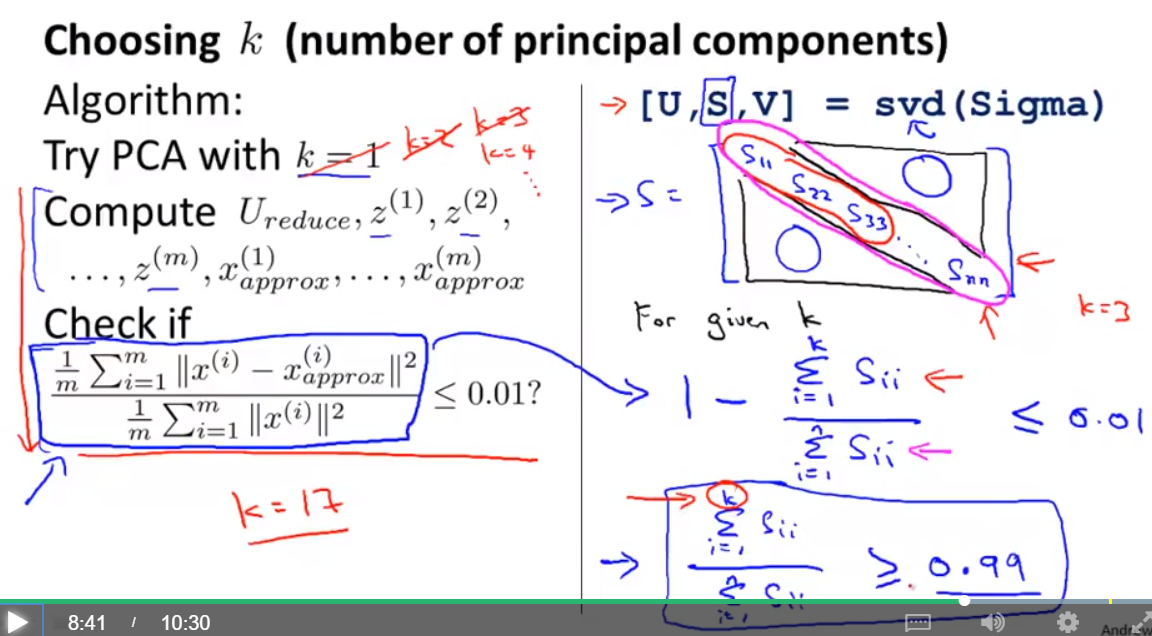


## Choosing the Number of Principal Components



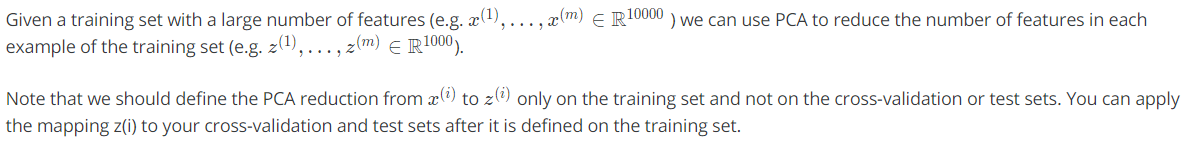
In other words, the squared projection error divided by the total variation should be less than one percent, so that **99% of the variance is retained**.

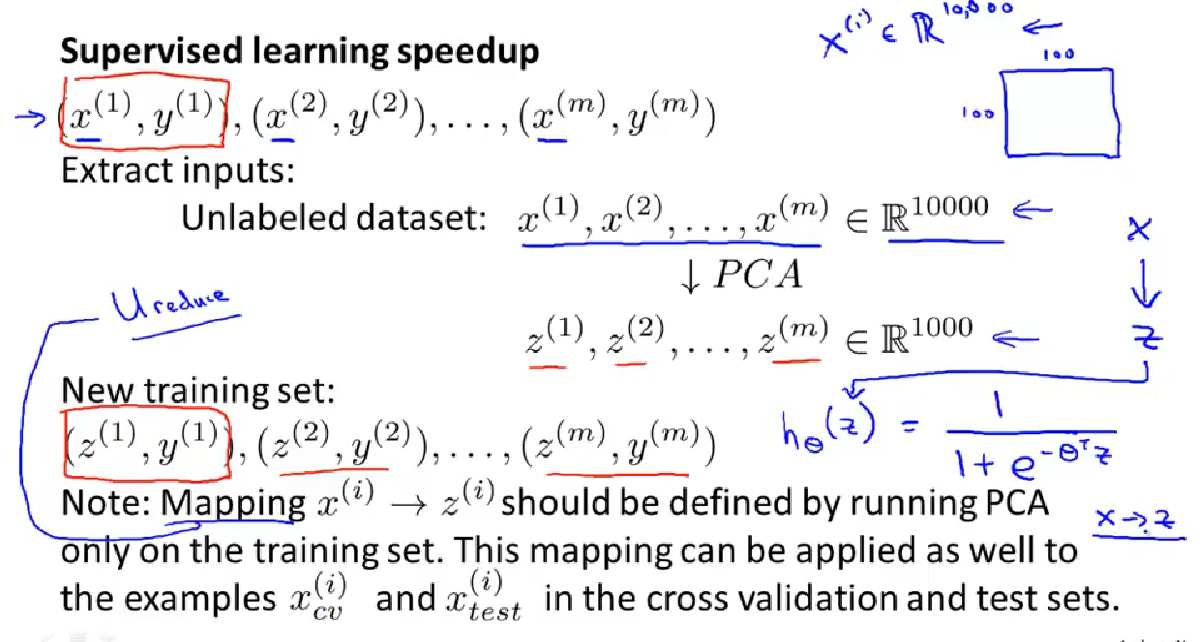


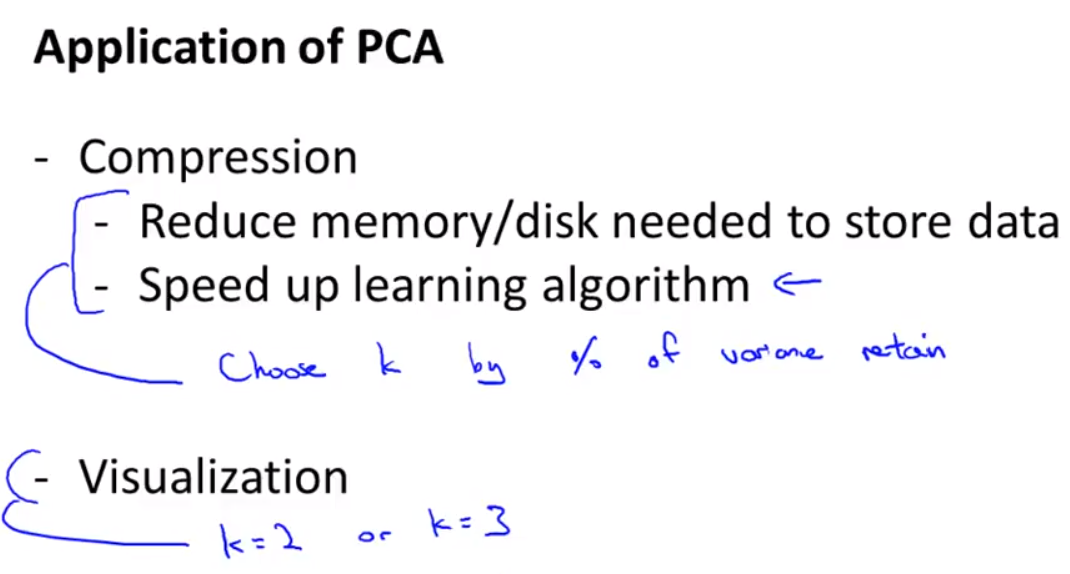


## Advice for applying PCA

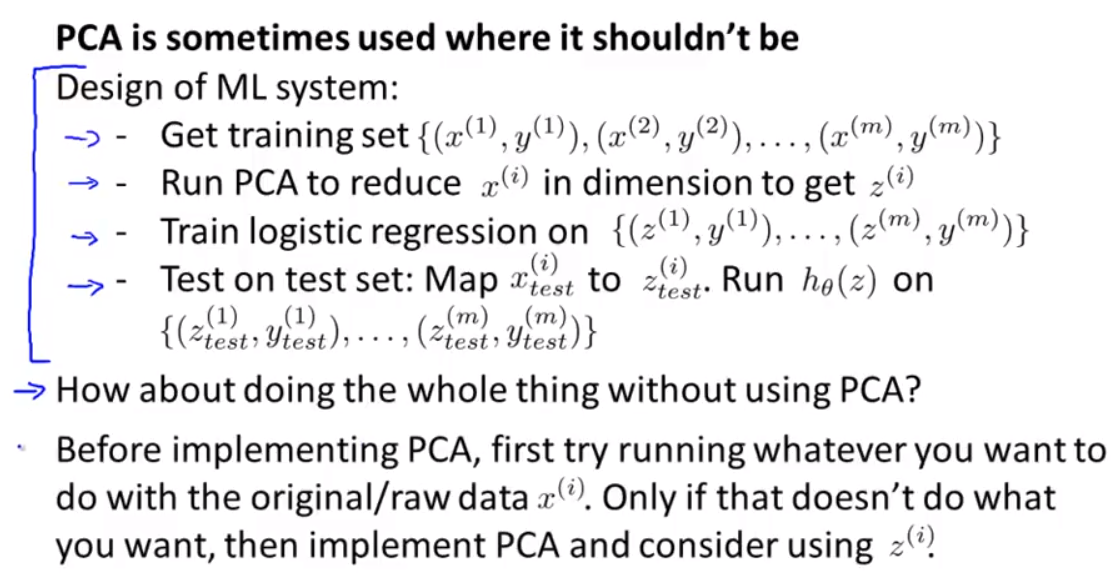
The most common use of PCA is to speed up supervised learning.







**Bad use of PCA:**trying to prevent overfitting. We might think that reducing the features with PCA would be an effective way to address overfitting. It might work, but is not recommended because it does not consider the values of our results y. Using just regularization will be at least as effective.



Don't assume you need to do PCA. **Try your full machine learning algorithm without PCA first.** Then use PCA if you find that you need it.