clustering

- say you're doing a controlled experiment to test a specific hypothesis
    - you have treatment and control data, and you know which is which, i.e. if you plot your data, you know which colour to assign to each point
    - in that case, the question is "are my treatment data significantly different from control", and you use a stats test to answer that
    - there's no clustering required, because your data already come with built-in labels!
- you might collect a bunch of data, with no specific hypothesis that you're testing, no treatment vs. control
    - your data points come without labels, i.e. they're unclustered, all have the same colour
    - you might wonder if your data naturally fall into various clusters/categories, or if they're just one big continuous cloud of smoothly varying data points
    - if they **do** form clusters, then you should probably analyze each cluster separately instead of lumping all your data together
    - clustering is a type of exploratory data analysis
- first step is to plot your data
    - if it's low enough dimensionality (1 or 2 or maybe 3D), then you can inspect it visually and look for clusters
    - if it's very high-dimensional data (e.g. the activity of many simultaneously Ca-imaged neurons), then you'll have to do some dimension reduction before you can visually inspect your data
    - if you see clusters in your data, then one way to label each data point is to manually draw boundaries between clusters, but this can be tedious
- let's look at two example automated clustering methods, and test them on 2D data:

k-means algorithm:

- probably the most commonly used clustering algorithm

0. Randomly initialize a set of cluster centers (i.e. means)
1. Assign each data point to the nearest cluster
2. Update the position of each cluster center by taking the mean of the positions of all its member points. Go to 1.

- After enough iterations, cluster centers will stop moving, and cluster membership of each point will become stable.
- Simple, fast, but it has some limitations:
    - need to specify how many clusters you want it to find (hence the 'k' in k-means)
    - because it uses only distance to assign points to clusters, it performs poorly for elongated clusters
- *k-means demo and exercises*

DBSCAN algorithm:

- DBSCAN = "Density-based spatial clustering of applications with noise"

- density-based instead of just distance based
- does better than k-means for elongated clusters
- figures out the number of clusters automatically, but it has two other parameters that have to be tweaked
- doesn't require that every point be assigned to a cluster - allows for outliers
- *DBSCAN demo and exercises*
- lots of other clustering algorithms in `sklearn.cluster`, see:
    - http://scikit-learn.org/stable/modules/clustering.html
    - http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html
- an even better, simpler density-based algorithm:
    - "Clustering by fast search and find of density peaks", Rodriguez and Laio, Science, 2014
    - http://science.sciencemag.org/content/344/6191/1492
    - unfortunately, no good Python library for it (yet)

dimension reduction

- say you're recording activity from 20 neurons simultaneously
- your whole dataset is 20 dimensional, and can be described by an nsamples x 20 array:
    - one sample timepoint per row, one column per neuron, activity level at each entry
- the activity of some of those neurons might be correlated, and therefore somewhat redundant
- you can't visualize data in 20D space, but you can if it's 2D or 3D
- dimension reduction algorithm can look for redundancy in the data and project it into a new smaller dimensional space that still captures the original data fairly well, without throwing away too much information
- most common kind is PCA: principal components analysis
    - PCA looks for directions of maximum variance in the data, and rotates the axes in such a way that makes them best explain the variance
- *PCA demo*
- lots of other kinds of dimension reduction, or "decompositions", in `sklearn.decomposition`:
    - http://scikit-learn.org/stable/modules/decomposition.html
    - very nice description of PCA, by former neuroscientist Jonathan Shlens:
        - "A Tutorial on Principal Component Analysis": https://arxiv.org/abs/1404.1100