BGGN 213

Unsupervised Learning

Lecture 8

Barry Grant

UC San Diego

http://thegrantlab.org/bggn213

Recap of Lecture 7

- Reviewed the major steps for R function writing
- Introduced the CRAN & Bioconductor repositories for packages that extend R functionality
- Explored installing and using the ggplot2, bio3d, rgl, rentrez, igraph, blogdown, shiny, and circlize packages
- Discussed how to judge the utility, usability and development status of R packages.

[Muddy Point Feedback Link]

Today's Menu

- Introduction to machine learning
  - <u>Unsupervised</u>, supervised and reinforcement learning

- Clustering
  - K-means clustering
  - Hierarchical clustering
  - Heatmap representations

- Dimensionality reduction, visualization and 'structure' analysis
  - Principal Component Analysis (PCA)

- Hands-on application to cell classification

Types of machine learning

- Unsupervised learning
  - ➡ Finding structure in unlabeled data
- Supervised learning
  - ➡ Making predictions based on labeled data
  - ➡ Predictions like regression or classification
- Reinforcement learning
  - ➡ Making decisions based on past experience

Types of machine learning

- Unsupervised learning
  - ➡ Finding structure in unlabeled data
- Supervised learning
  - ➡ Making predictions based on labeled data
  - ➡ Predictions like regression or classification
- Reinforcement learning
  - ➡ Making decisions based on past experience

Today's Menu

- Introduction to machine learning
  - Unsupervised, supervised and reinforcement learning

- Clustering
  - K-means clustering
  - Hierarchical clustering
  - Heatmap representations

- Dimensionality reduction, visualization and ‘structure’ analysis
  - Principal Component Analysis (PCA)

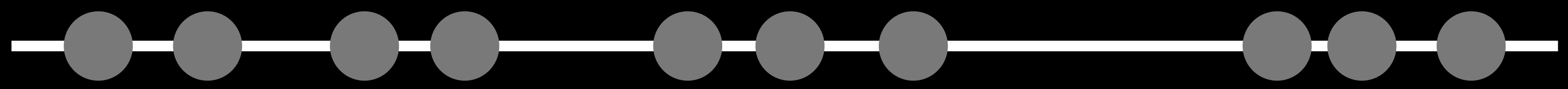
- Hands-on application to cell classification

k-means clustering algorithm

- Breaks observations into ***k*** pre-defined number of clusters
- You define ***k*** the number of clusters!

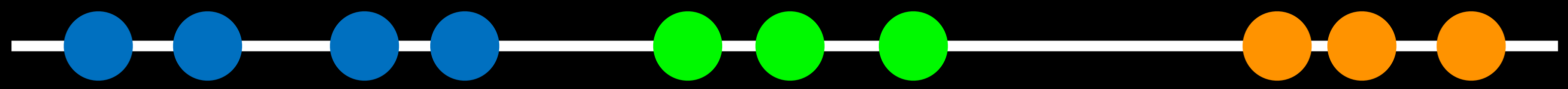
k-means clustering algorithm

- Breaks observations into ***k*** pre-defined number of clusters
- You define ***k*** the number of clusters!
  - ➡ Imagine you had data that you could plot along a line and you knew you had to put them into *k*=3 “clusters” (e.g. data from three types of tumor cells)



k-means clustering algorithm

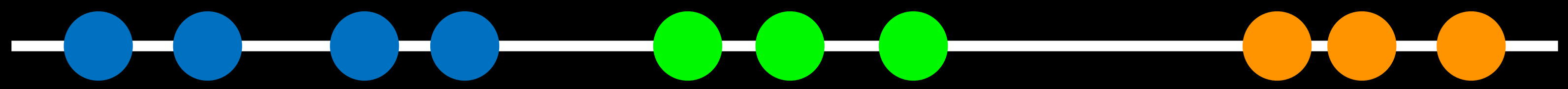
- Breaks observations into ***k*** pre-defined number of clusters
- You define ***k*** the number of clusters!
  - ➡ Imagine you had data that you could plot along a line and you knew you had to put them into *k*=3 "clusters" (e.g. data from three types of tumor cells)



Here your eyes can clearly see 3 natural groupings

k-means clustering algorithm

- Breaks observations into ***k*** pre-defined number of clusters
- You define ***k*** the number of clusters!
  - ➡ Imagine you had data that you could plot along a line and you knew you had to put them into k=3 "clusters" (e.g. data from three types of tumor cells)

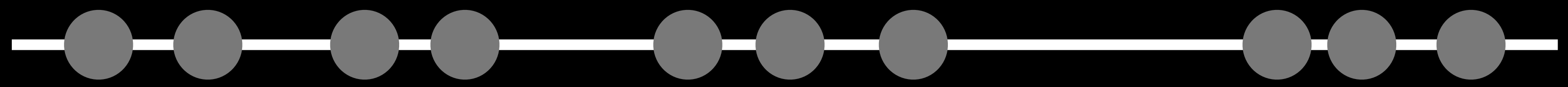


Here your eyes can clearly see 3 natural groupings

How does k-means attempt to define this grouping?

Step 1.

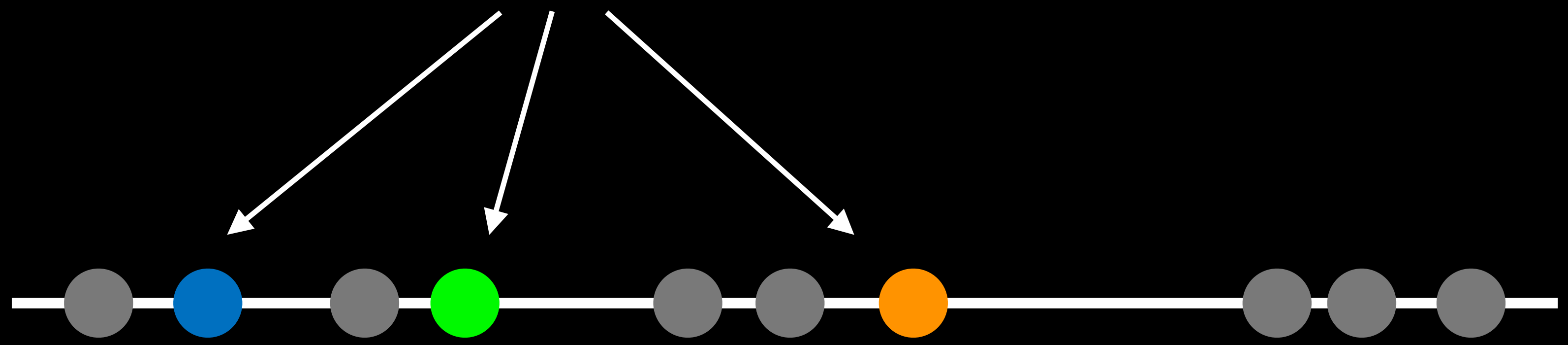
Select ***k*** (the number of clusters)



Step 2.

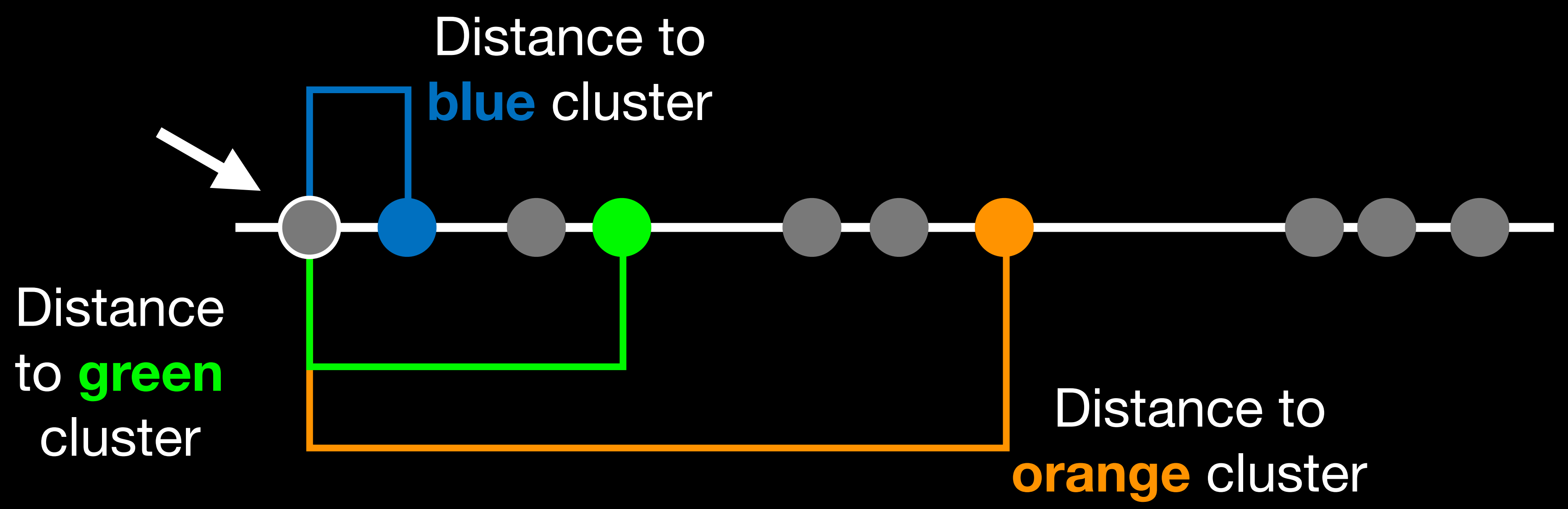
Select ***k=3*** distant data points at random

These are the initial clusters



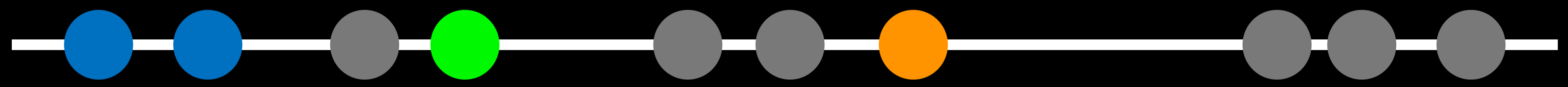
Step 3.

Measure distance between the 1st point and the ***k=3*** initial clusters



Step 4.

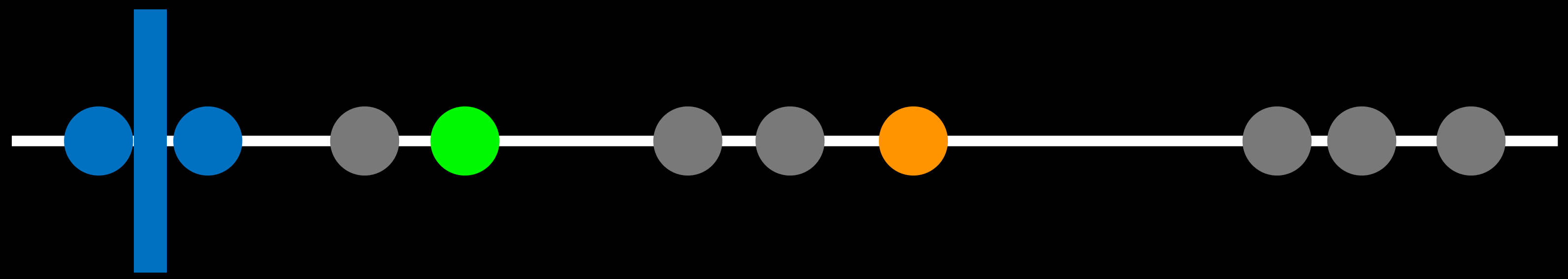
Assign the 1st point to the nearest cluster



Step 5.

Update cluster centers

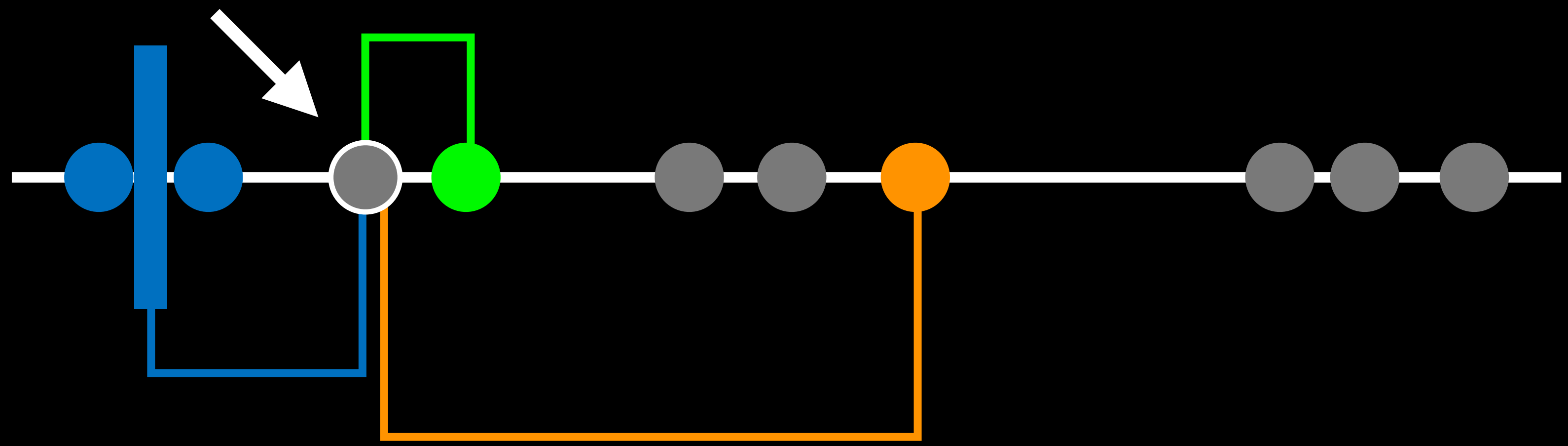
Calculate the mean value for the blue cluster including the new point



Step 6.

Assign next point to closest cluster

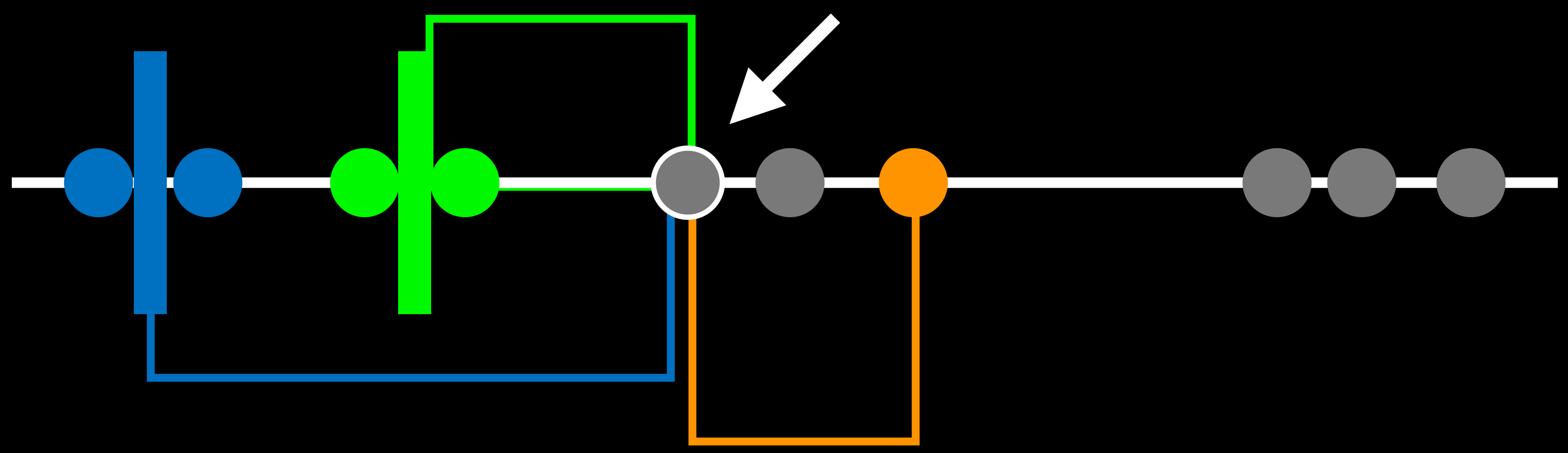
Use updated cluster centers for distance calculation



Step 7.

Update cluster centers and move to next point

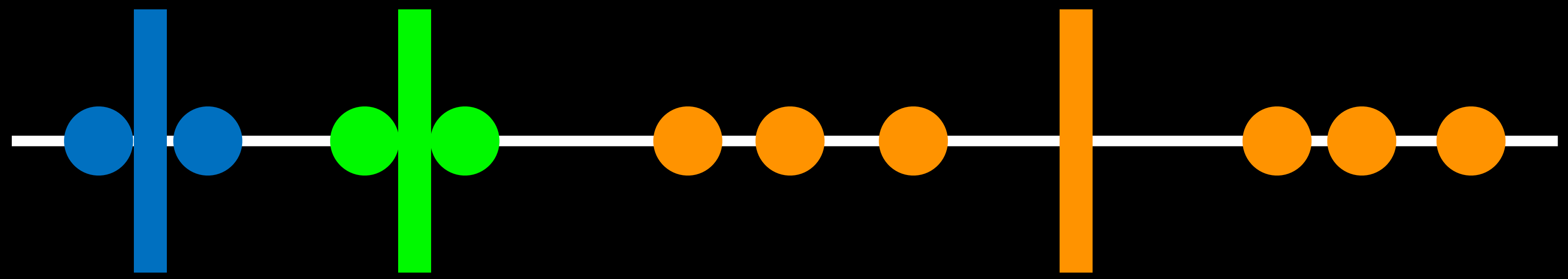
Use updated cluster centers for distance calculation



Step 8.

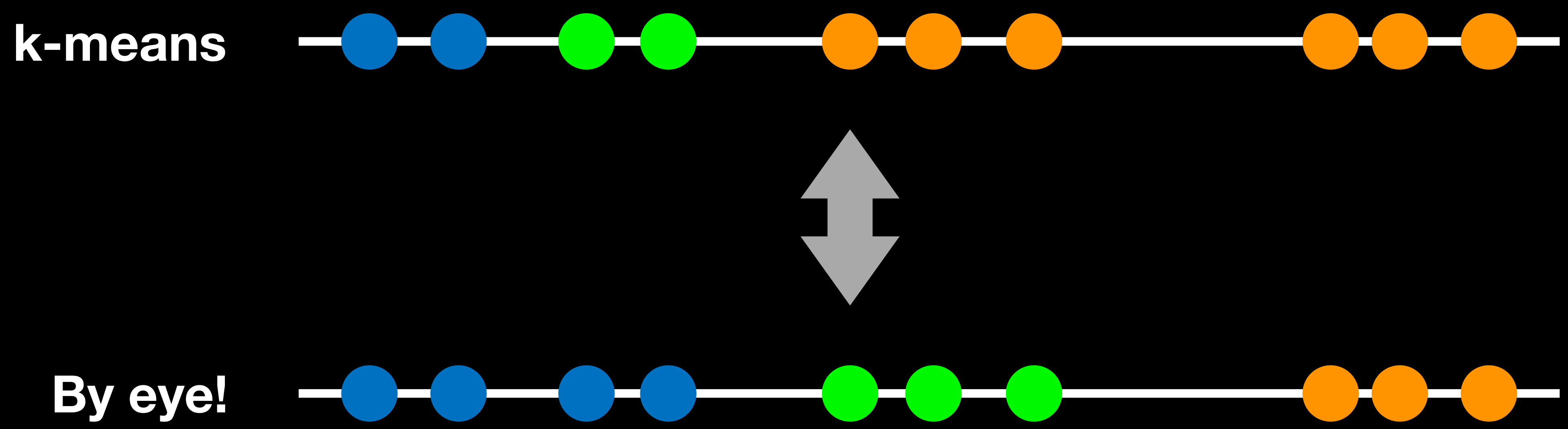
Repeat for each point

Each time updating cluster centers



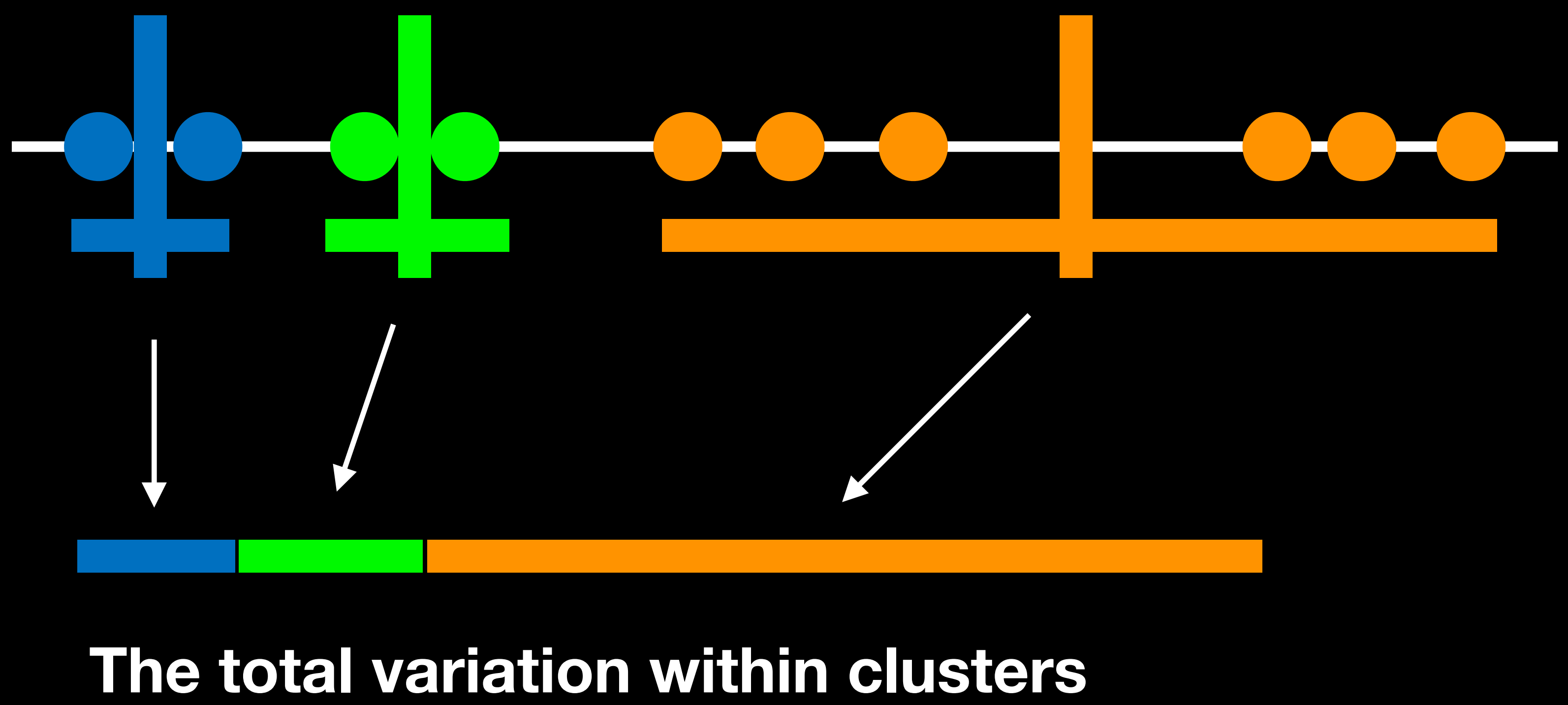
Hmm….

Here the k-means result does not look as good as what we were able to do by eye!



Step 9.

Assess the quality of the clustering by adding up the variation within each cluster

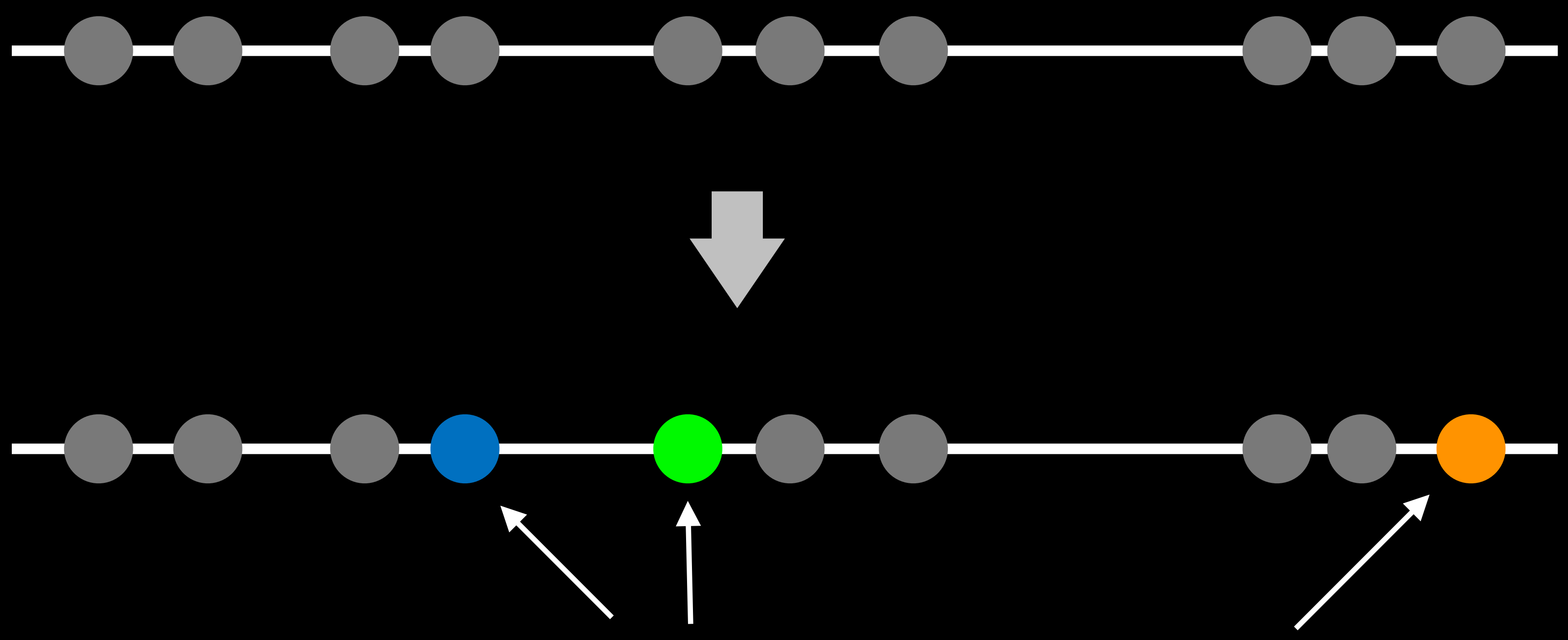


K-means keeps track of these clusters and their total **variance** and then does the whole thing over again with different starting points

Step 10.

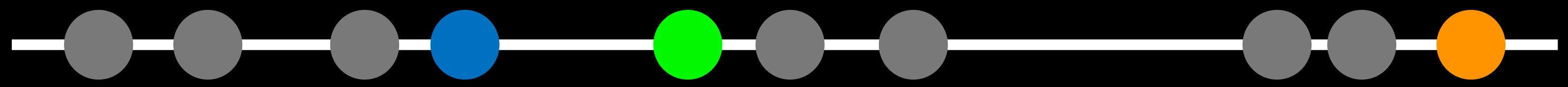
Repeat with different starting points

Back to the beginning and do all steps over again…

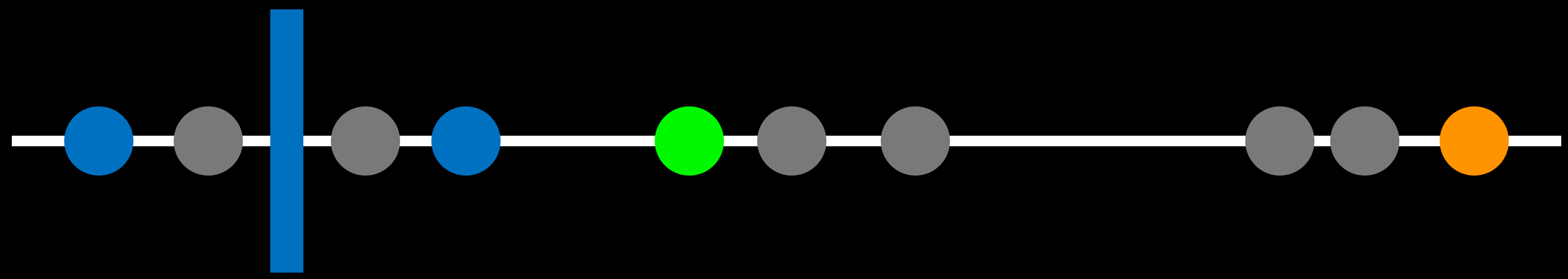


Pick new points as “initial” clusters

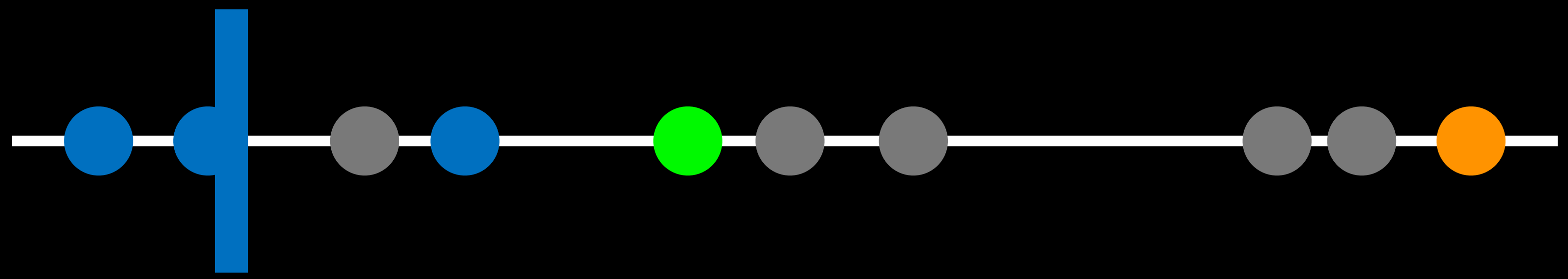
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



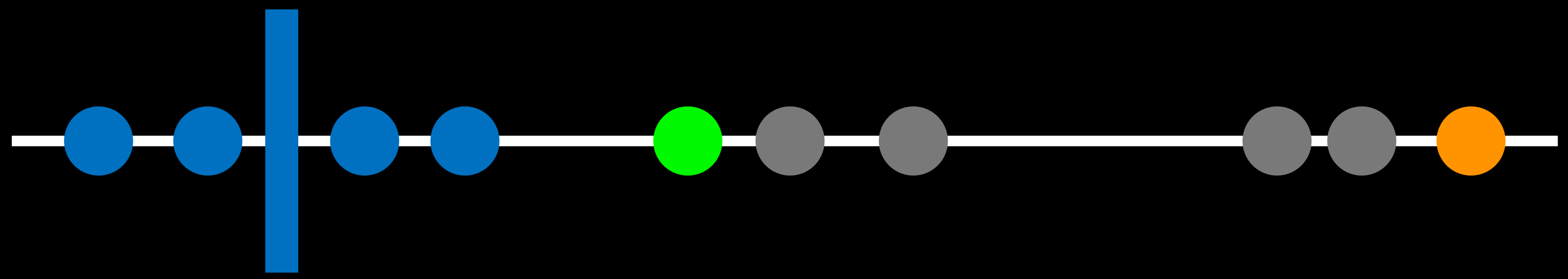
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



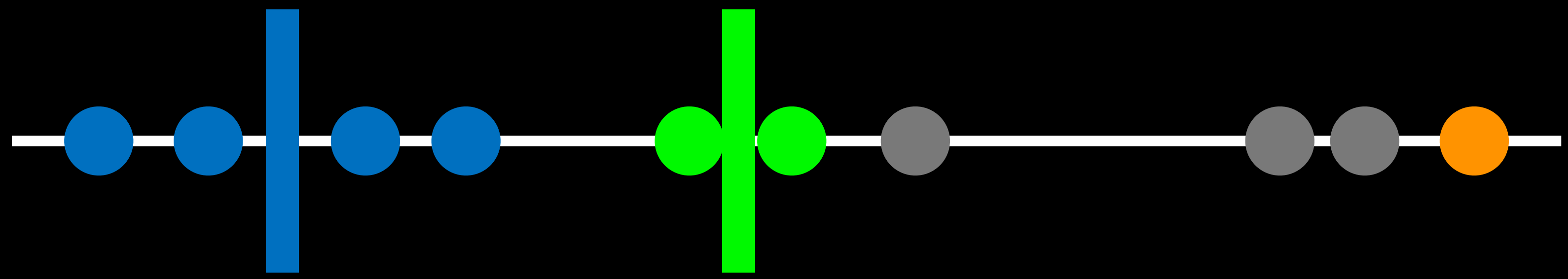
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



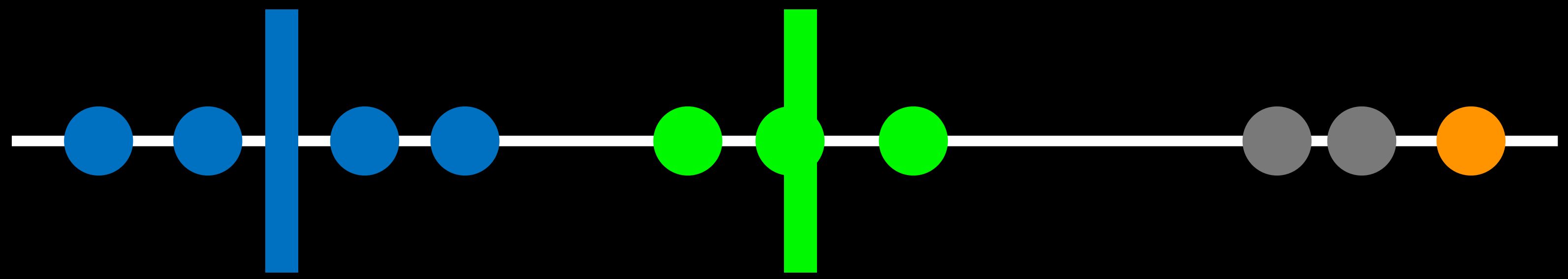
…Pick k=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



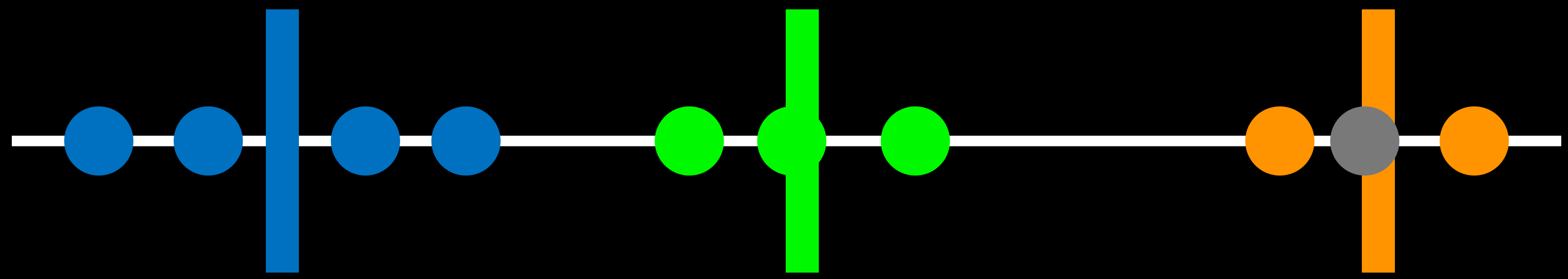
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



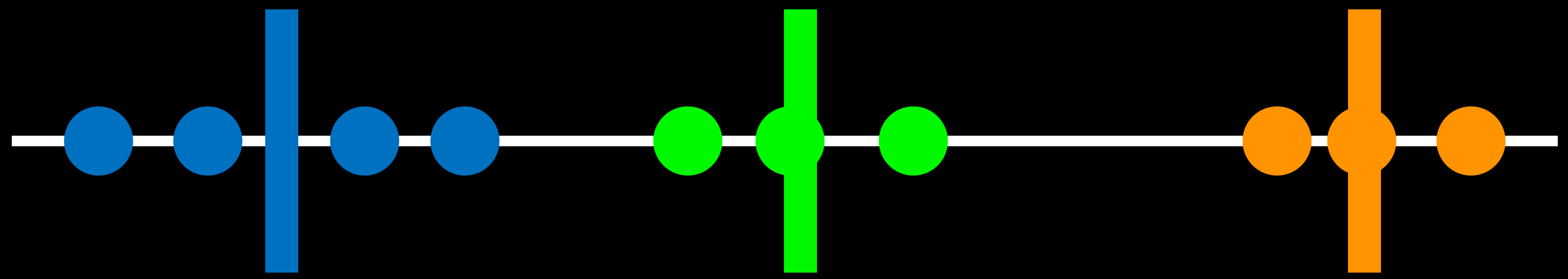
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



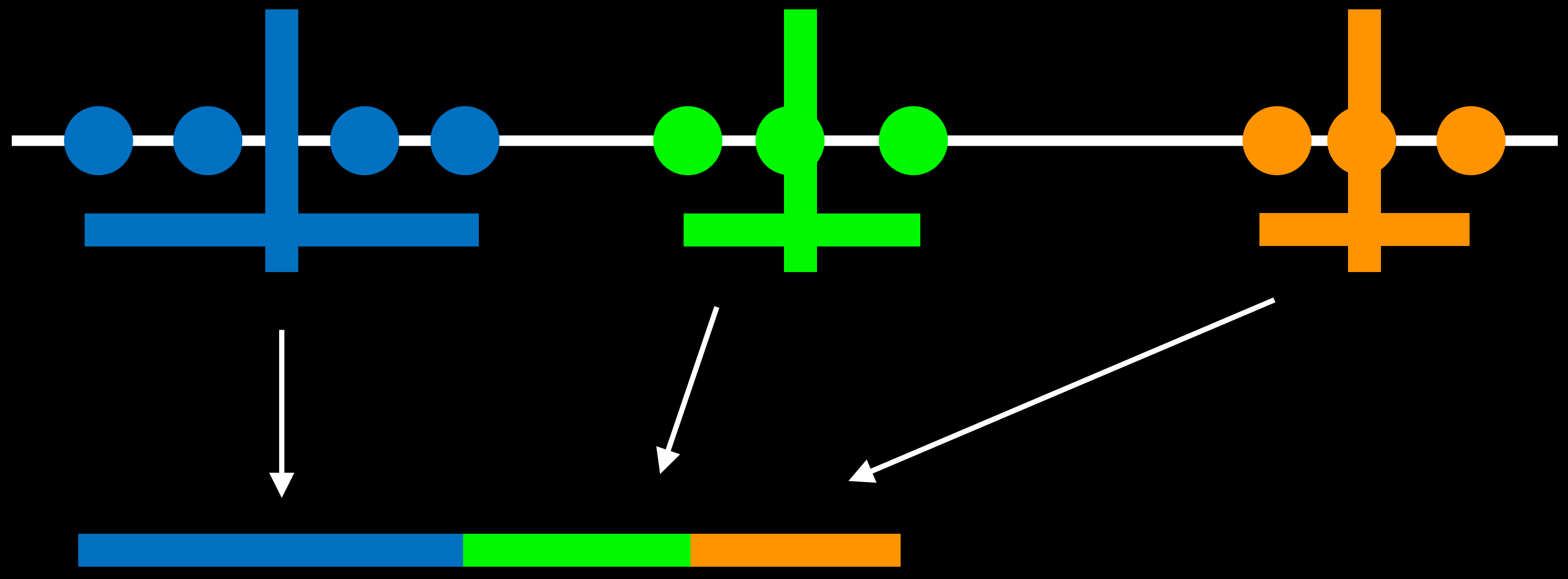
…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



…Pick *k*=3 initial clusters and add the remaining points to the cluster with the nearest mean, recalculating the mean each time a new point is added…



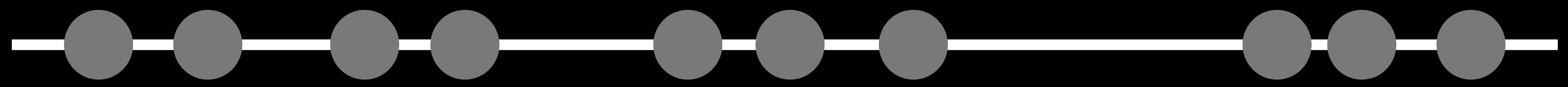
Now the data are all assigned to clusters, we again sum the variation within each cluster



The total variation within clusters

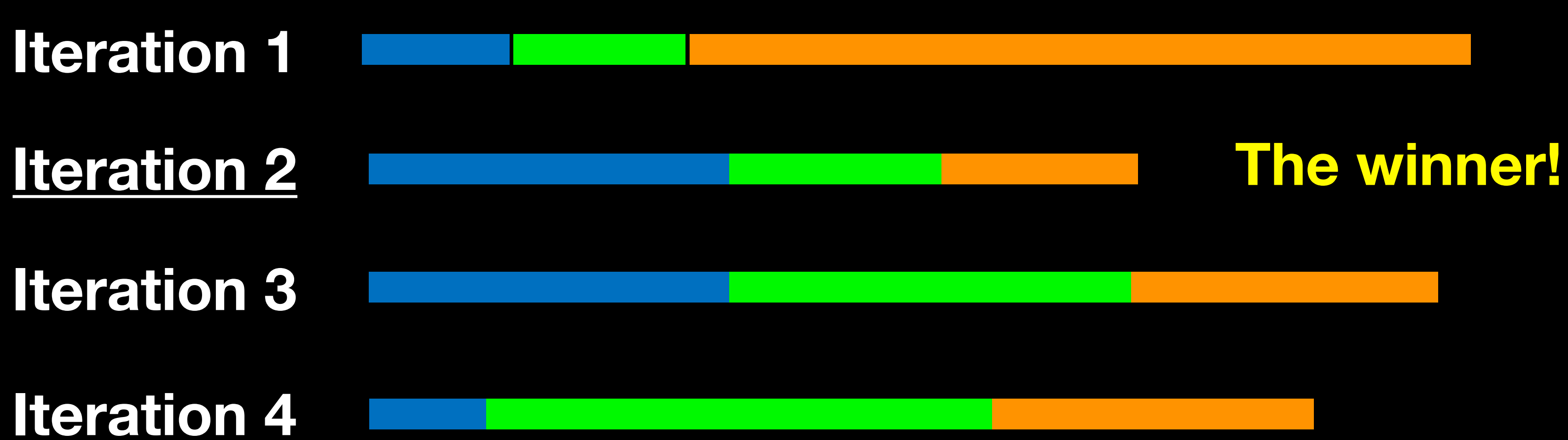
Step 10.

Repeat again with different starting points



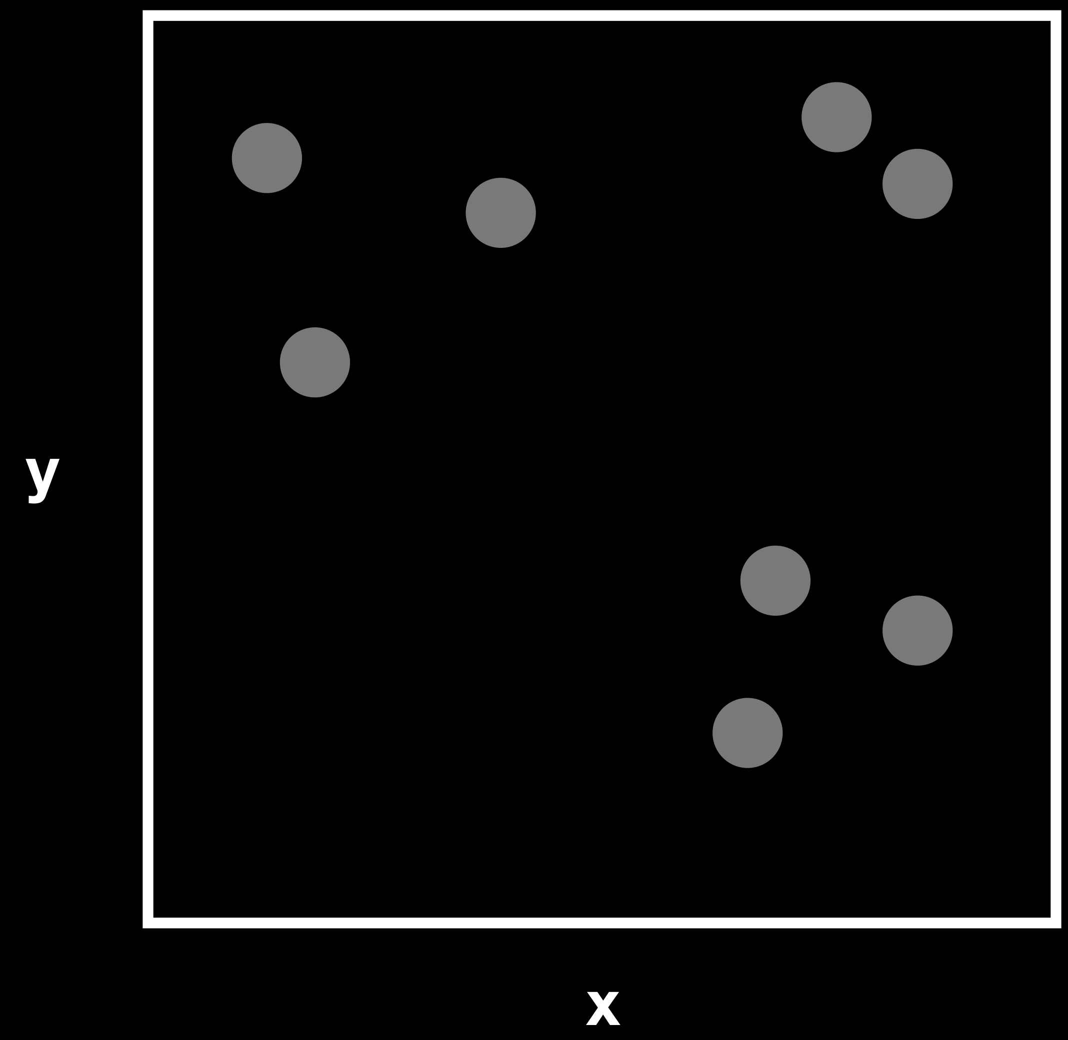
After several iterations k-means clustering knows it has the *best clustering so-far* based on the smallest total variation with clusters.

However, it does not know if it has found the *best overall*. So it will perform several more iterations with different starting points…

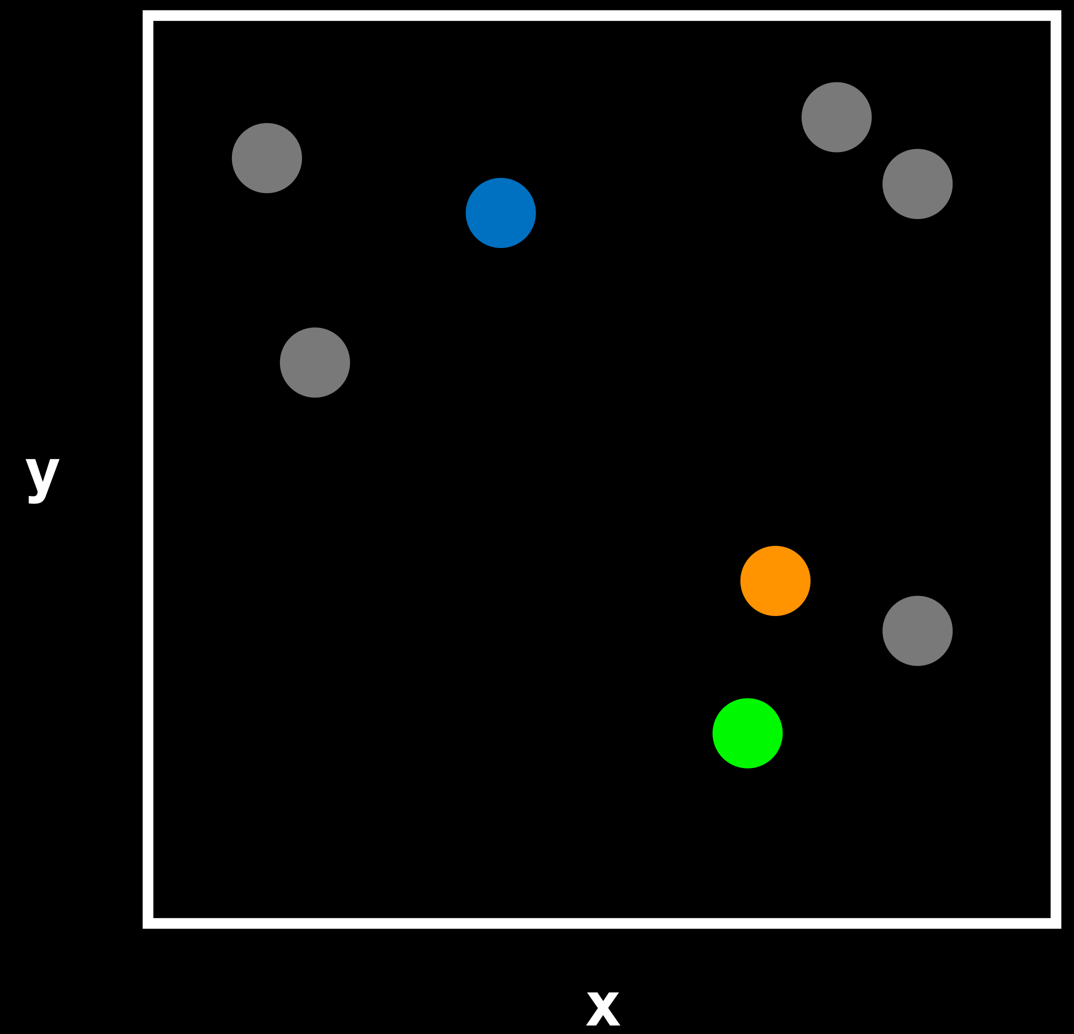


The total variation within clusters

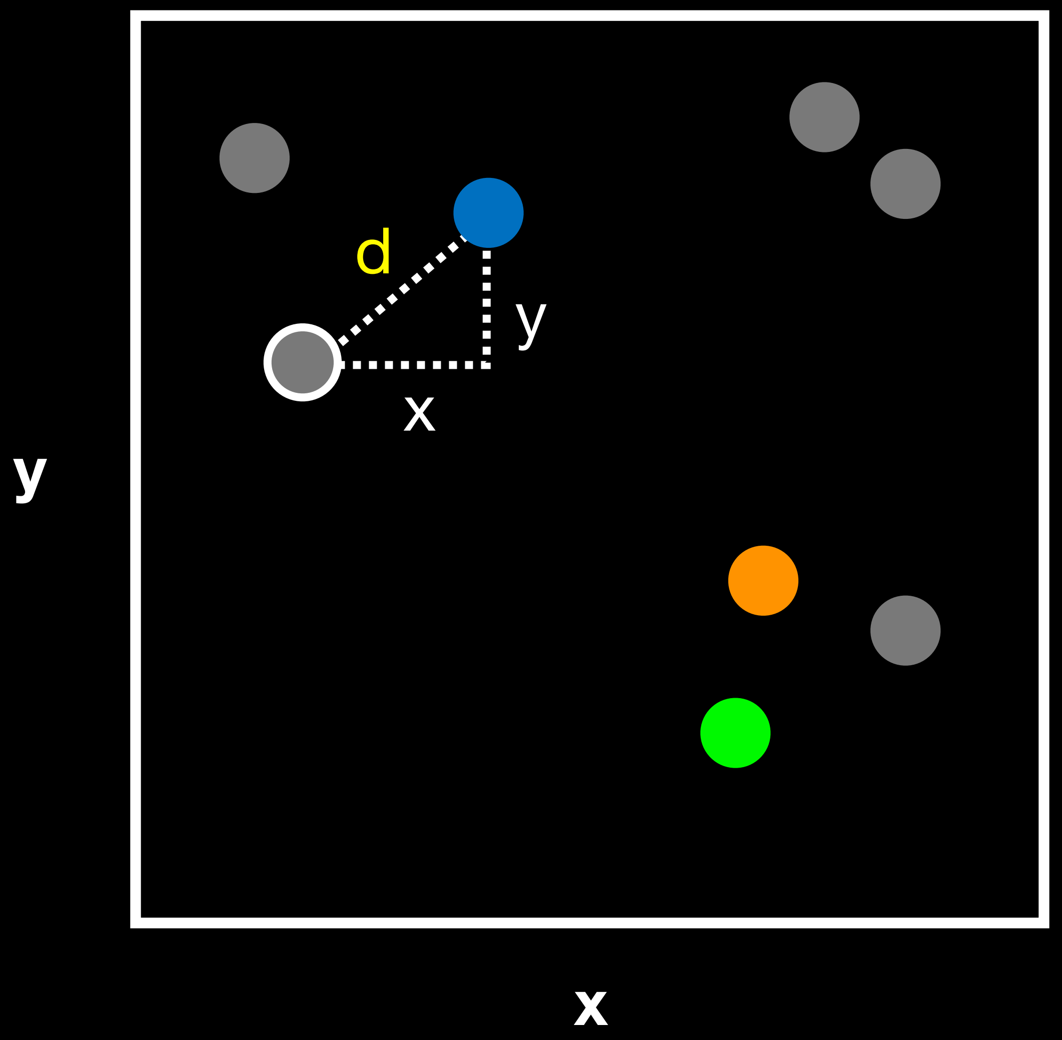
What if we have more dimensions?



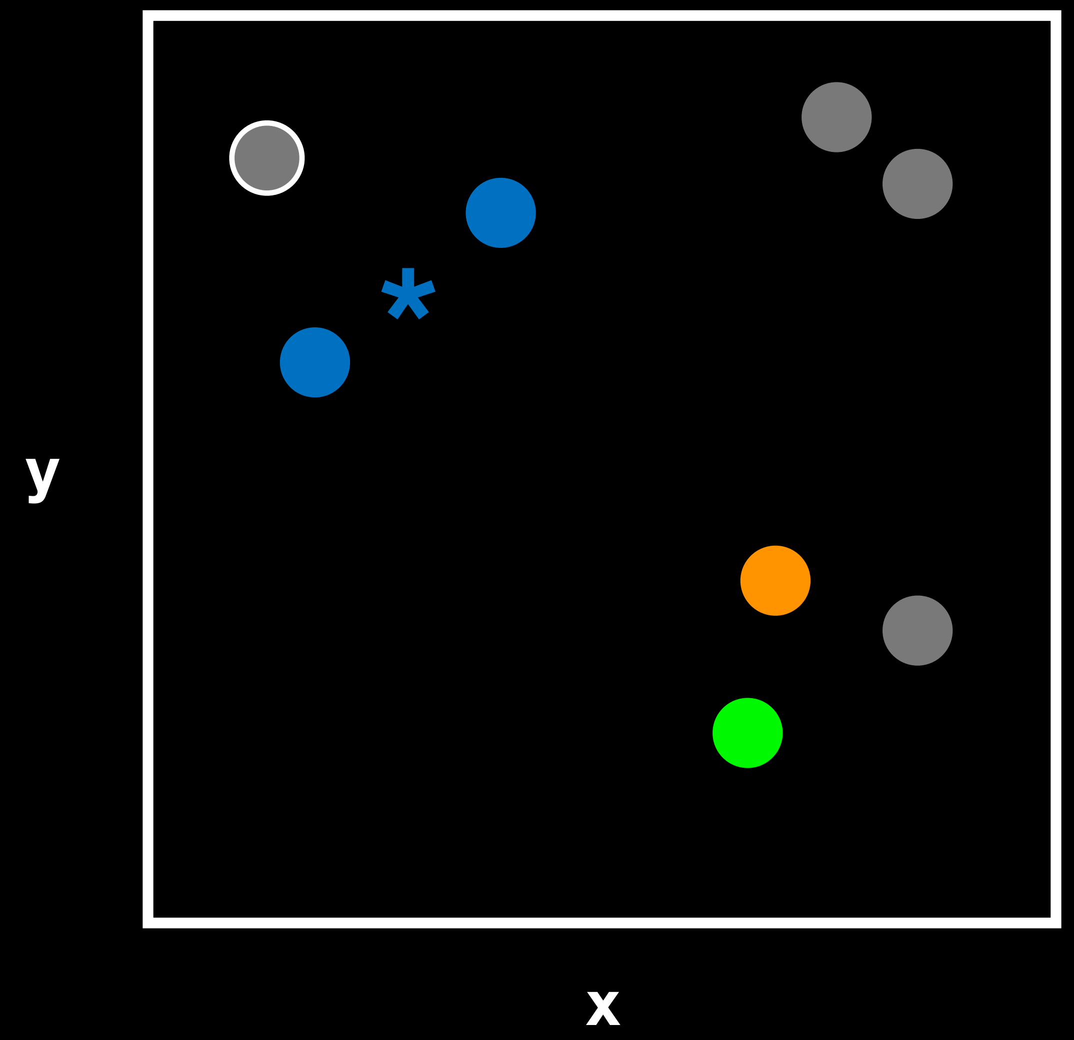
Just like before, we pick 3 random points…



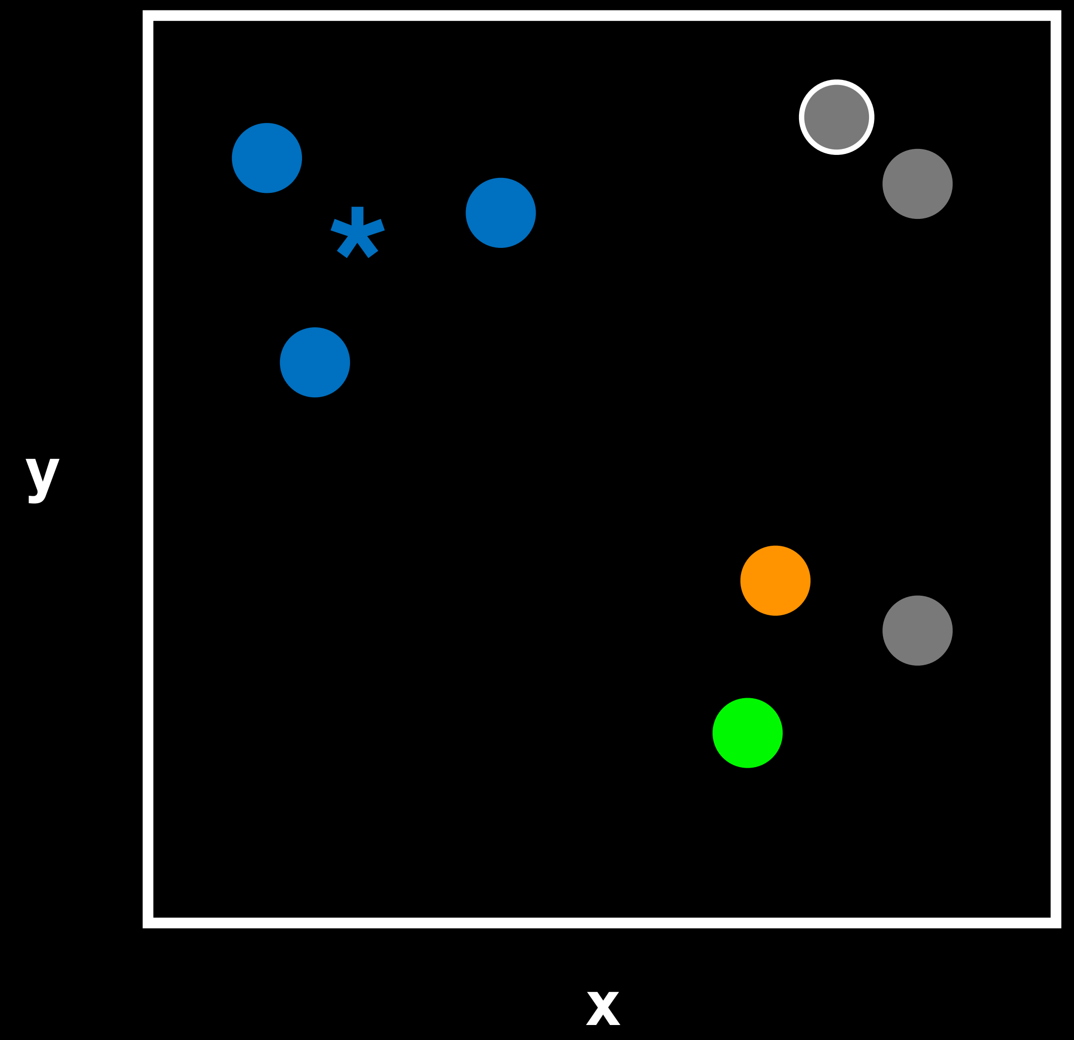
…and use the Euclidean distance.
In 2 dimensions the Euclidean distance is the same as the Pythagorean theorem d = sqrt(x^2 + y^2)



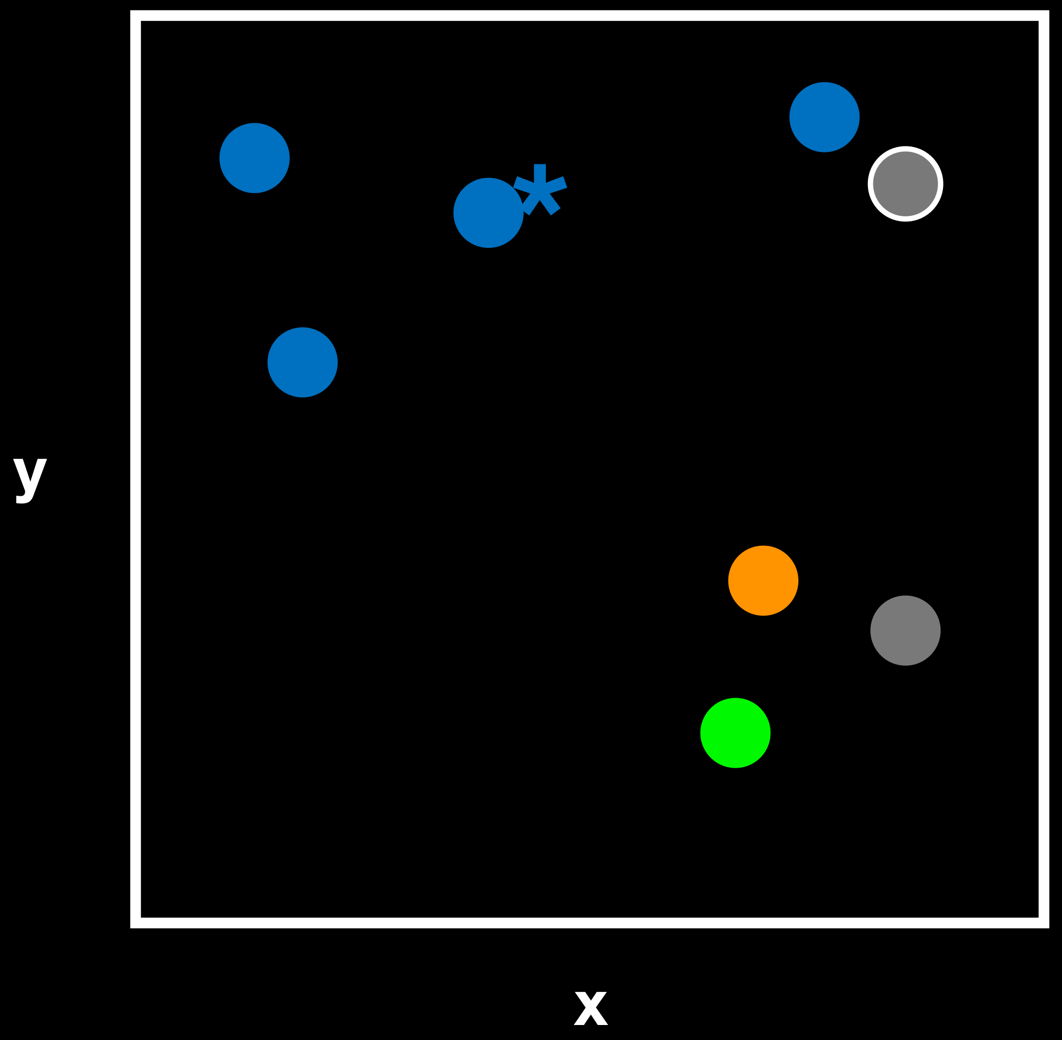
…assign point to nearest cluster and update cluster center *



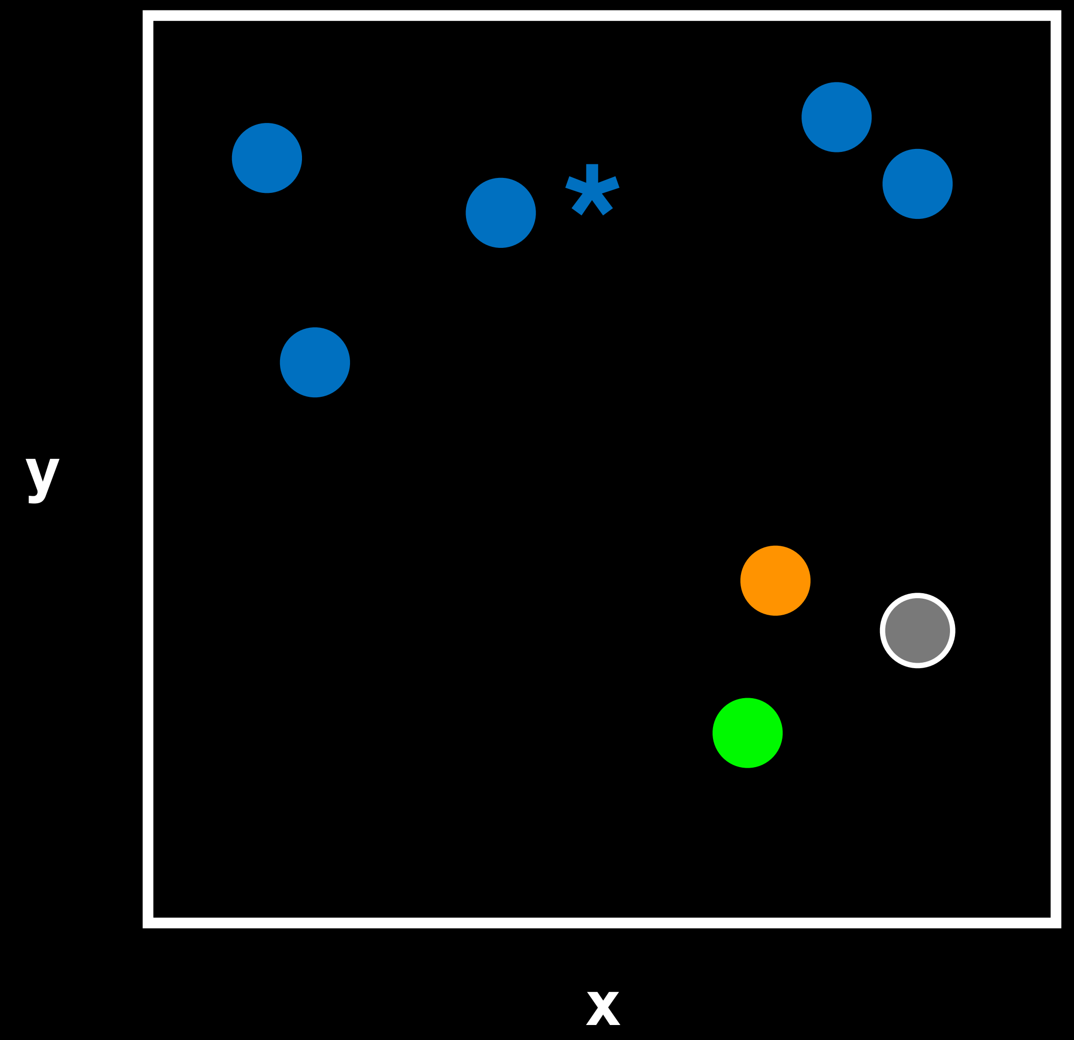
…and continue



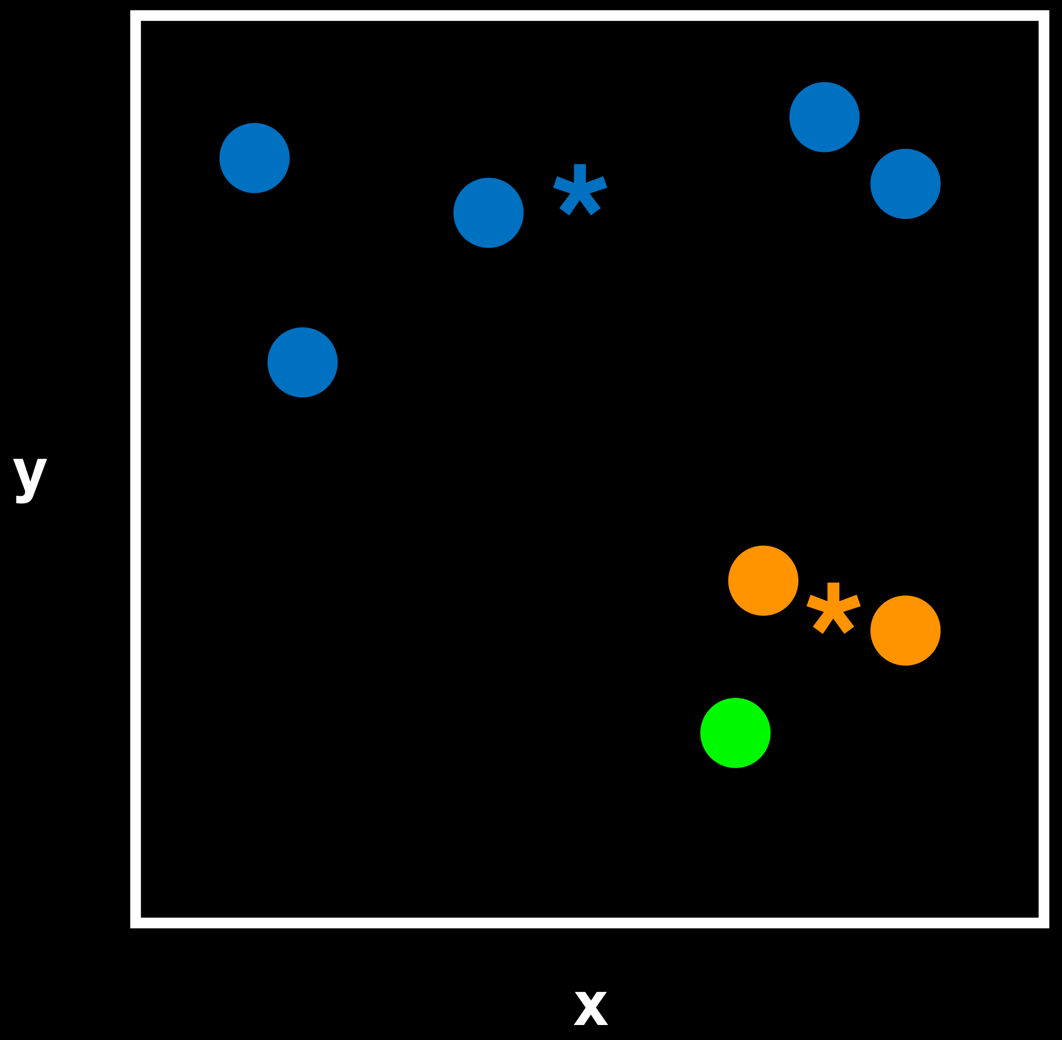
…and continue



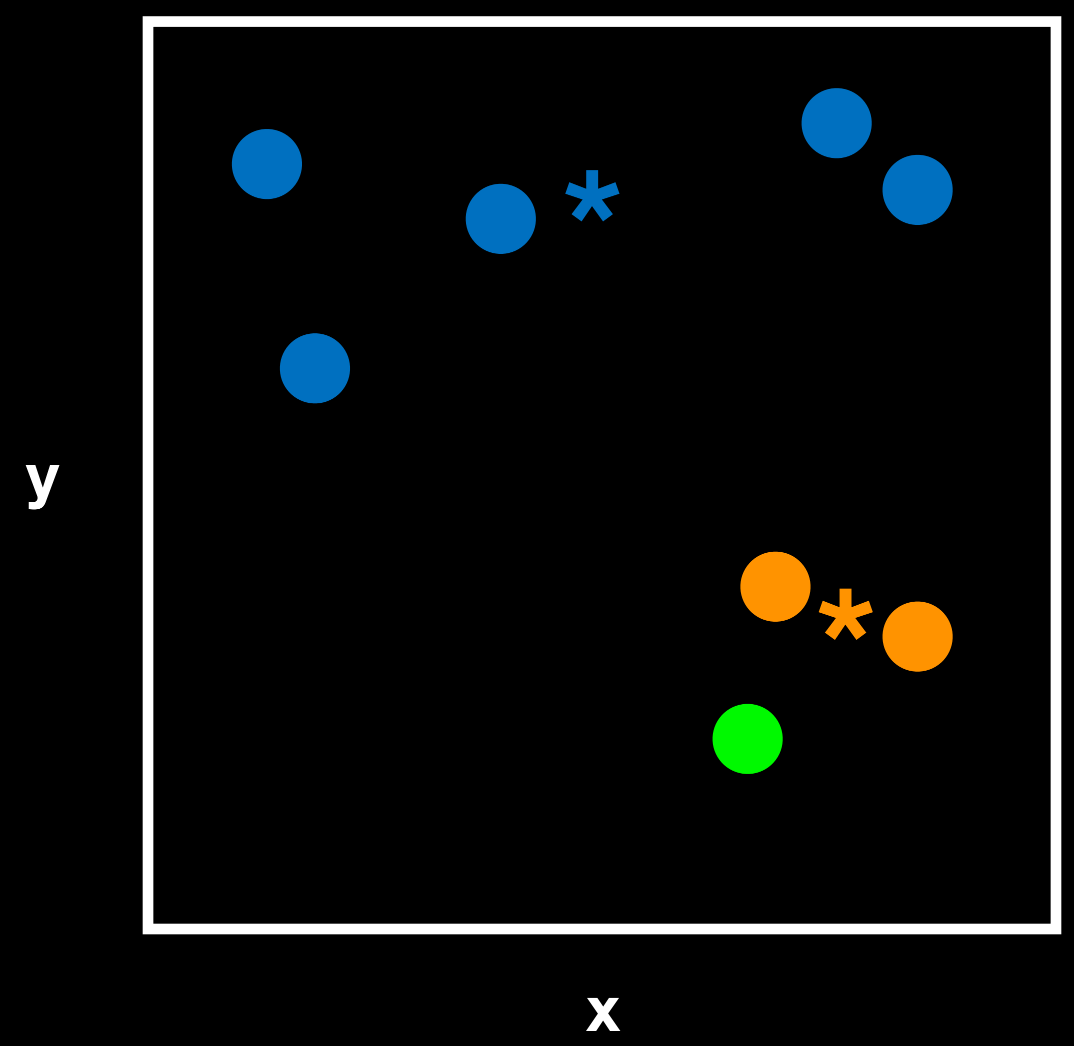
…and continue



…and continue



Again we have to use a number of different starting conditions before deciding on a good clustering!



What if we have even more dimensions?

Cell Samples

	#1	#2	#3
Gene 1	12	6	-13
Gene 2	-7	13	10
Gene 3	8	6	-9
Gene 4	9	5	-11
Gene 5	-3	1	6
Gene 6	10	4	-8

What if we have even more dimensions?

Cell Samples

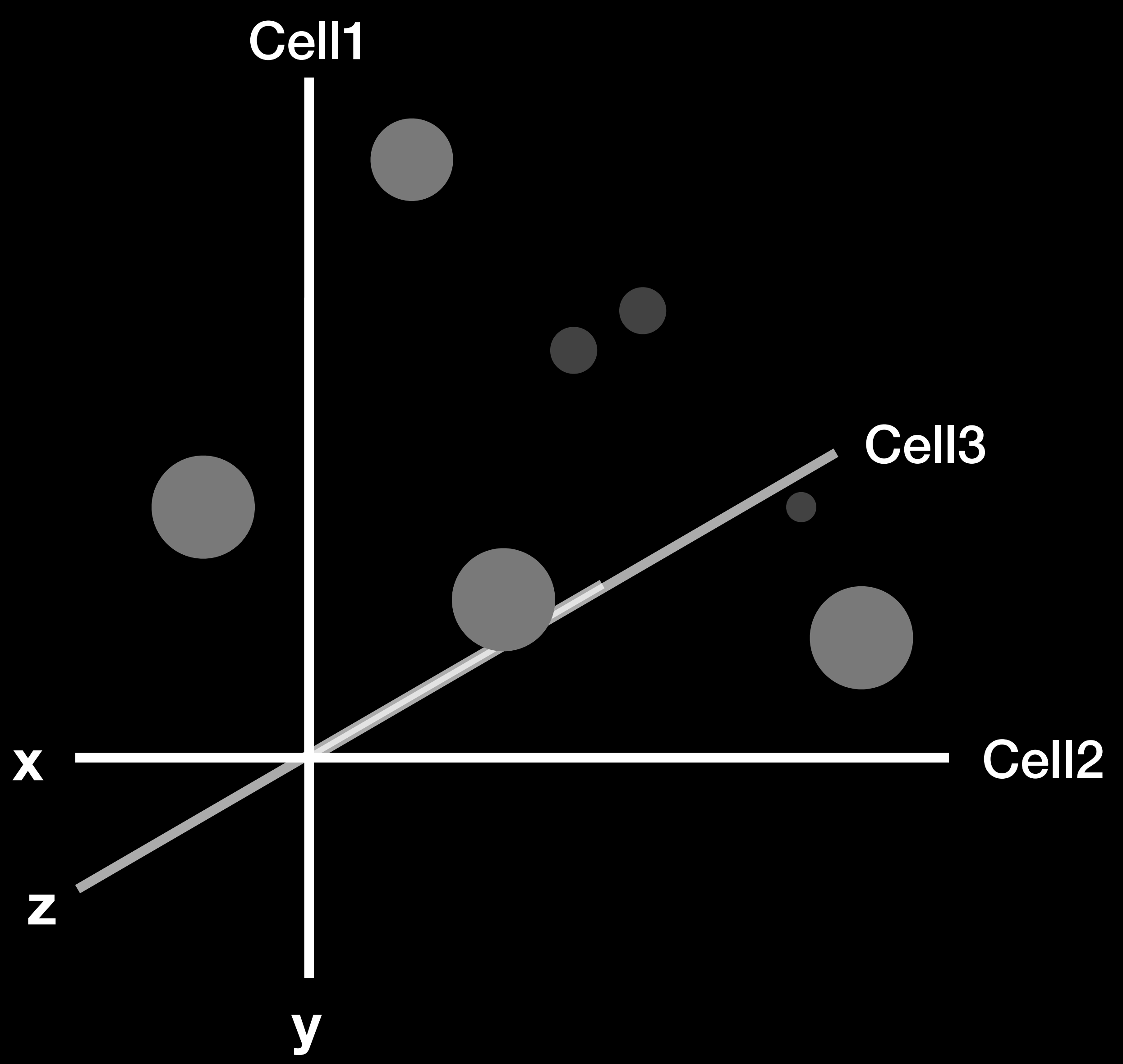
	#1	#2	#3
Gene 1	12	6	-13
Gene 2	-7	13	10
Gene 3	8	6	-9
Gene 4	9	5	-11
Gene 5	-3	1	6
Gene 6	10	4	-8
	x	**y**	**z**

We could simply plot them by relabeling the cell samples as **x**, **y**, and **z** (i.e. a 3D plot)

What if we have even more dimensions?

Cell Samples

	#1	#2	#3
Gene 1	12	6	-13
Gene 2	-7	13	10
Gene 3	8	6	-9
Gene 4	9	5	-11
Gene 5	-3	1	6
Gene 6	10	4	-8
	x	y	z

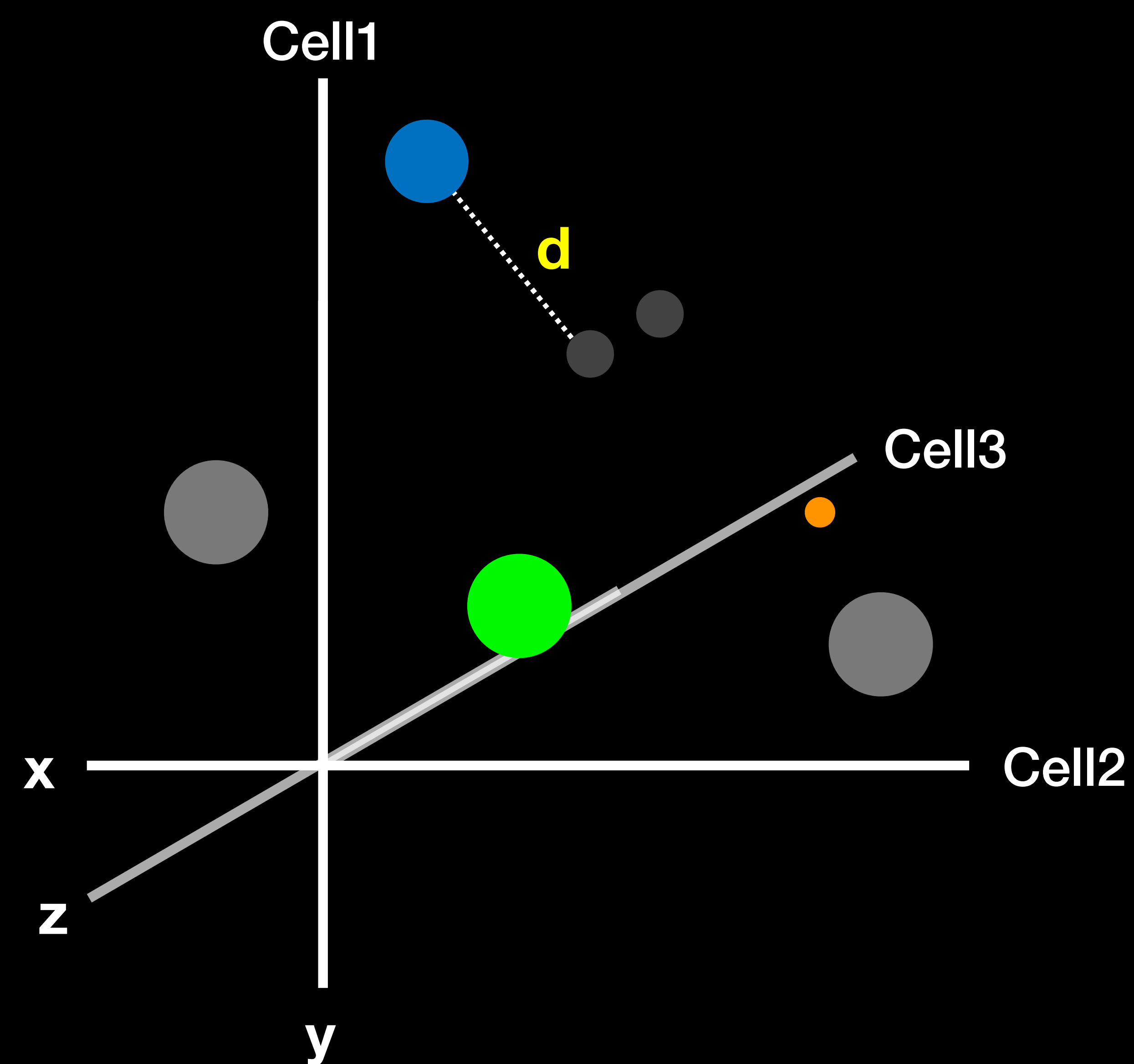


…and go through exactly the same procedure with initial cluster assignment followed by distance calculation etc…

d = sqrt(x^2 + y^2 + z^2)

Cell Samples

	#1	#2	#3
Gene 1	12	6	-13
Gene 2	-7	13	10
Gene 3	8	6	-9
Gene 4	9	5	-11
Gene 5	-3	1	6
Gene 6	10	4	-8
	x	y	z



…and go through exactly the same procedure with initial cluster assignment followed by distance calculation etc…

$d = \mathrm{sqrt}(x^2 + y^2 + z^2)$

Cell Samples

	#1	#2	#3
Gene 1	12	6	-13
Gene 2	-7	13	10
Gene 3	8	6	-9
Gene 4	9	5	-11
Gene 5	-3	1	6
Gene 6	10	4	-8

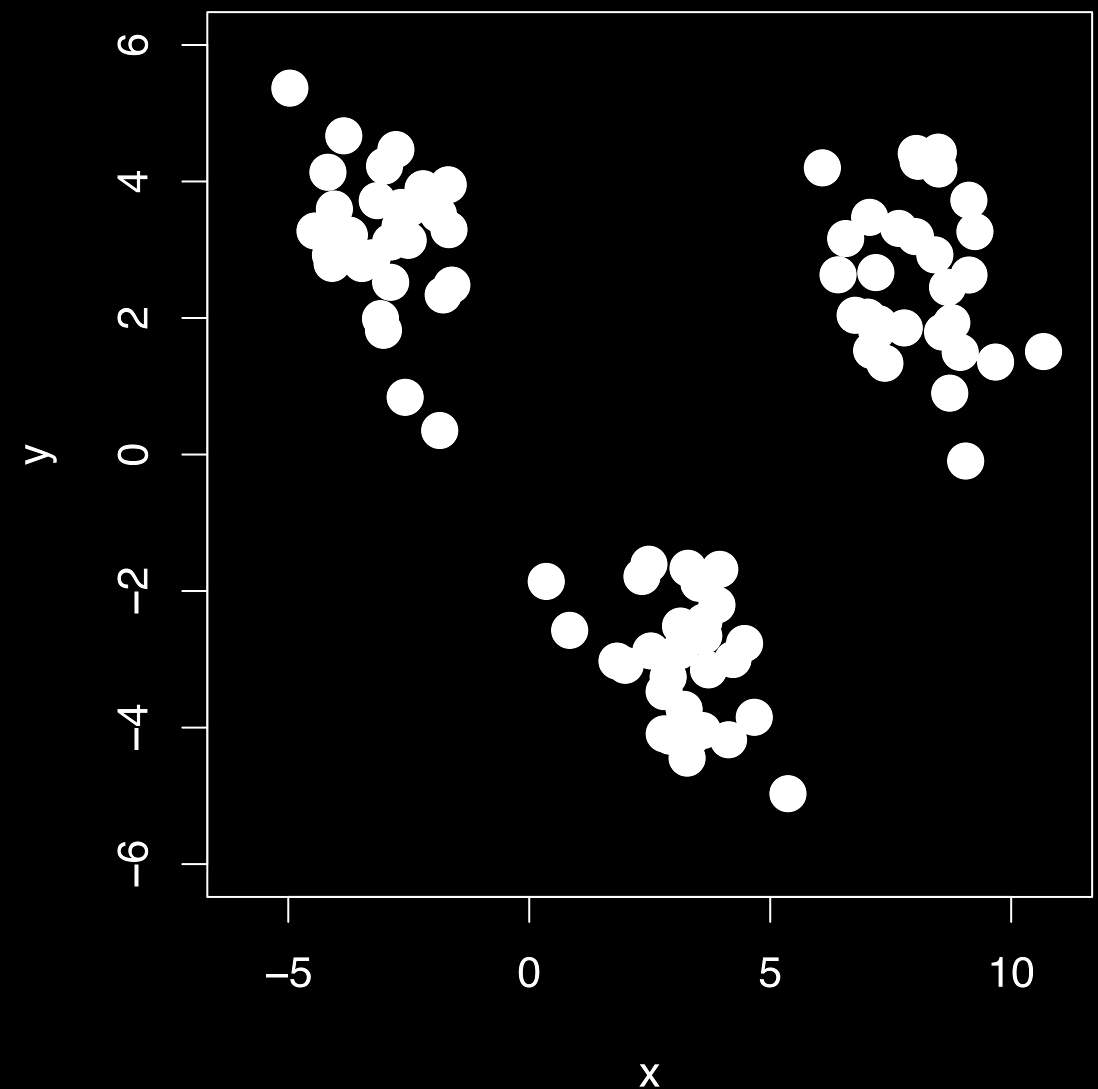
Of course we don't actually need to plot anything.

We can just calculate the Euclidean distance along any number of dimensions and perform our k-means clustering in the same way.

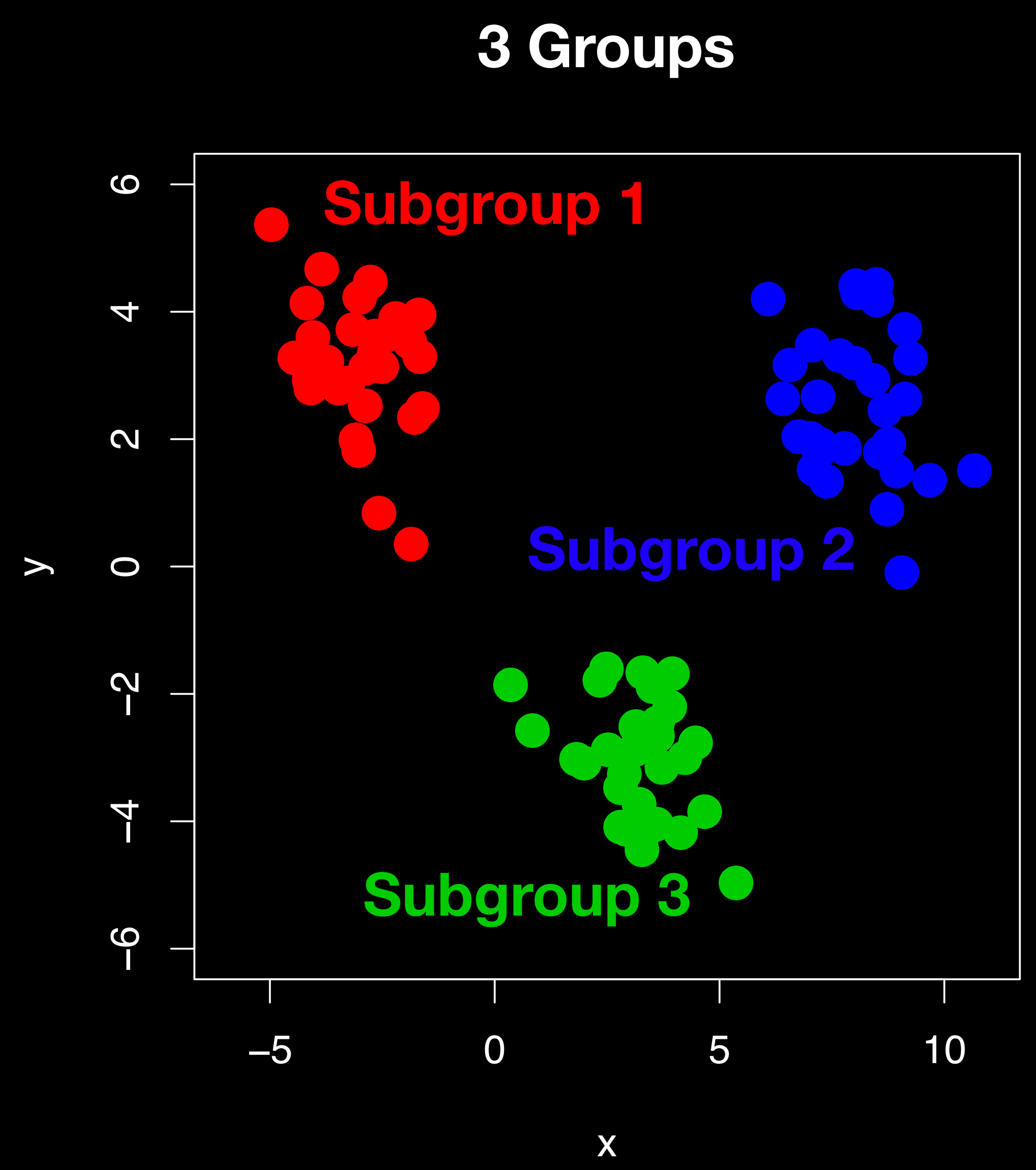
k-means in R

```r
# k-means algorithm with 3 centers, run 20 times
kmeans(x, centers= 3, nstart= 20)
```

- Input **x** is a numeric matrix, or data.frame, with one observation per row, one feature per column
- k-means has a random component
- Run algorithm multiple times to improve odds of the best model



y
x
−5
0
5
10
−6
−4
−2
0
2
4
6


3 Groups
Subgroup 1
Subgroup 2
Subgroup 3
6
4
2
0
−2
−4
−6
y
−5
0
5
10
x

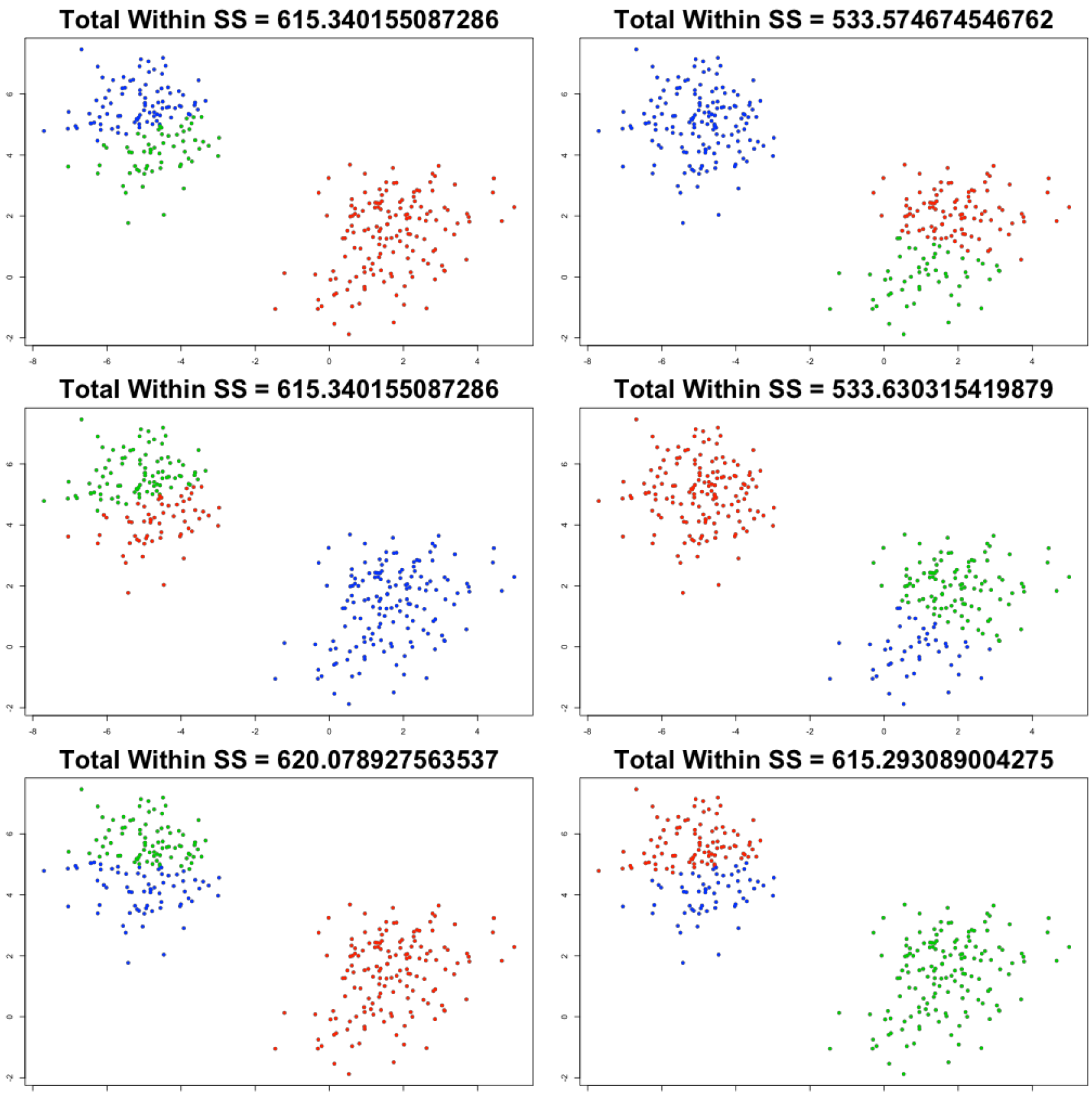
Model selection

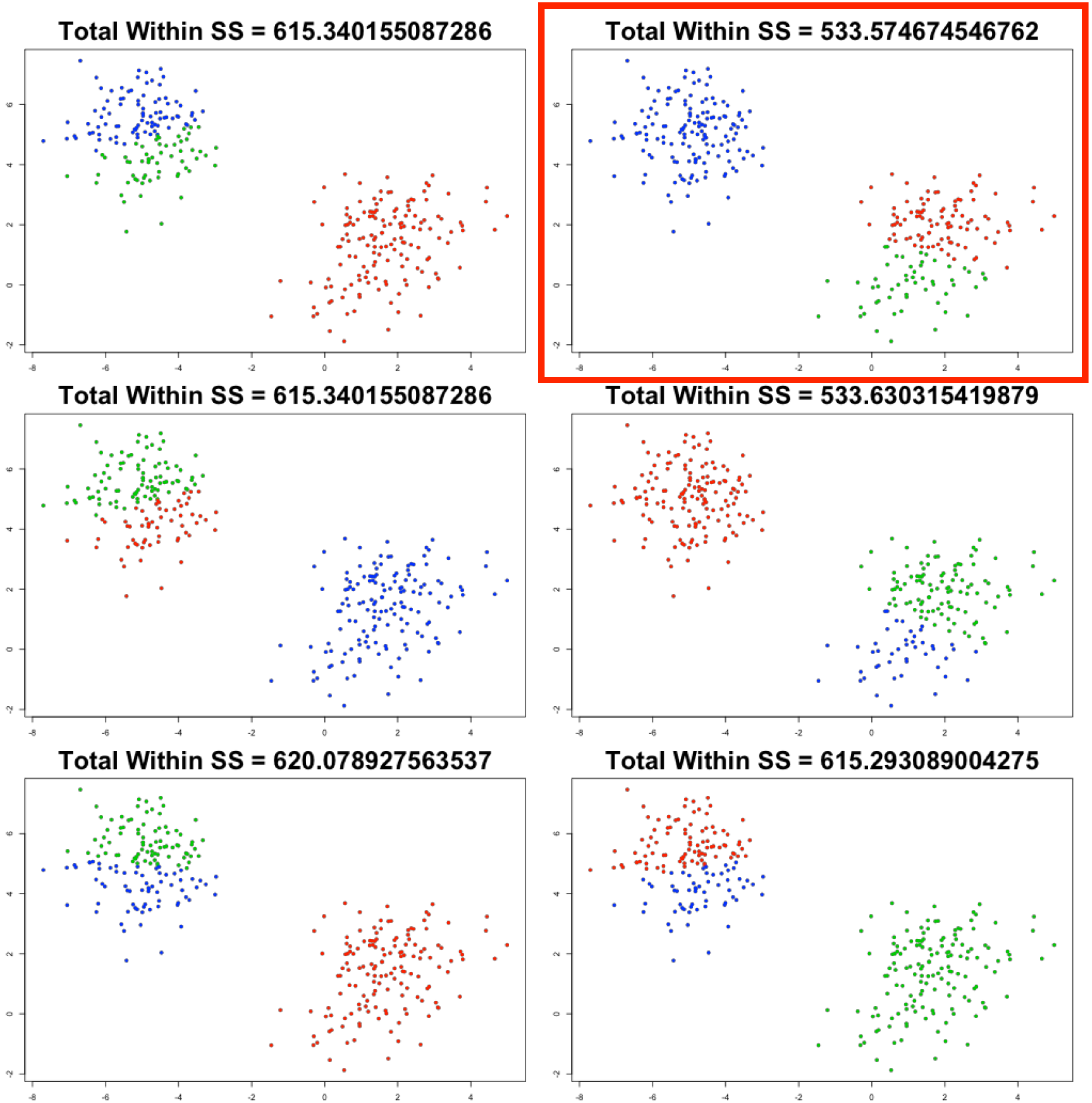
- Recall k-means has a random component
- Best outcome is based on total within cluster sum of squares:
  - ➡ For each cluster
    - For each observation in the cluster
      - Determine squared distance from observation to cluster center
  - ➡ Sum all of them together

Model selection

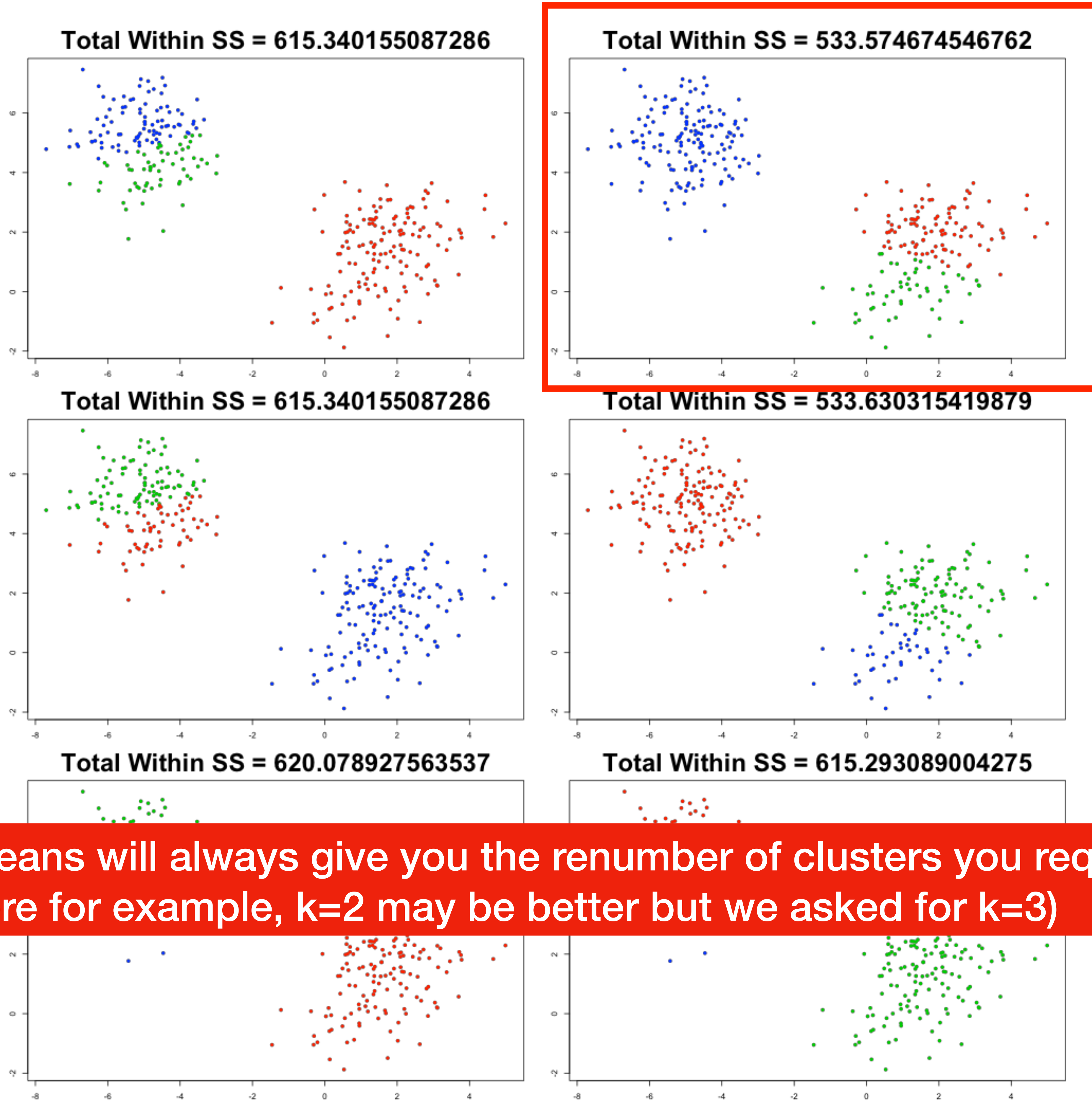
```
# k-means algorithm with 5 centers, run 20 times
kmeans(x, centers=5, nstart=20)
```

- Running algorithm multiple times (i.e. setting `nstart`) helps find the global minimum total within cluster sum of squares
- Increasing the default value of **`nstart`** is often sensible


Total Within SS = 615.340155087286
Total Within SS = 533.574674546762
Total Within SS = 615.340155087286
Total Within SS = 533.630315419879
Total Within SS = 620.078927563537
Total Within SS = 615.293089004275

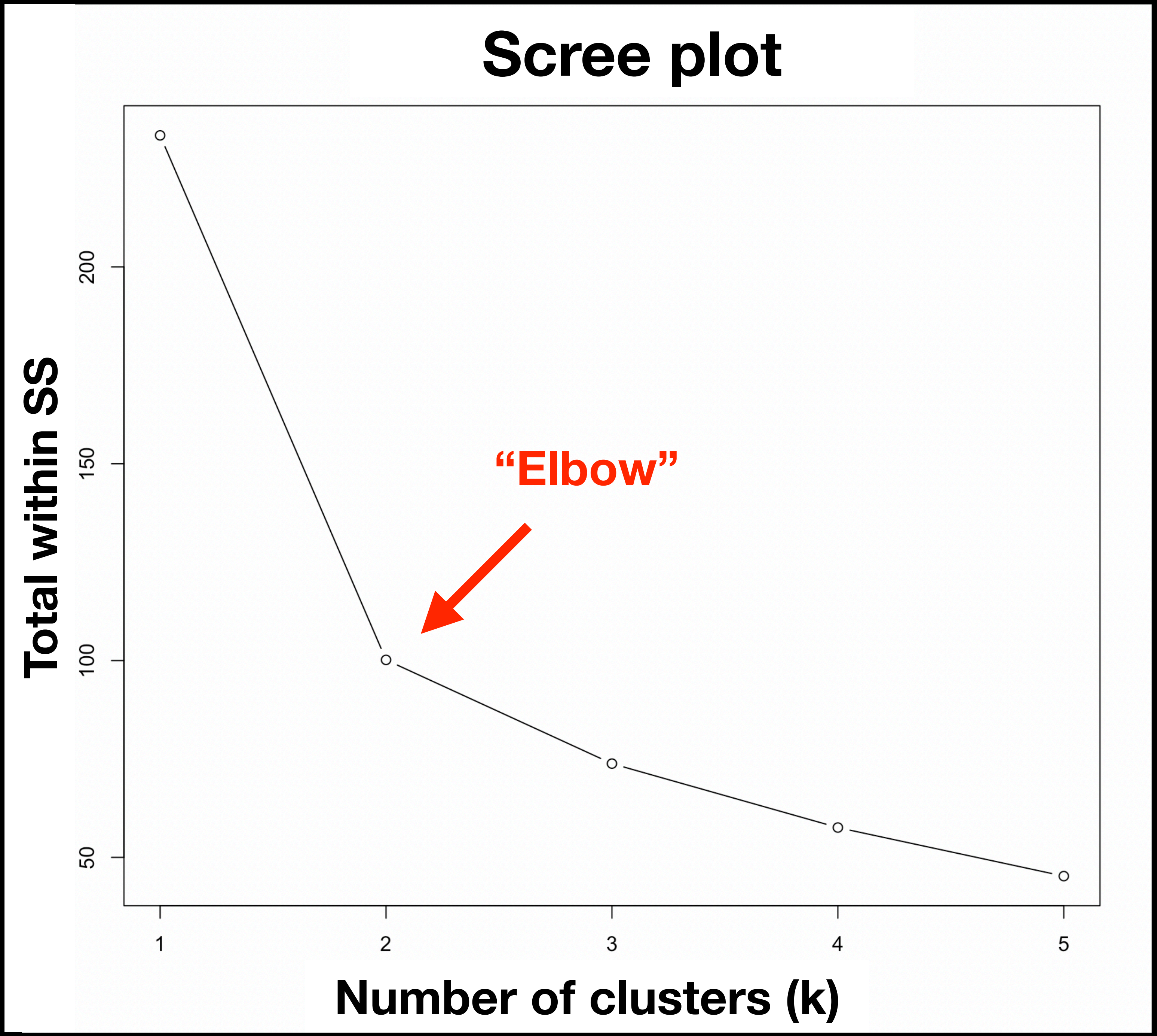


Winner has the smallest within cluster SS



Note. k-means will always give you the renumber of clusters you request!
(Here for example, k=2 may be better but we asked for k=3)

Determining number of clusters



Trial and error is not the best approach

Systematically try a range of different k values and plot a "scree plot".

Here there is a large reduction in SS with **k=2** but after that the values do not go down as quickly!

Your Turn!

```
# Generate some example data for clustering
tmp <- c(rnorm(30,-3), rnorm(30,3))
x <- cbind(x=tmp, y=rev(tmp))

plot(x)
```

```
Use the kmeans() function setting k to 2 and nstart=20

Inspect/print the results

Q. How many points are in each cluster?
Q. What ‘component’ of your result object details
      - cluster size?
      - cluster assignment/membership?
      - cluster center?

Plot x colored by the kmeans cluster assignment and
      add cluster centers as blue points
```

Today’s Menu

- Introduction to machine learning
  - Unsupervised, supervised and reinforcement learning

- Clustering
  - K-means clustering
  - Hierarchical clustering
  - Heatmap representations

- Dimensionality reduction, visualization and ‘structure’ analysis
  - Principal Component Analysis (PCA)

- Hands-on application to cell classification

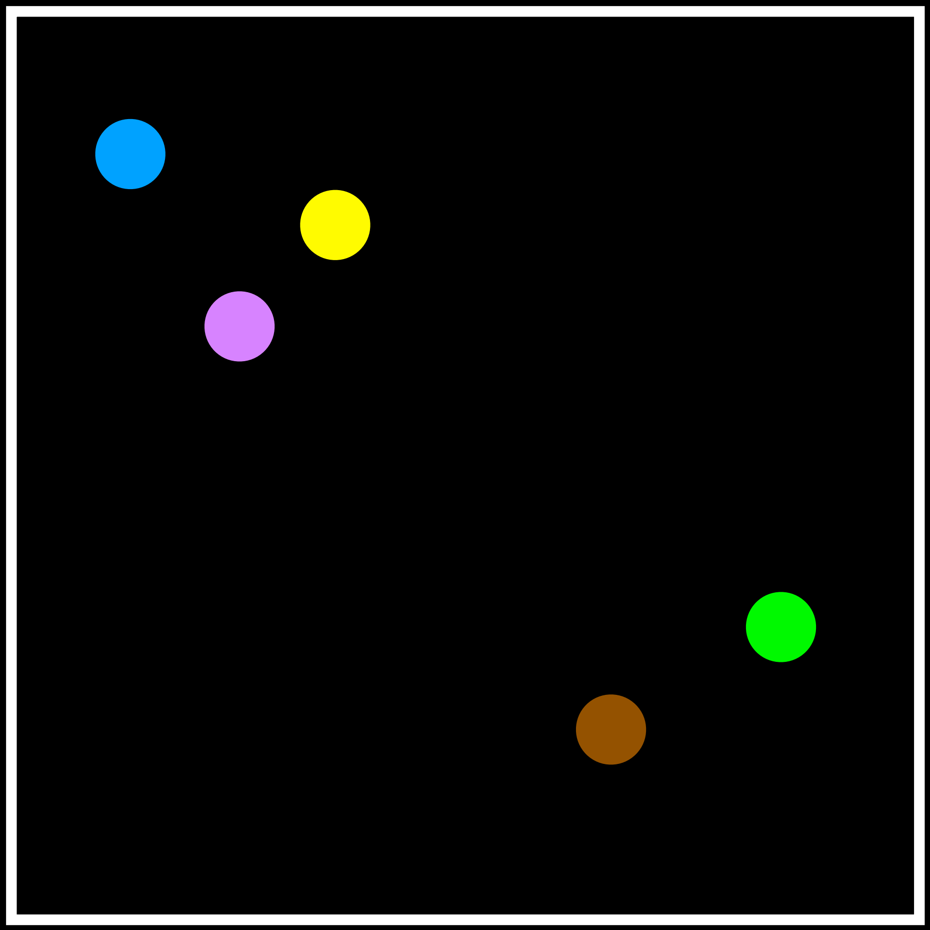
Hierarchical clustering

- Number of clusters is not known ahead of time
- Two kinds of hierarchical clustering:
  - ➡ bottom-up
  - ➡ top-down

Hierarchical clustering

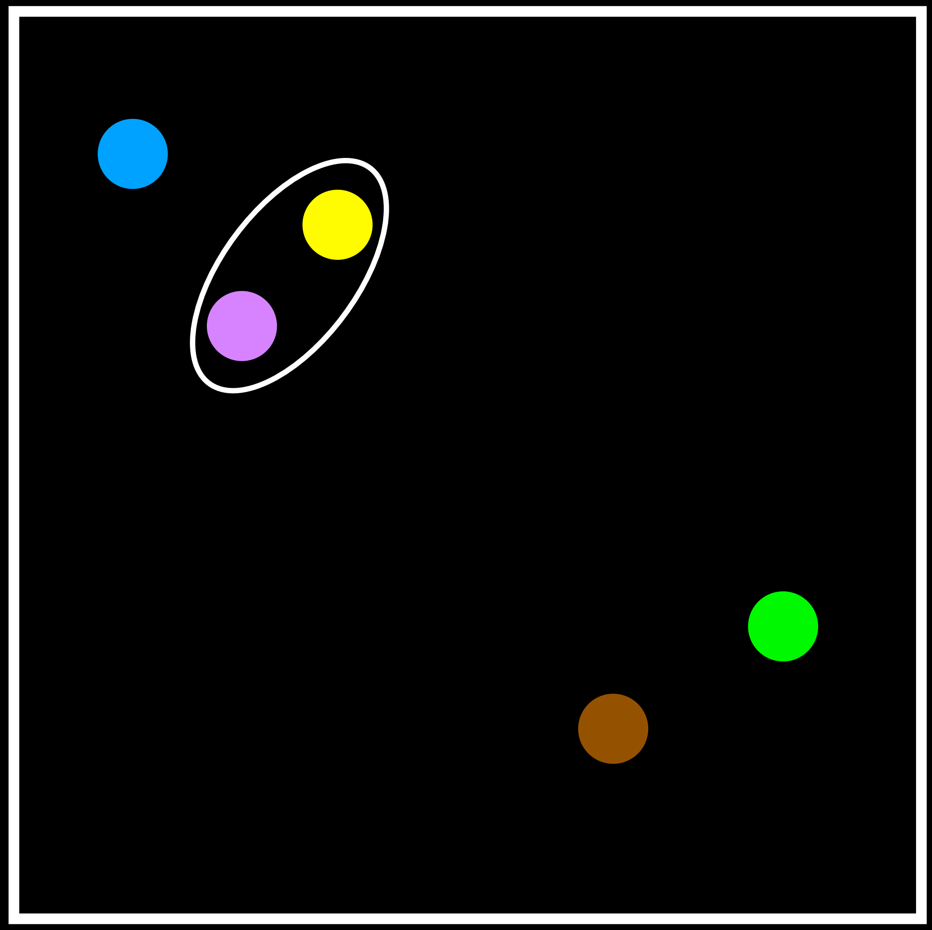
Simple example:

5 clusters: Each point starts as it’s own “cluster”!



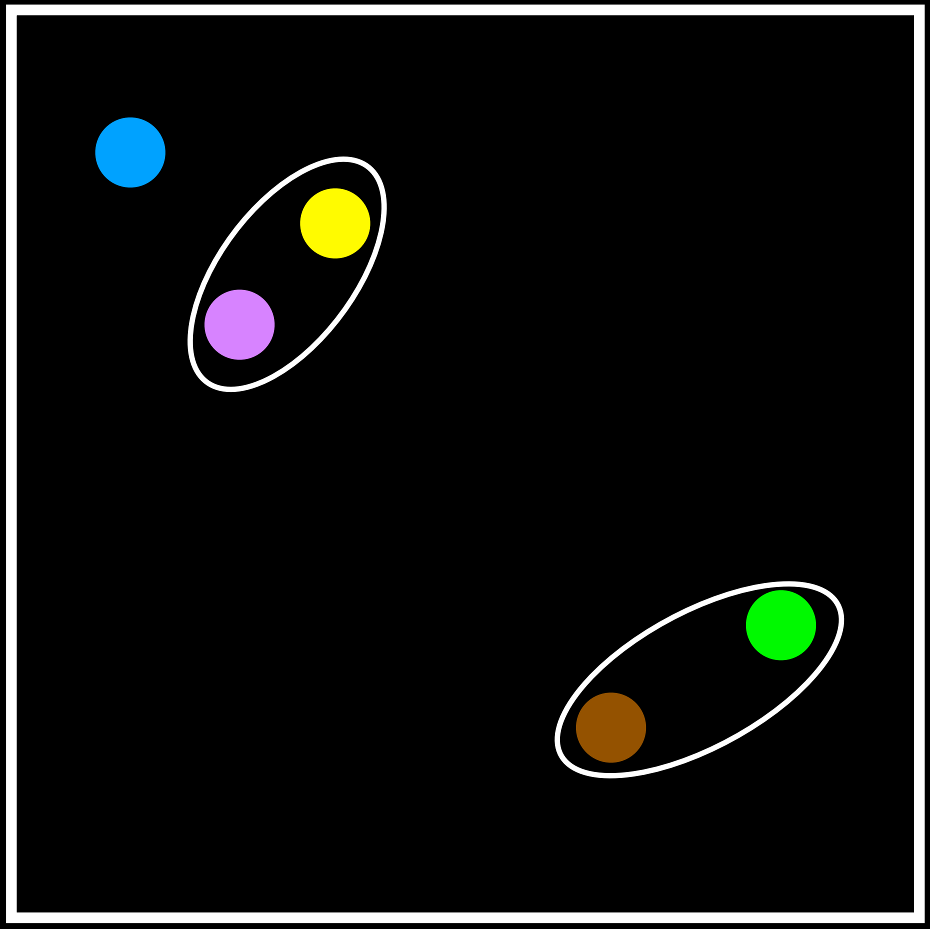
Hierarchical clustering

4 clusters



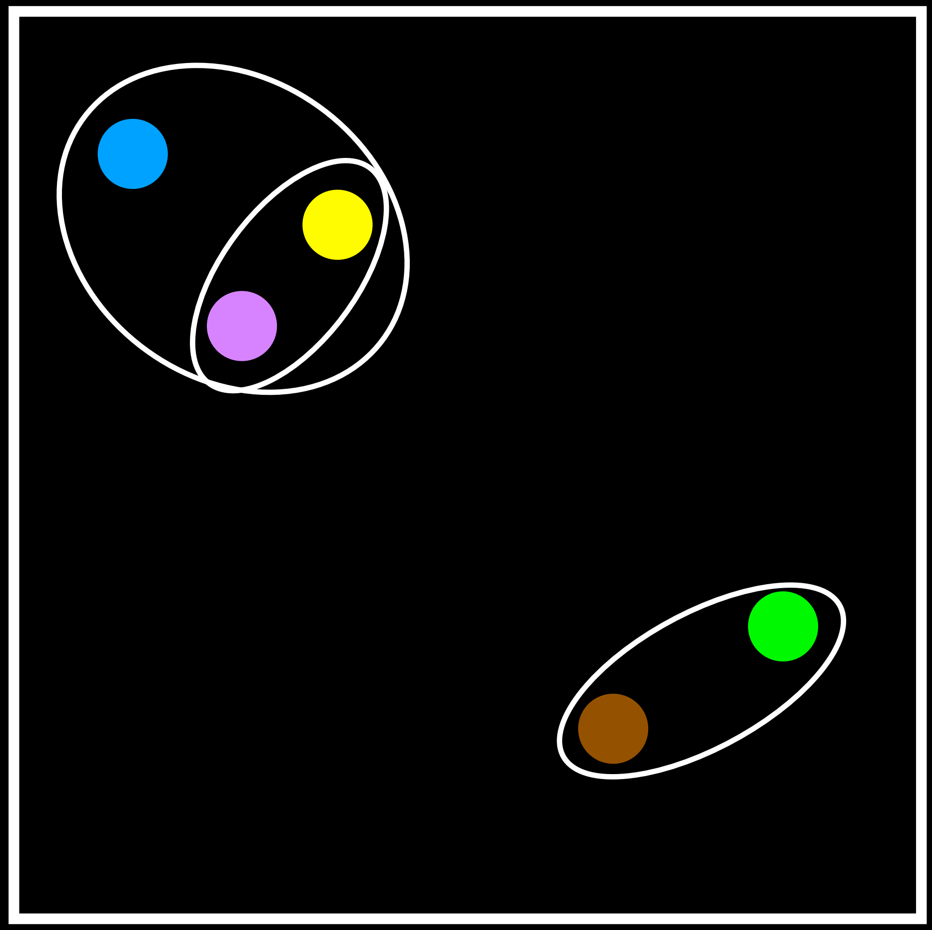
Hierarchical clustering

3 clusters



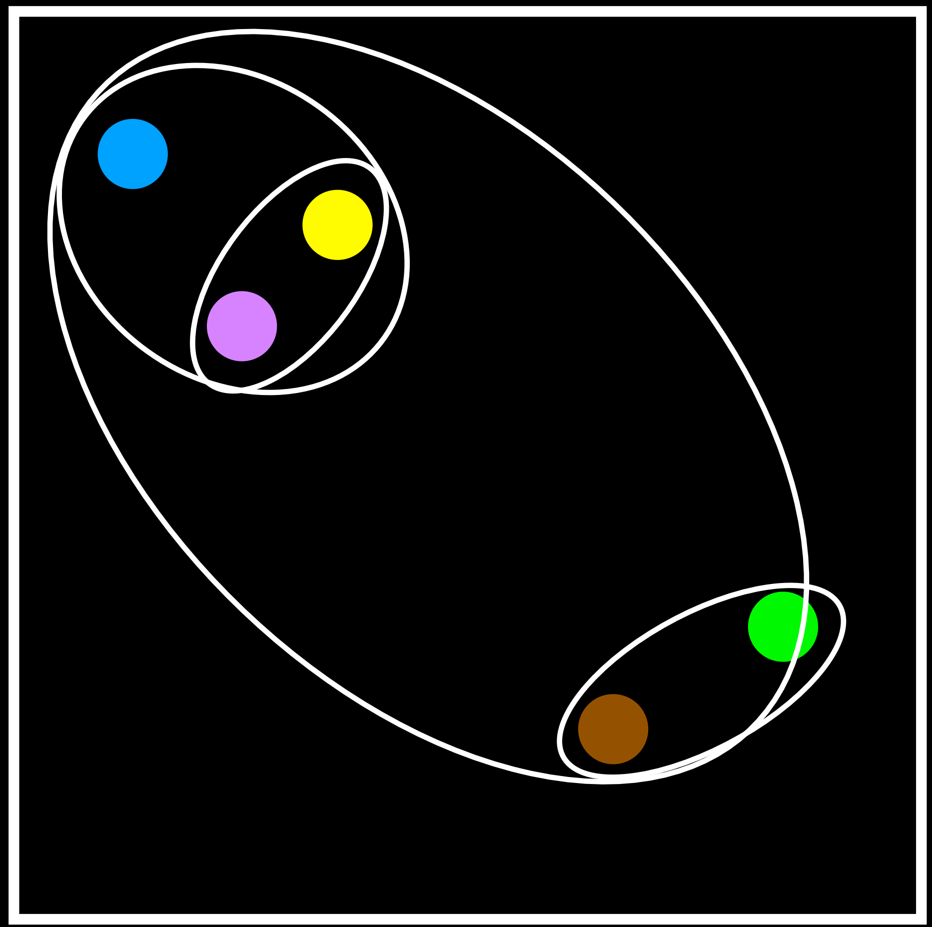
Hierarchical clustering

2 clusters



Hierarchical clustering

End: 1 cluster



Hierarchical clustering in R

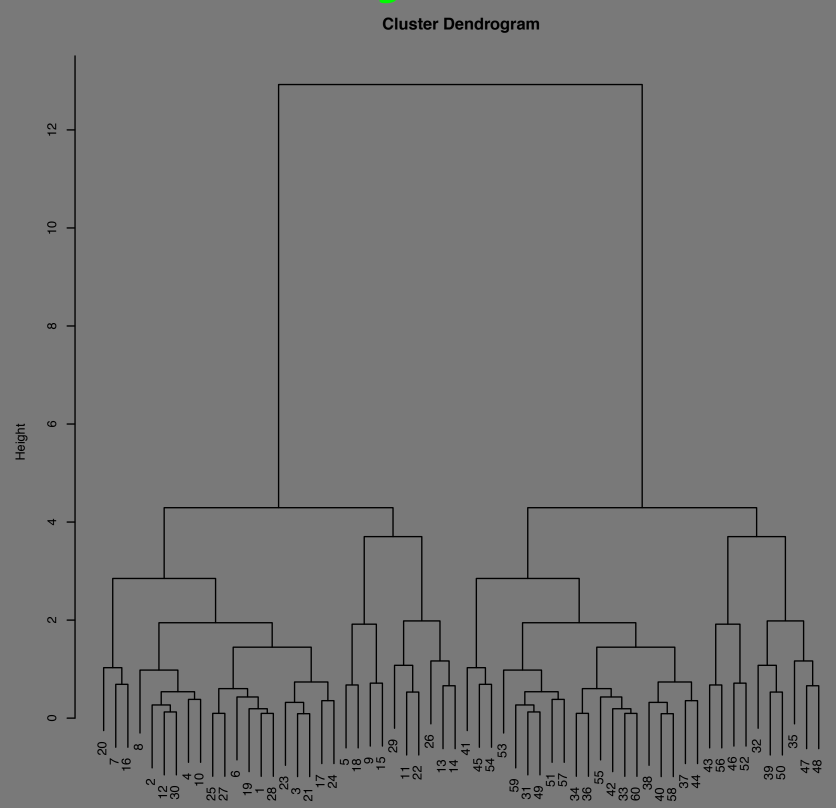
```
# First we need to calculate point (dis)similarity
#   as the Euclidean distance between observations
dist_matrix <- dist(x)

# The hclust() function returns a hierarchical
#  clustering model
hc <- hclust(d = dist_matrix)

# the print method is not so useful here
hc

Call:
hclust(d = dist_matrix)

Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

Lets have a closer look...

```
# Our input is a distance matrix from the dist()
# function. Lets make sure we understand it first
dist_matrix <- dist(x)

dim(dist_matrix)
NULL

View( as.matrix(dist_matrix) )

dim(x)
[1] 60  2

dim( as.matrix(dist_matrix) )
[1] 60 60

# Note. symmetrical pairwise distance matrix
```

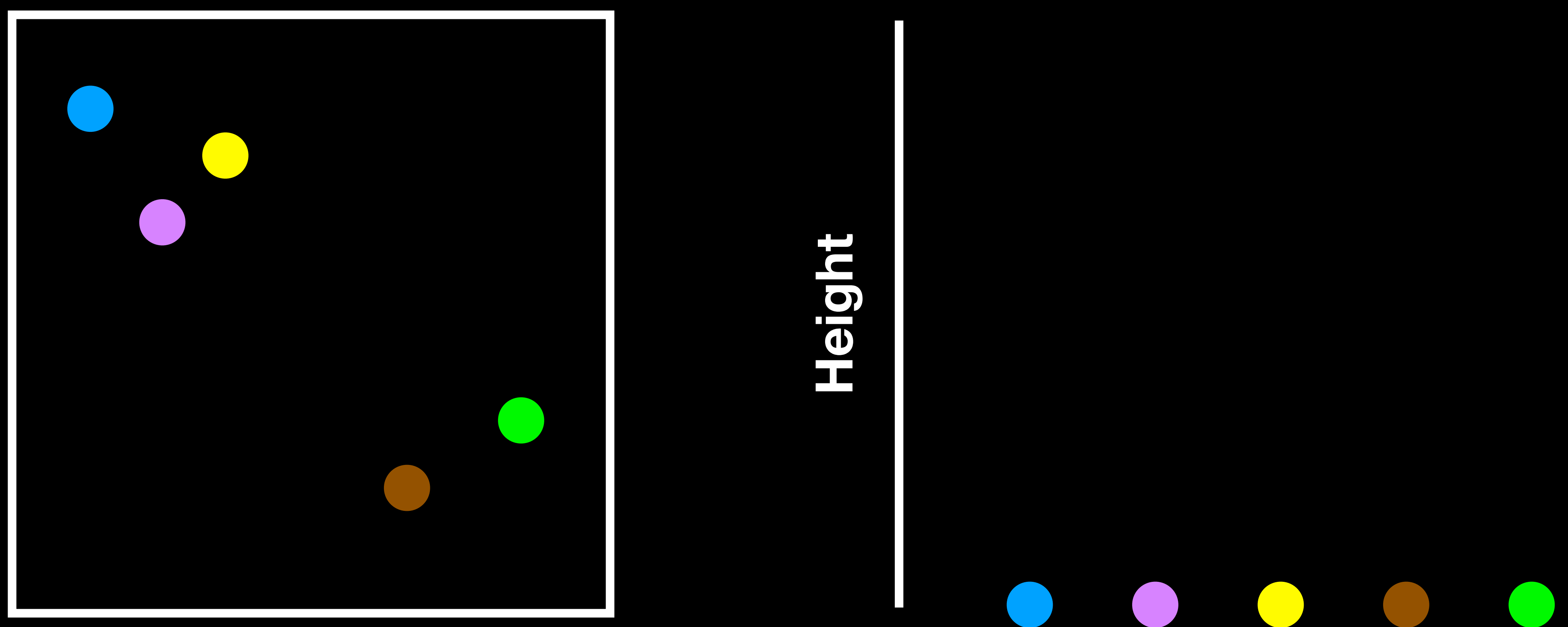
Interpreting results

```
# Create hierarchical cluster model: hc
hc <- hclust(dist(x))


# We can plot the results as a dendrogram
plot(hc)

# What do you notice?
# Does the dendrogram
# make sense based on
# your knoweledge of x?
```



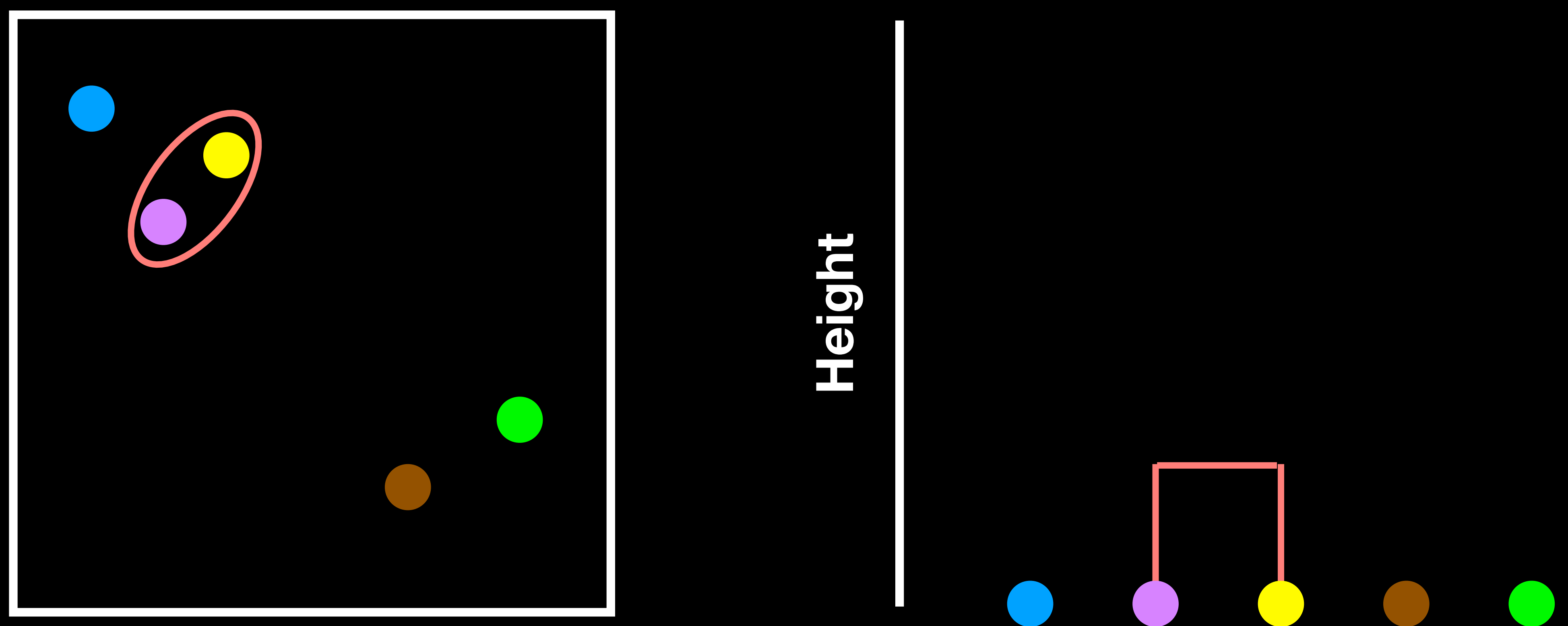
Dendrogram

- Tree shaped structure used to interpret hierarchical clustering models



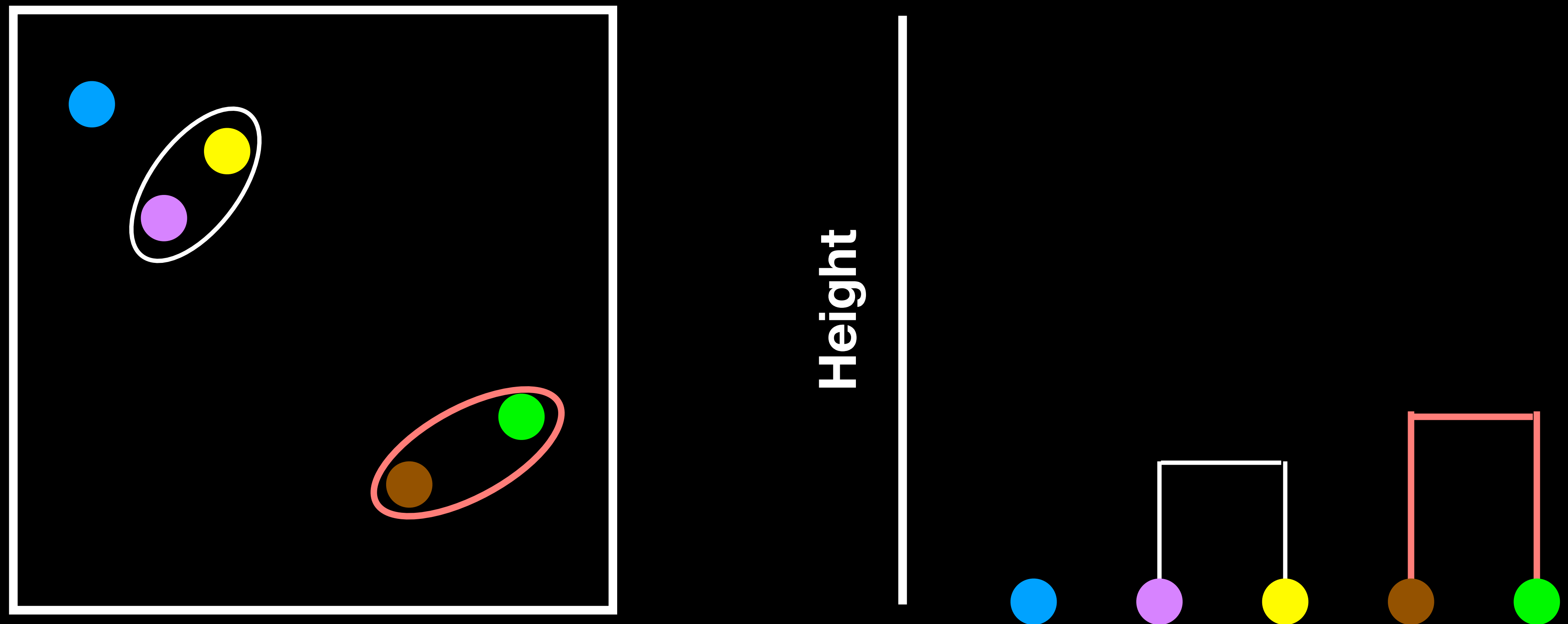
Dendrogram

- Tree shaped structure used to interpret hierarchical clustering models



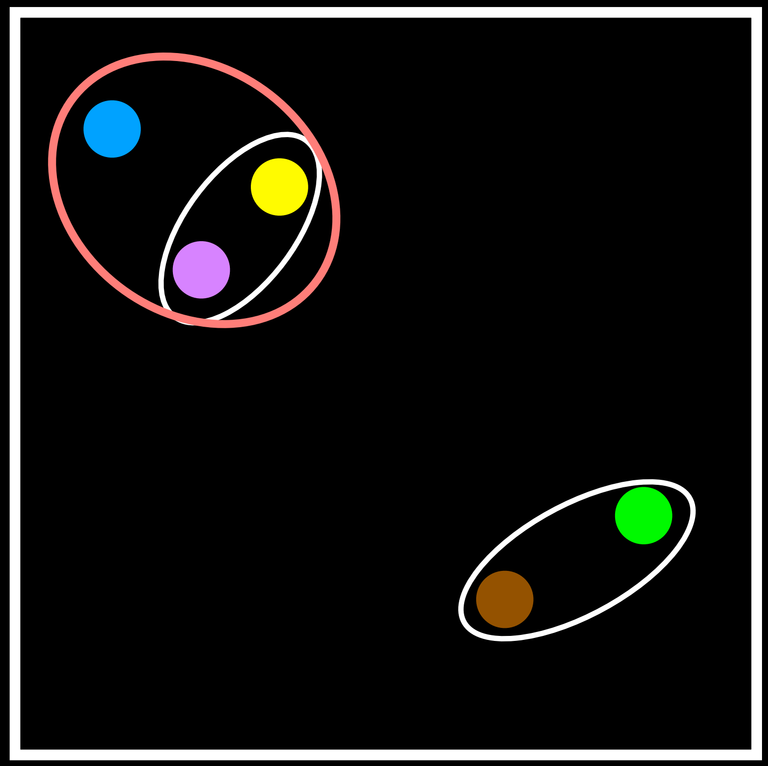
Dendrogram

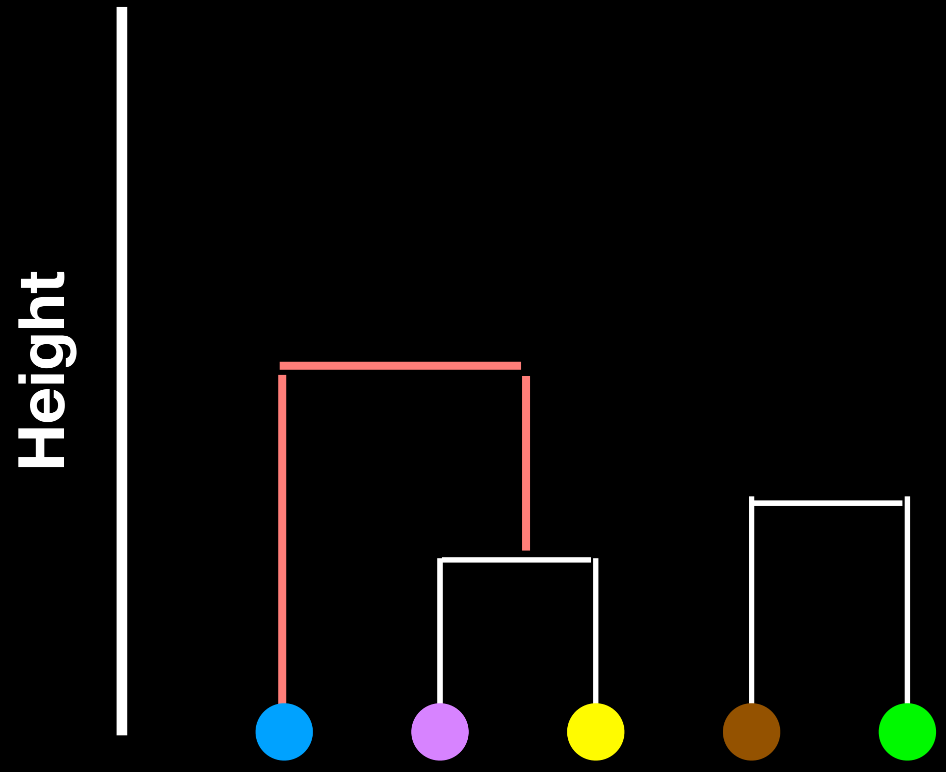
- Tree shaped structure used to interpret hierarchical clustering models



Dendrogram

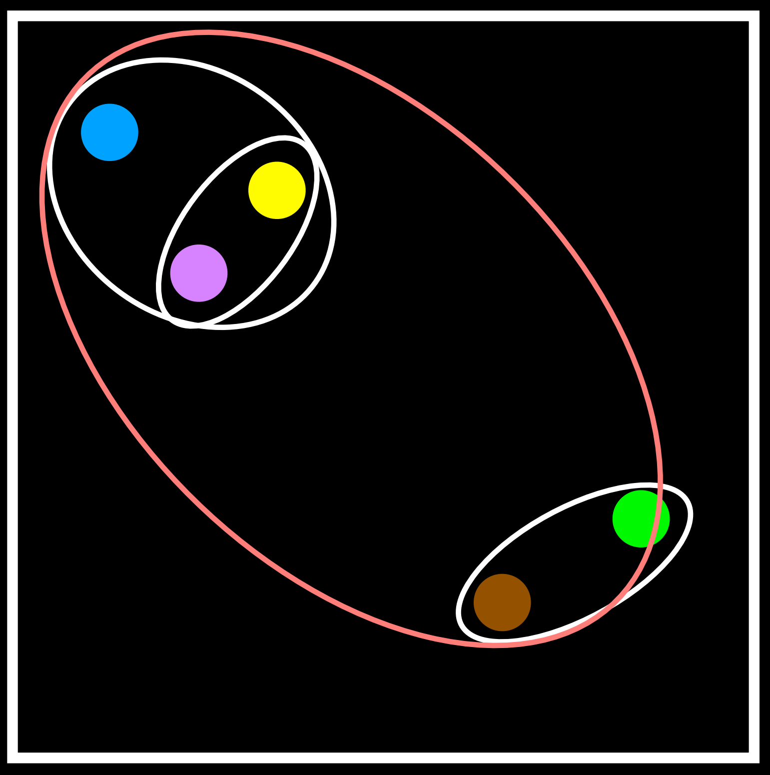
- Tree shaped structure used to interpret hierarchical clustering models

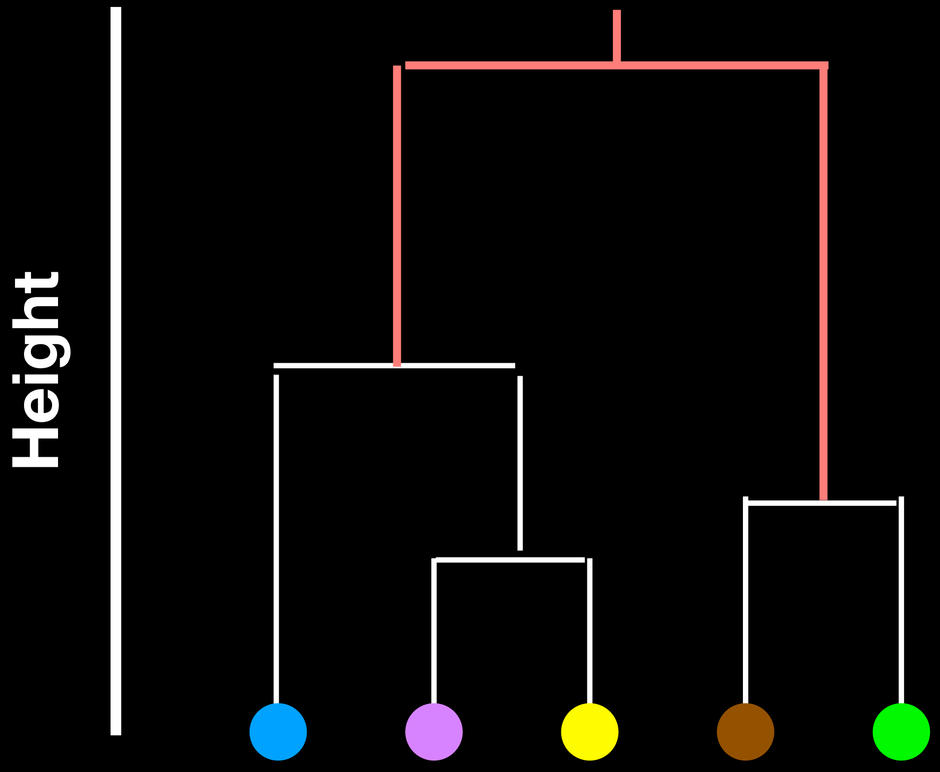




Dendrogram

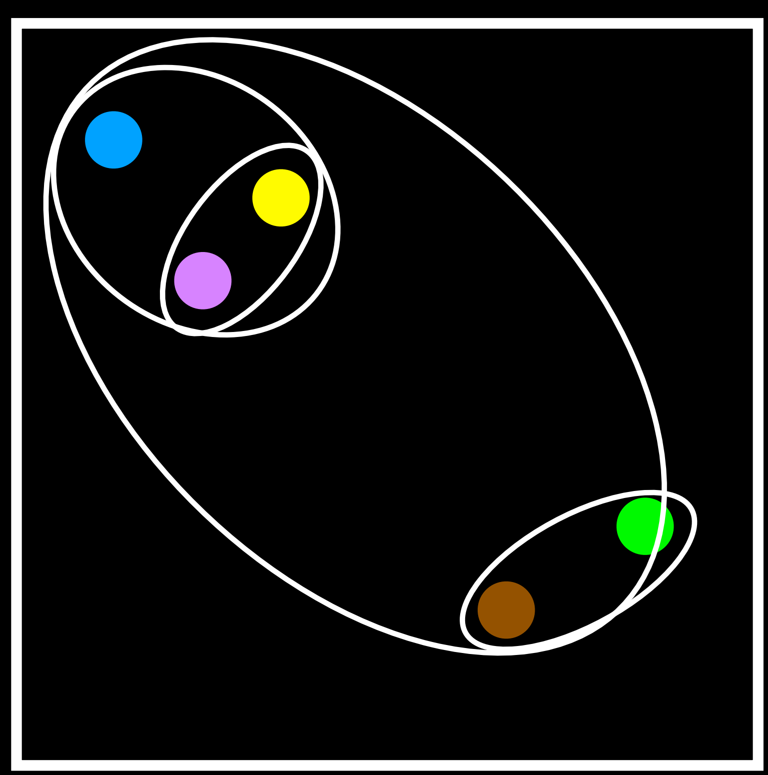
- Tree shaped structure used to interpret hierarchical clustering models

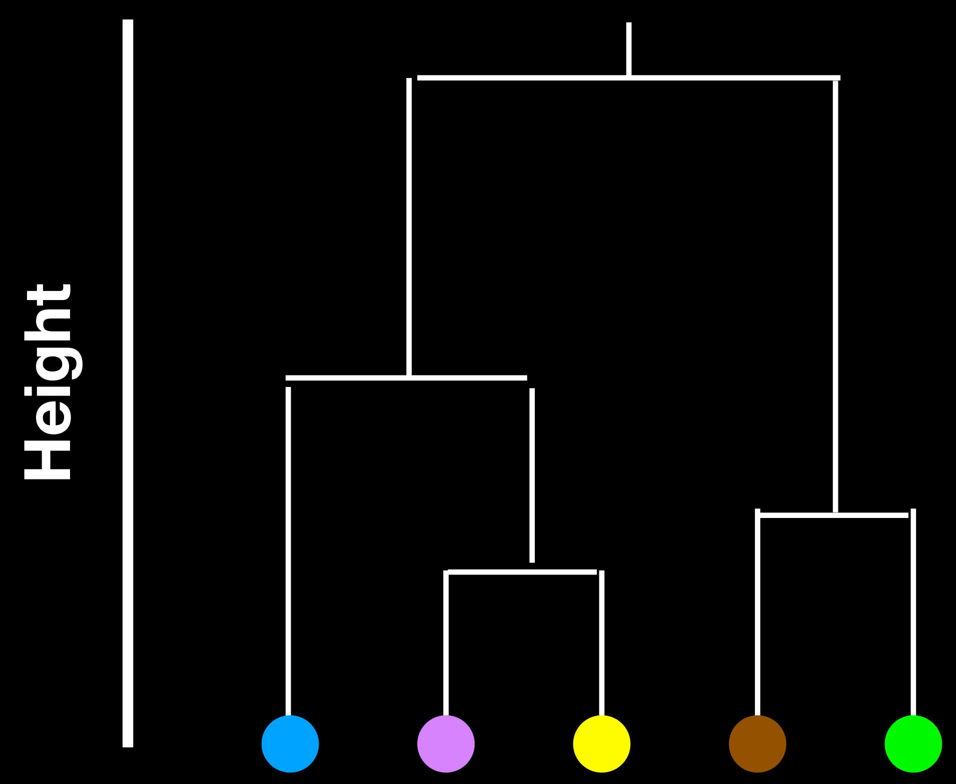




Dendrogram

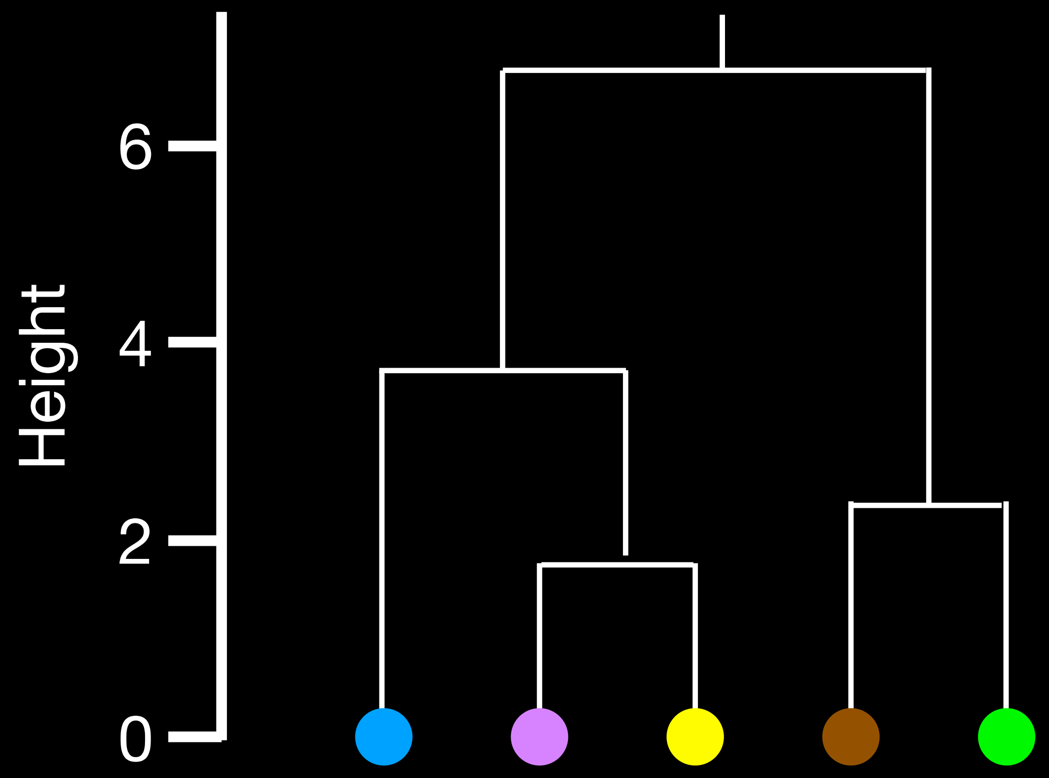
- Tree shaped structure used to interpret hierarchical clustering models





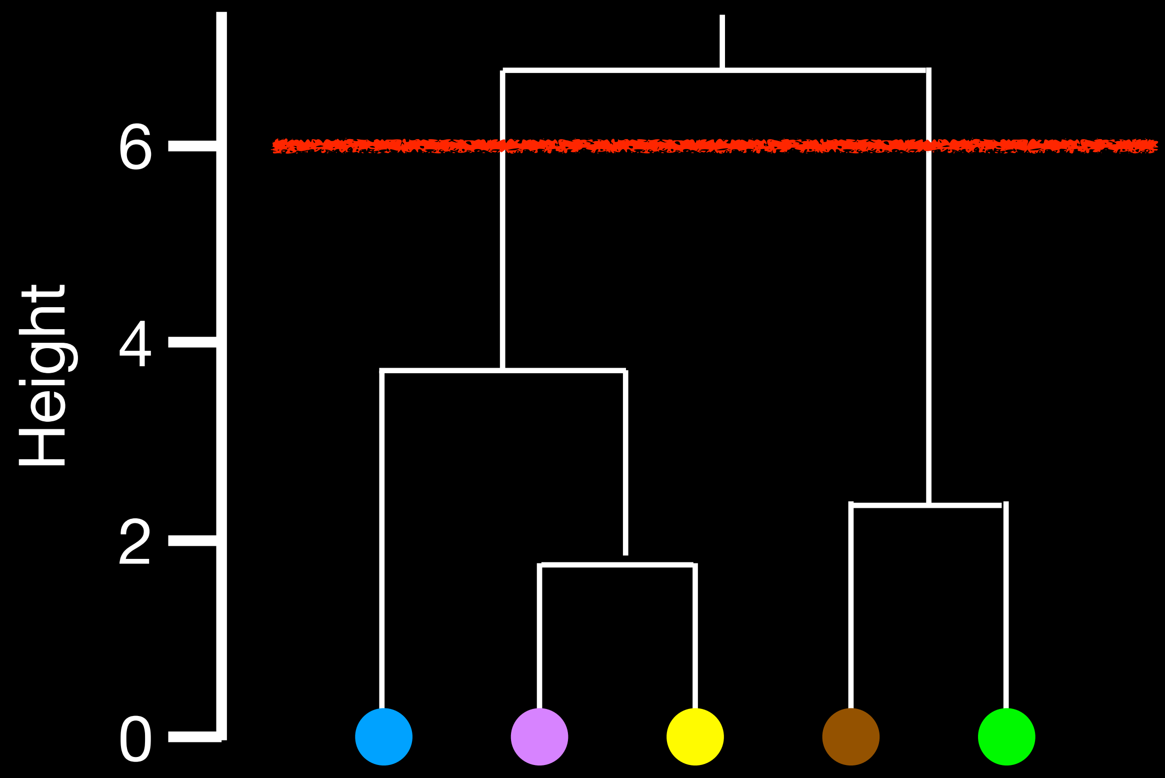
Dendrogram plotting in R

```
# Draws a dendrogram
plot(hc)
```



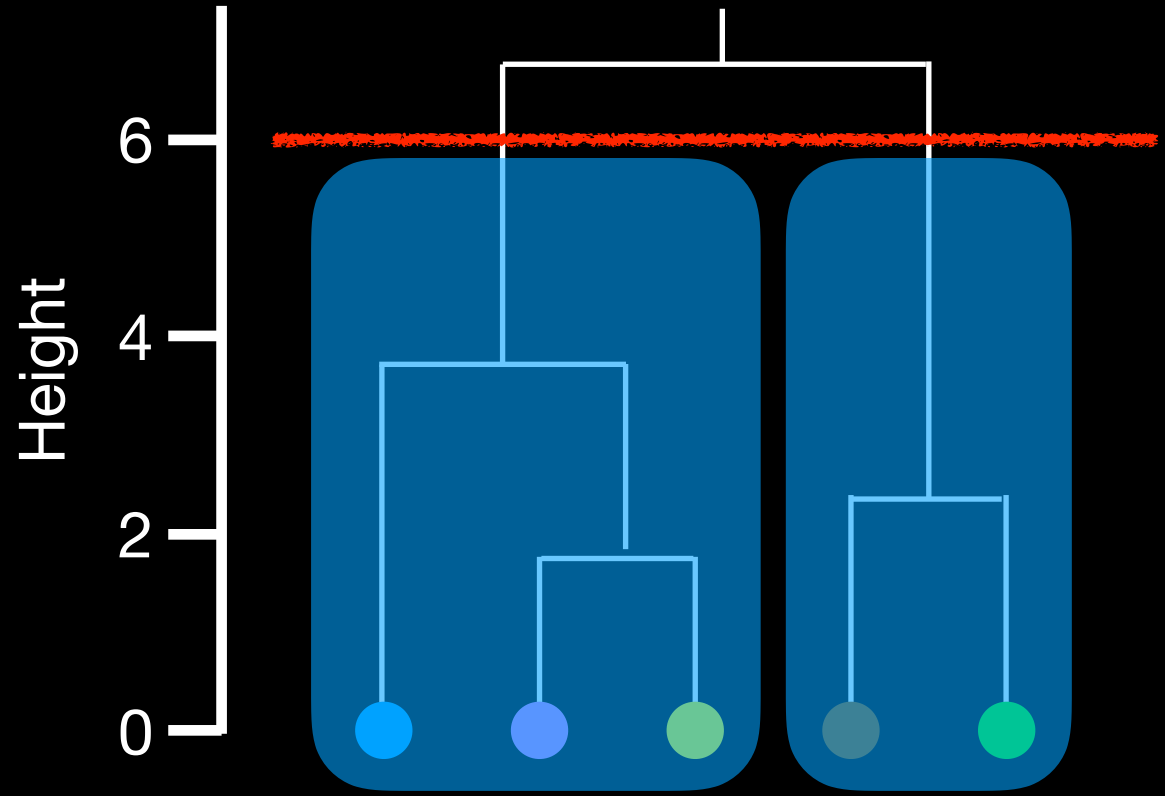
Dendrogram plotting in R

```
# Draws a dendrogram
plot(hc)
abline(h=6, col="red")
```



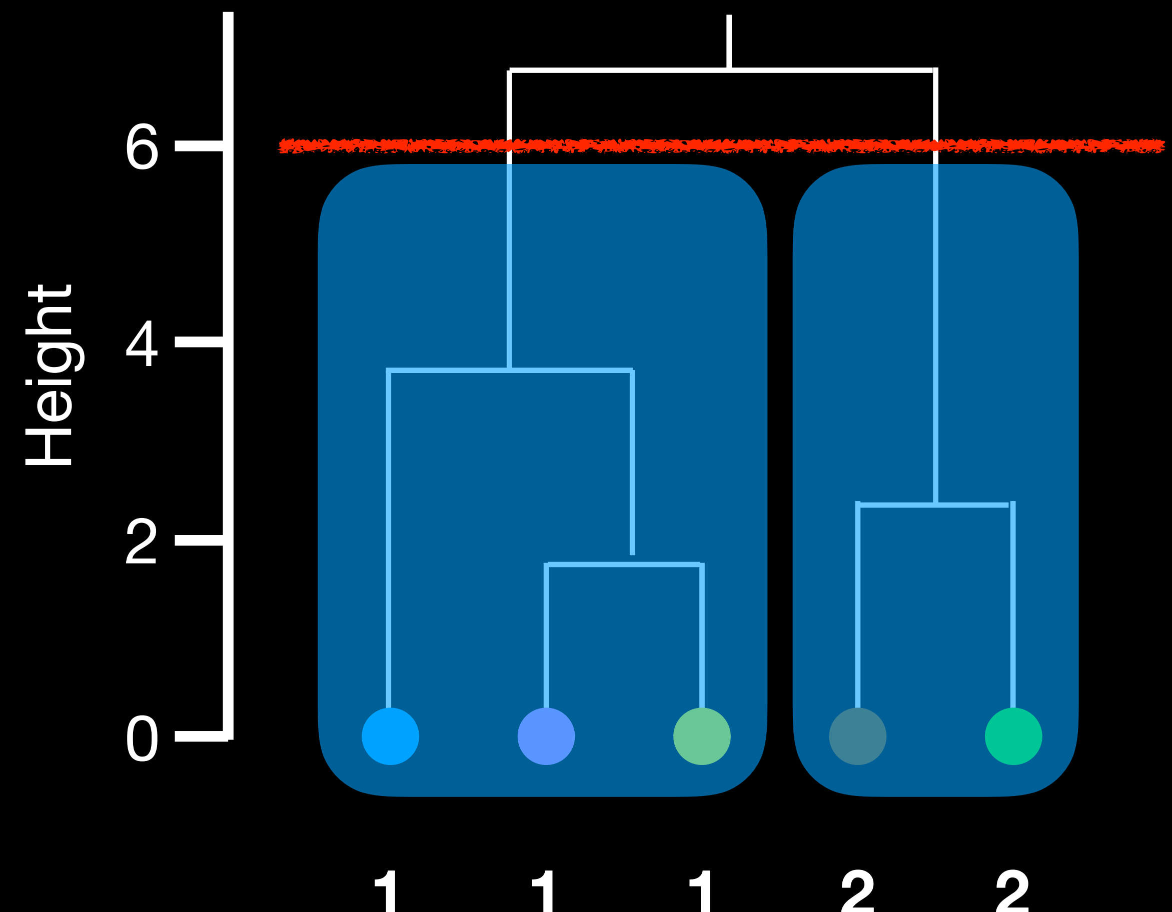
Dendrogram plotting in R

```
# Draws a dendrogram
plot(hc)
abline(h=6, col="red")
```



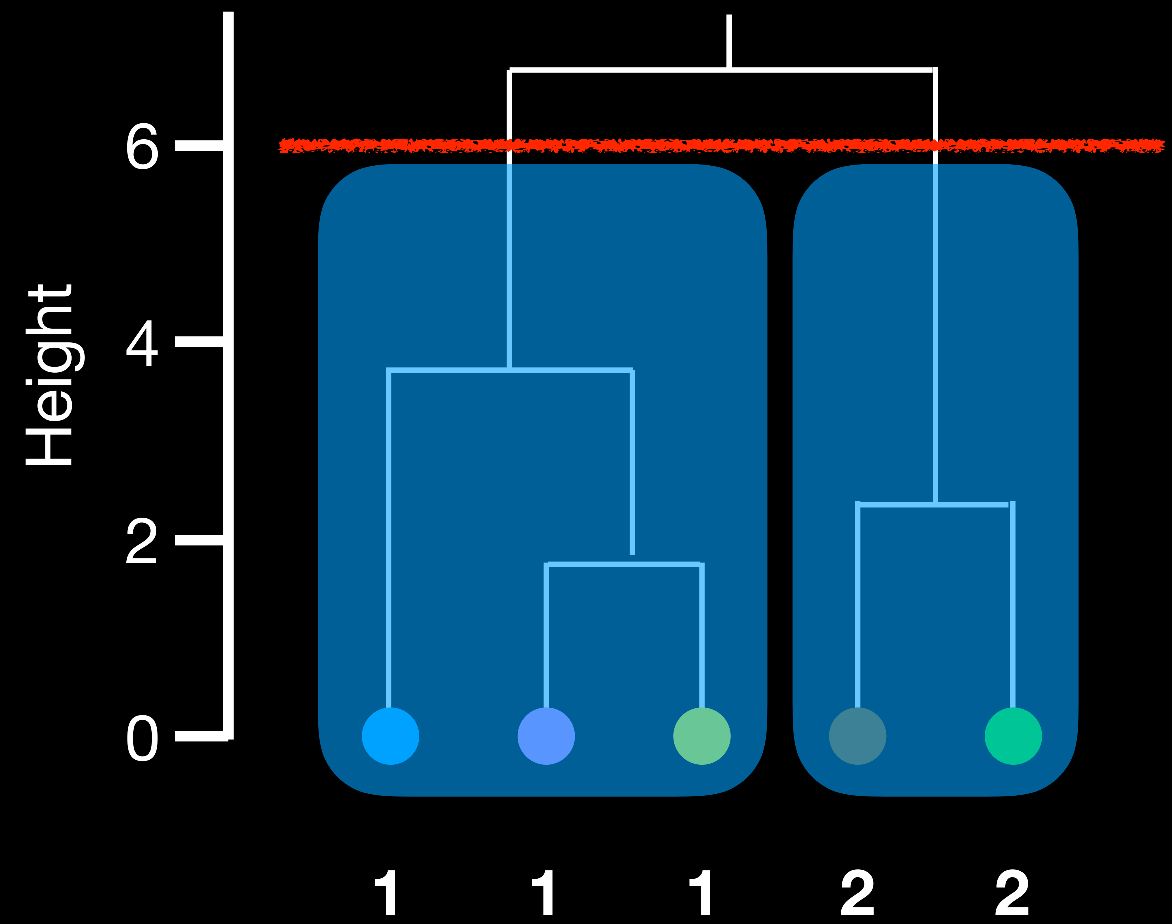
Dendrogram plotting in R

```
# Draws a dendrogram
plot(hc)
abline(h=6, col="red")
cutree(hc, h=6) # Cut by height h
[1] 1,1,1,2,2
```



Dendrogram plotting in R

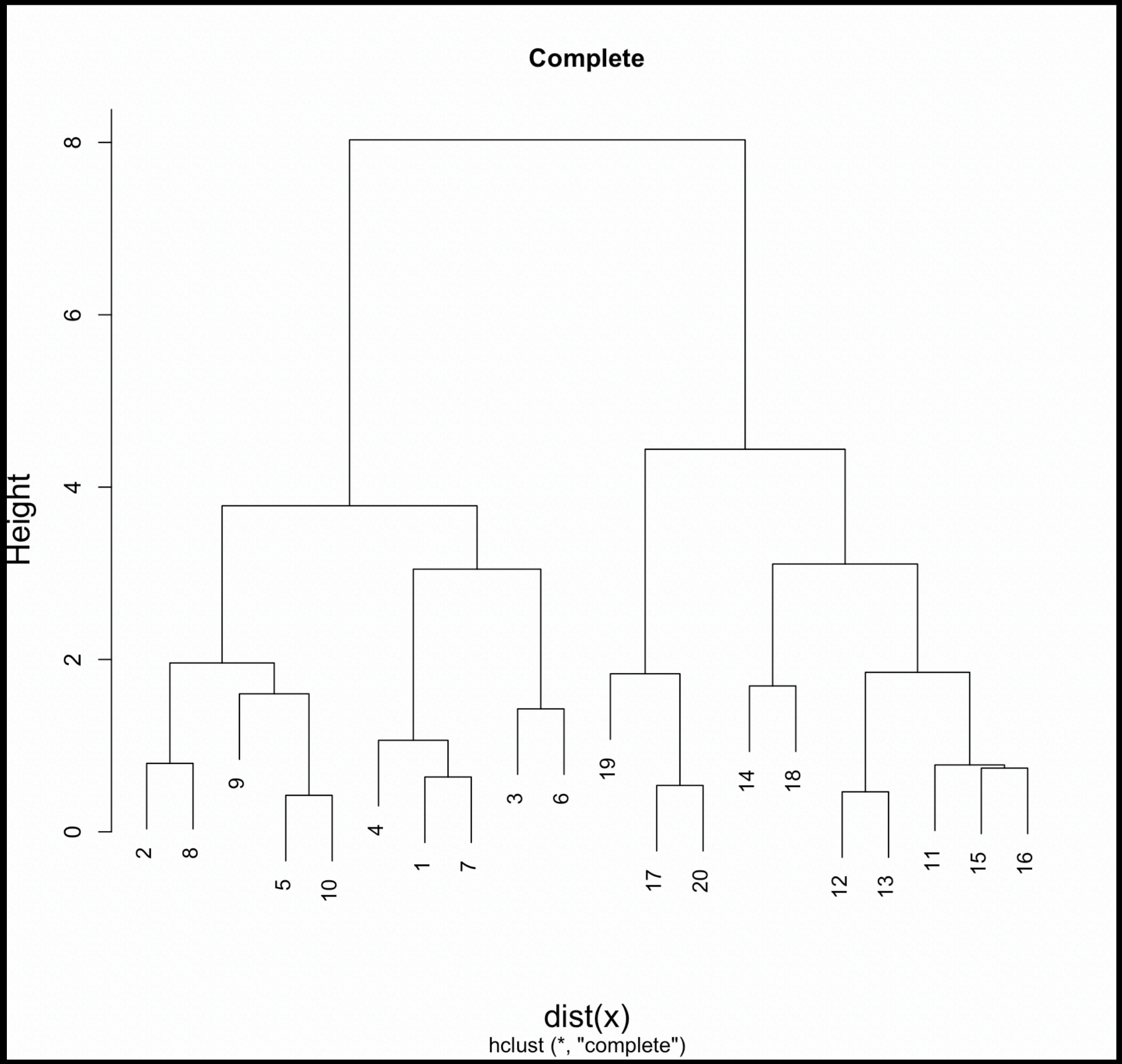
```
# Draws a dendrogram
plot(hc)
abline(h=6, col="red")
cutree(hc, k=2 ) # Cut into k grps
[1] 1,1,1,2,2
```

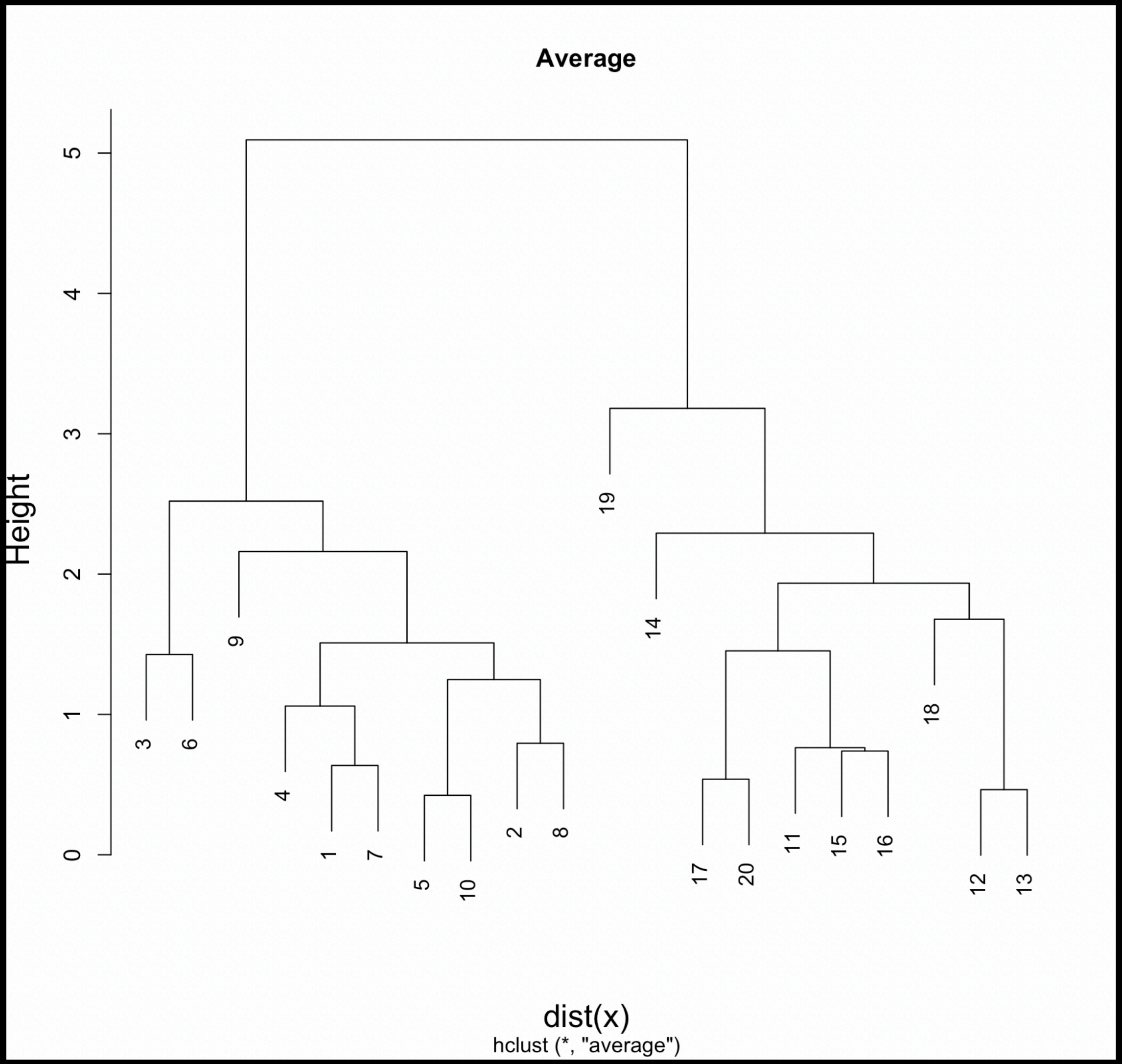


Linking clusters in hierarchical clustering

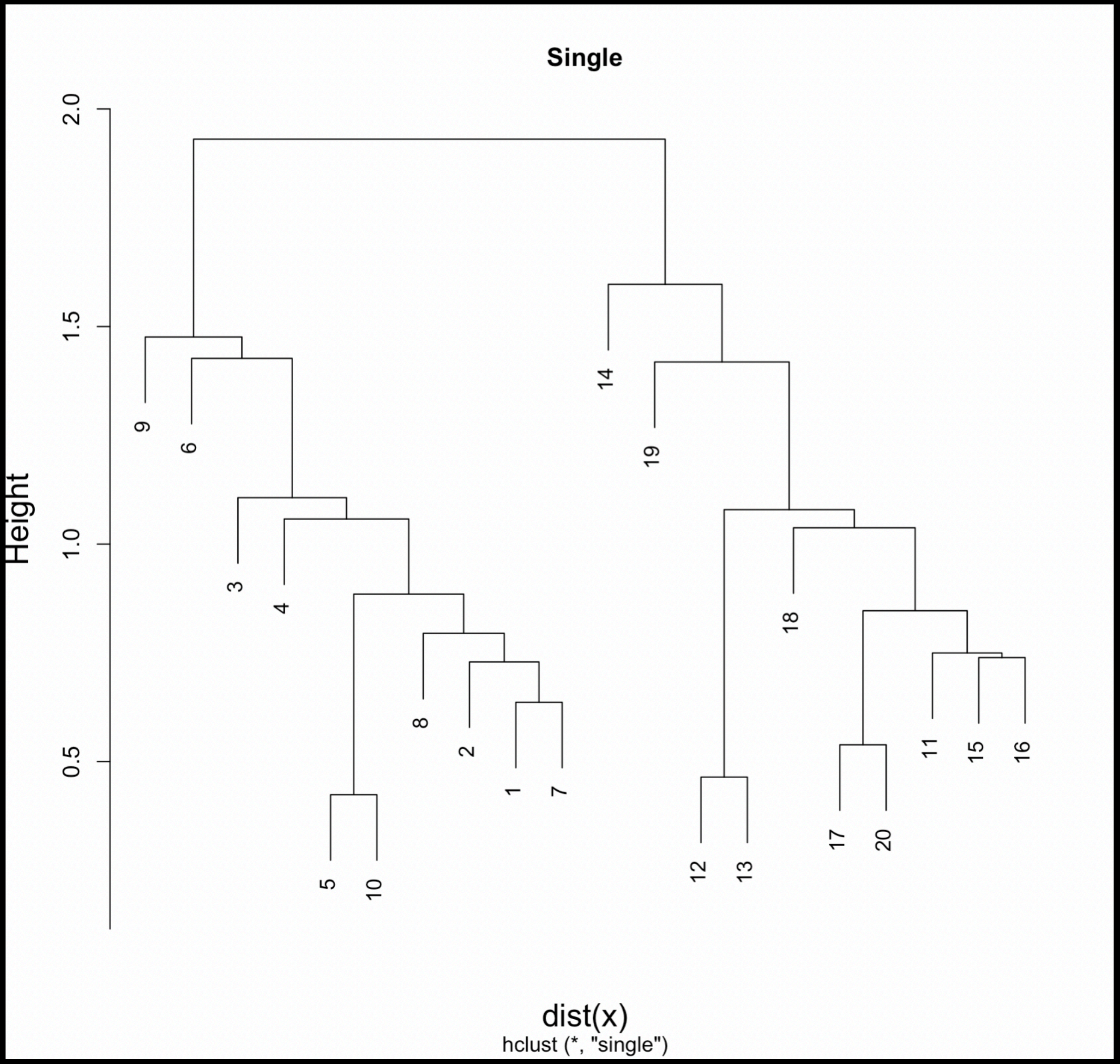
- How is distance between clusters determined?
- There are four main methods to determine which cluster should be linked:
  - ➡ **Complete**: pairwise similarity between all observations in cluster 1 and cluster 2, and uses largest of similarities
  - ➡ **Single**: same as above but uses smallest of similarities
  - ➡ **Average**: same as above but uses average of similarities
  - ➡ **Centroid**: finds centroid of cluster 1 and centroid of cluster 2, and uses similarity between two centroids

Linking methods: complete and average

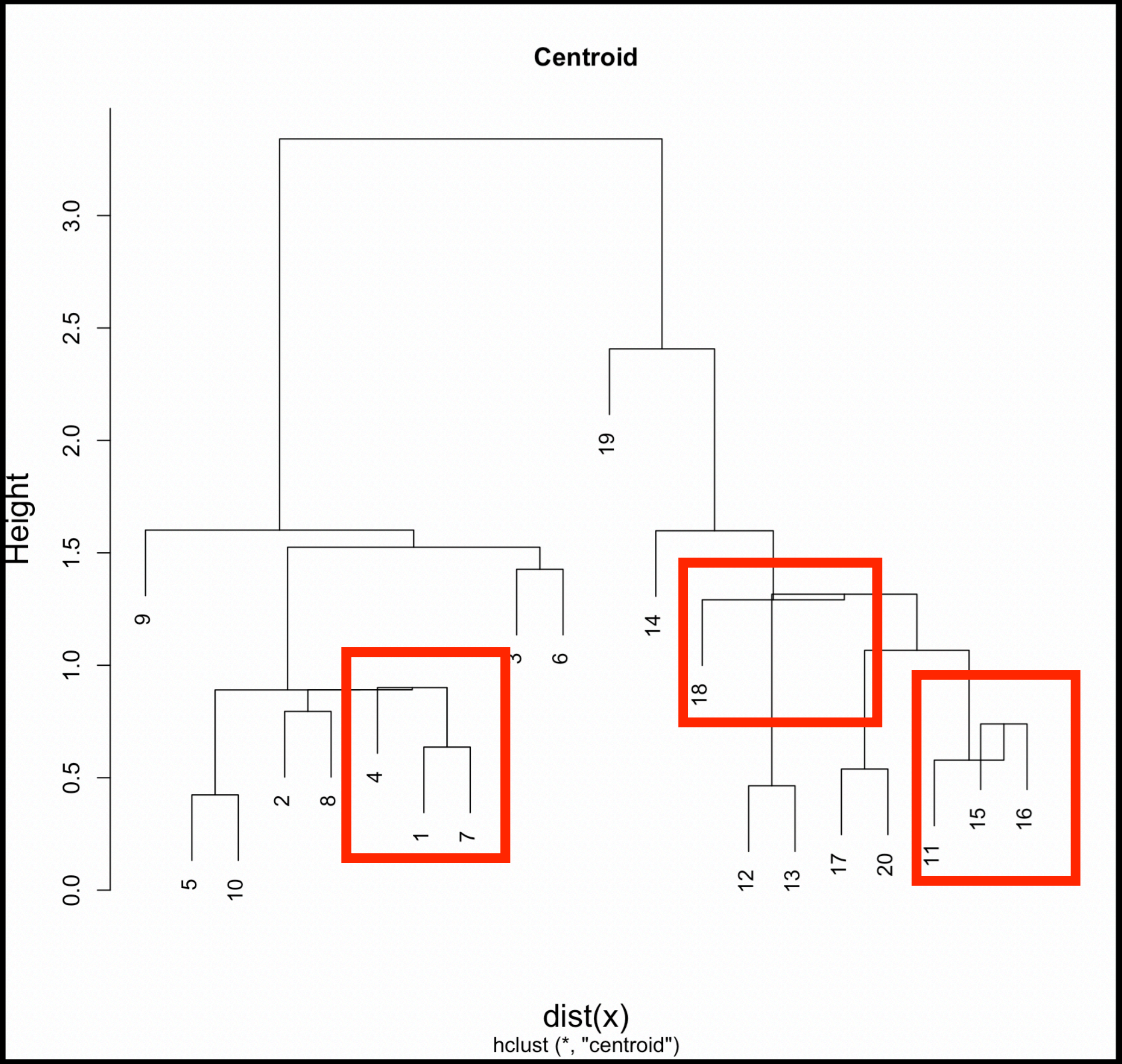




Linking method: single



Linking method: centroid



Linkage in R

```
# Using different hierarchical clustering methods
hc.complete <- hclust(d, method="complete")

hc.average  <- hclust(d, method="average")

hc.single   <- hclust(d, method="single")
```

Your Turn!

```
# Step 1. Generate some example data for clustering
x <- rbind(
  matrix(rnorm(100, mean=0, sd = 0.3), ncol = 2),   # c1
  matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2), # c2
  matrix(c(rnorm(50, mean = 1, sd = 0.3),           # c3
           rnorm(50, mean = 0, sd = 0.3)), ncol = 2))
colnames(x) <- c("x", "y")

# Step 2. Plot the data without clustering
plot(x)

# Step 3. Generate colors for known clusters
#         (just so we can compare to hclust results)
col <- as.factor( rep(c("c1","c2","c3"), each=50) )

plot(x, col=col)
```

Q. Use the **dist**(), **hclust**(), **plot**() and **cutree**() functions to return 2 and 3 clusters

Q. How does this compare to your known 'col' groups?

Today's Menu

- Introduction to machine learning
  - Unsupervised, supervised and reinforcement learning

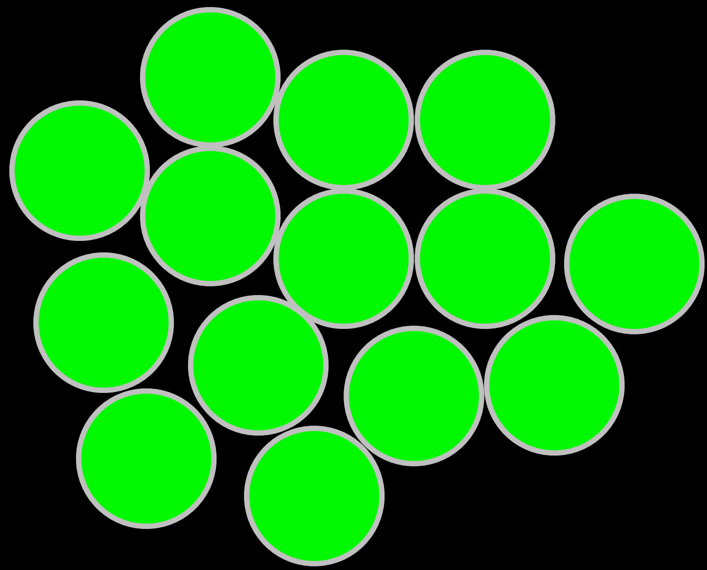
- Clustering
  - K-means clustering
  - Hierarchical clustering
  - Heatmap representations

- Dimensionality reduction, visualization and 'structure' analysis
  - Principal Component Analysis (PCA)

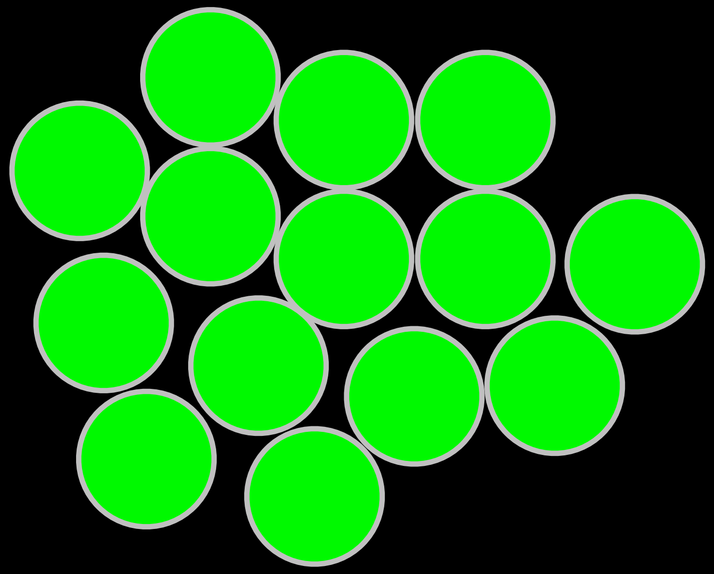
- Hands-on application to cell classification

PCA: The absolute basics

Bunch of “normal” cells

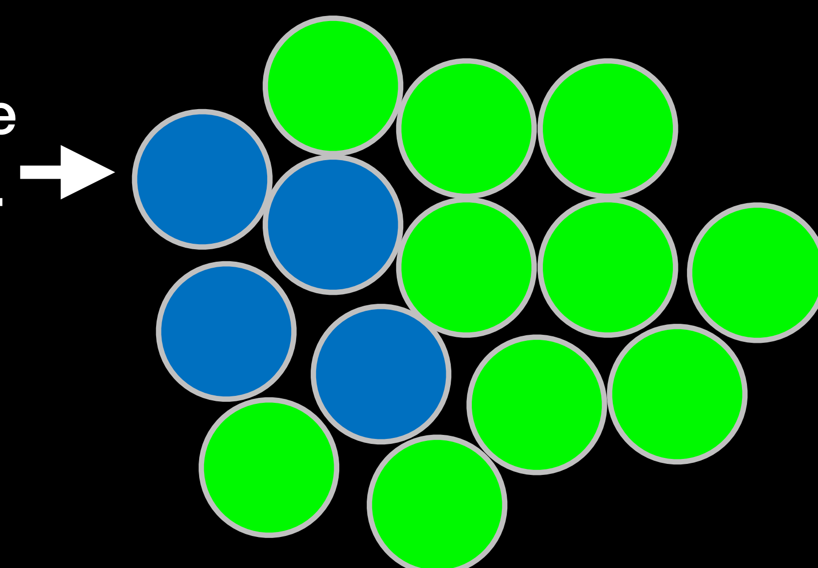


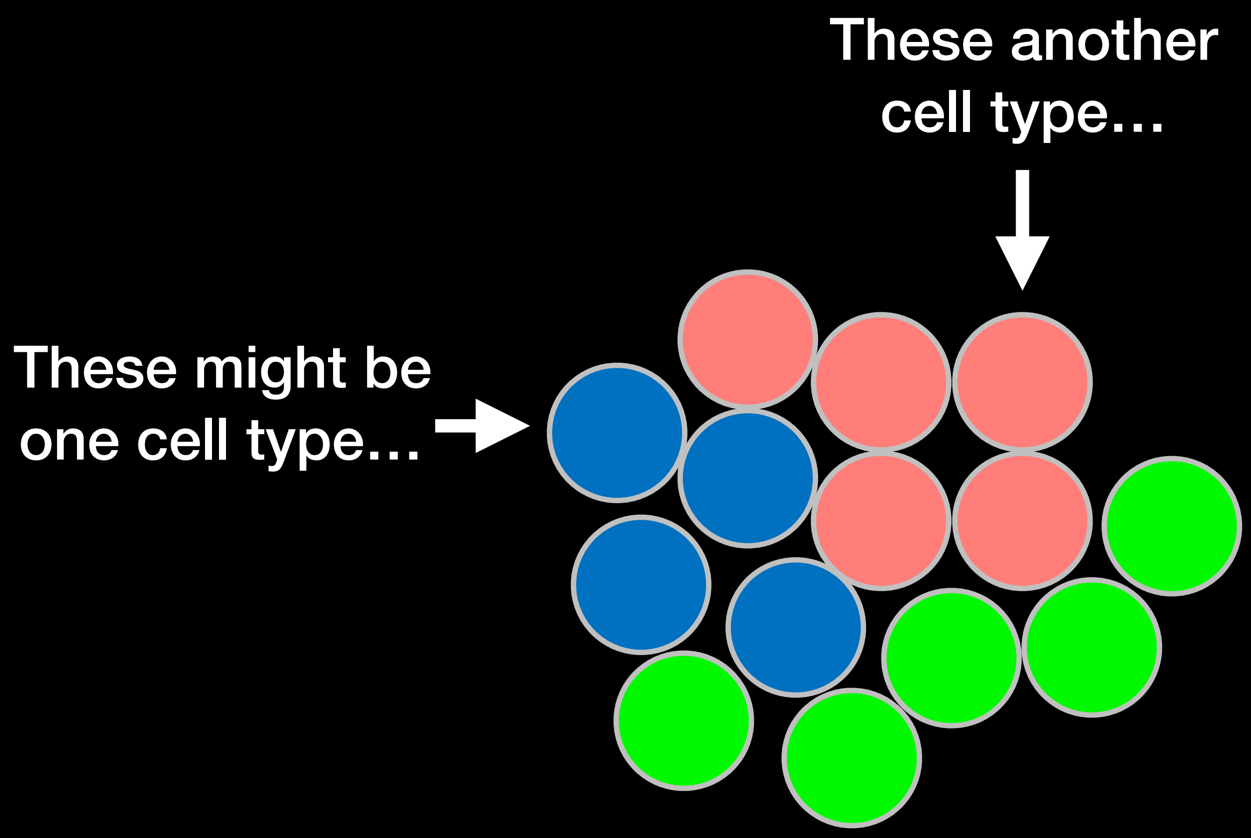
Bunch of “normal” cells

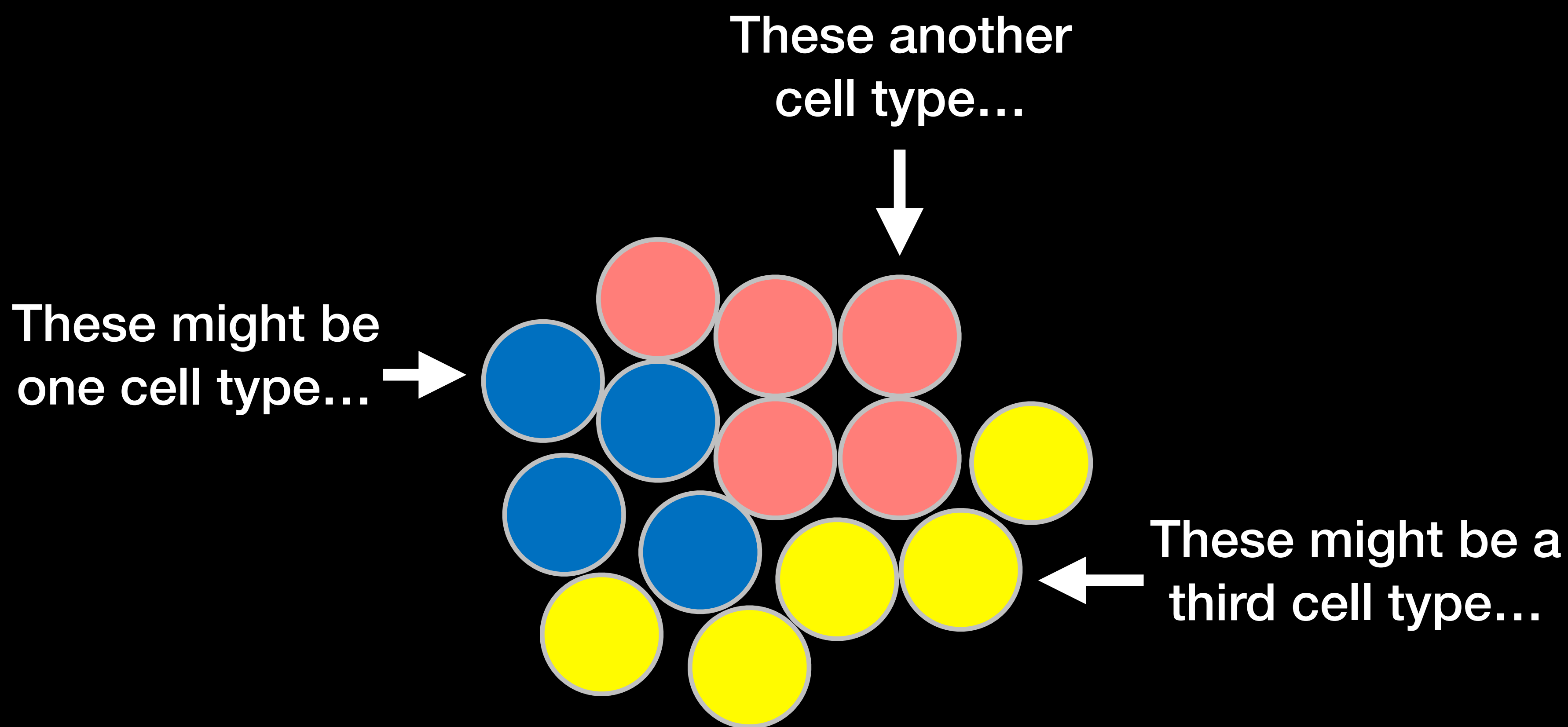


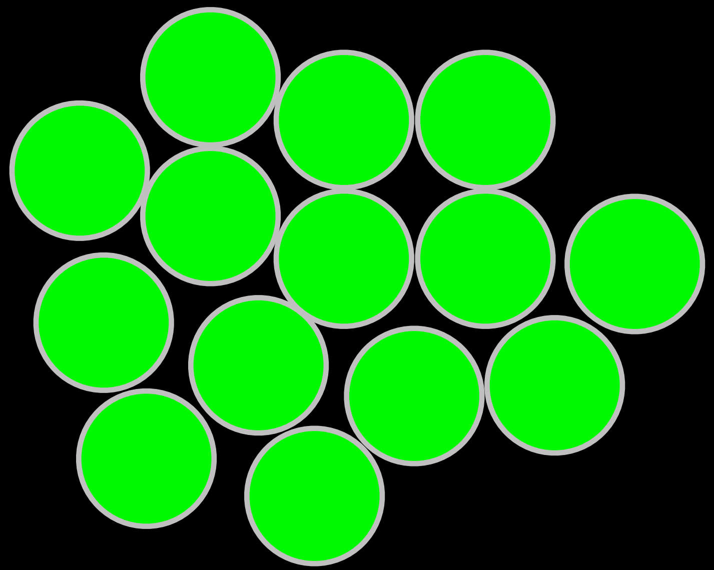
Even though they look the same we suspect that there are differences…

These might be one cell type…

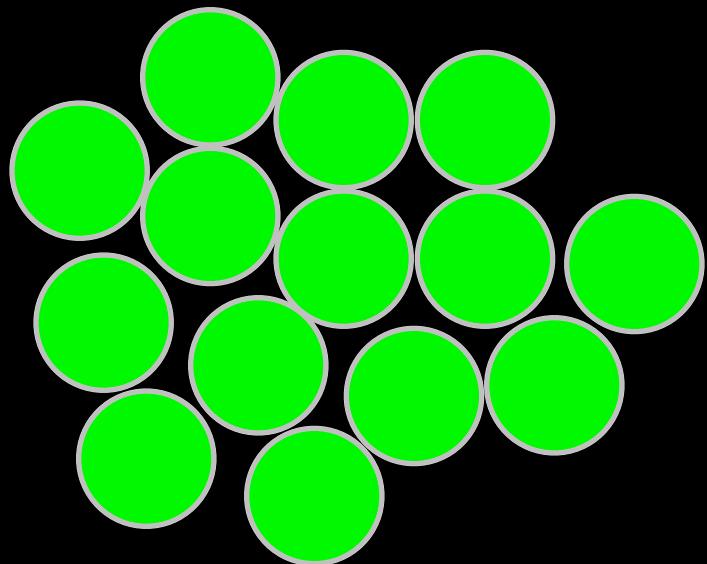



These another
cell type…
These might be
one cell type…


These another
cell type…
These might be
one cell type…
These might be a
third cell type…



Unfortunately we can’t observe
the differences visually



Unfortunately we can't observe the differences visually

So we sequence the mRNA in each cell to identify which genes are active and at what levels.

Here is the data…

	Cell1	Cell2	Cell3	Cell4	…
Gene1	3	0.25	2.8	0.1	…
Gene2	2.9	0.8	2.2	1.8	…
Gene3	2.2	1	1.5	3.2	…
Gene4	2	1.4	2	0.3	…
Gene5	1.3	1.6	1.6	0	…
Gene6	1.5	2	2.1	3	…
Gene7	1.1	2.2	1.2	2.8	…
Gene8	1	2.7	0.9	0.3	…
Gene9	0.4	3	0.6	0.1	…

Each column shows how much each gene is transcribed in each cell

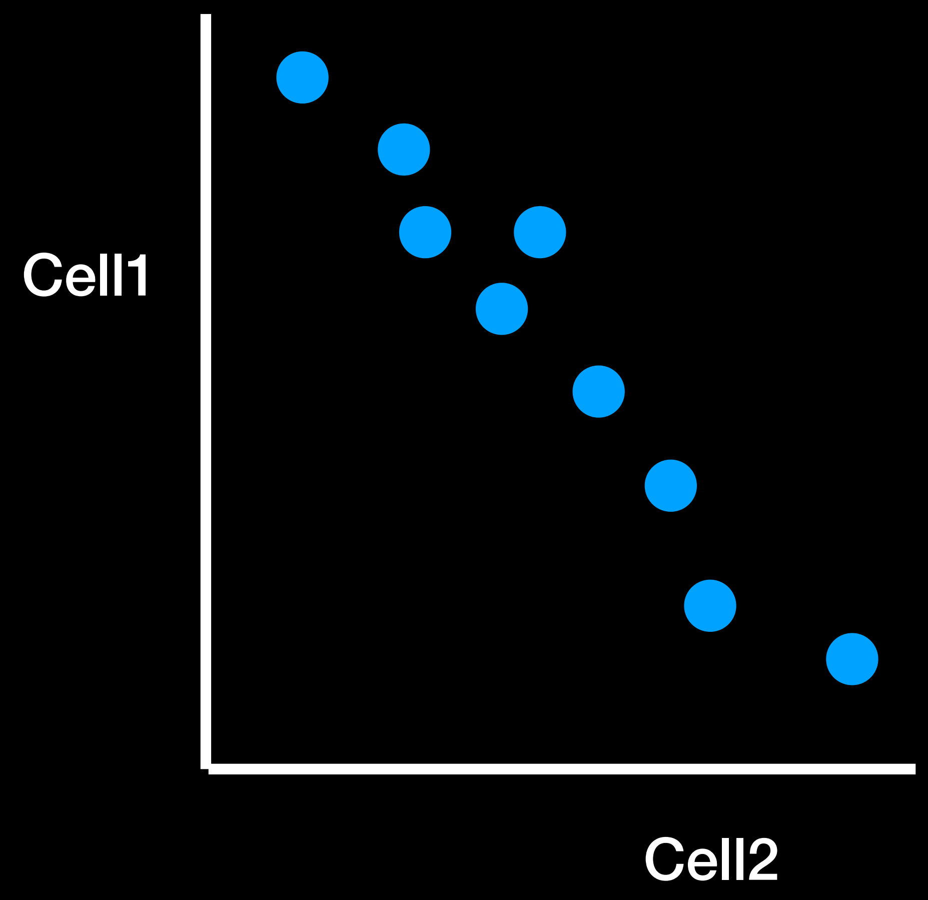
Here is the data…

	Cell1	Cell2	Cell3	Cell4	…
Gene1	3	0.25	2.8	0.1	…
Gene2	2.9	0.8	2.2	1.8	…
Gene3	2.2	1	1.5	3.2	…
Gene4	2	1.4	2	0.3	…
Gene5	1.3	1.6	1.6	0	…
Gene6	1.5	2	2.1	3	…
Gene7	1.1	2.2	1.2	2.8	…
Gene8	1	2.7	0.9	0.3	…
Gene9	0.4	3	0.6	0.1	…

For now lets consider only two cells

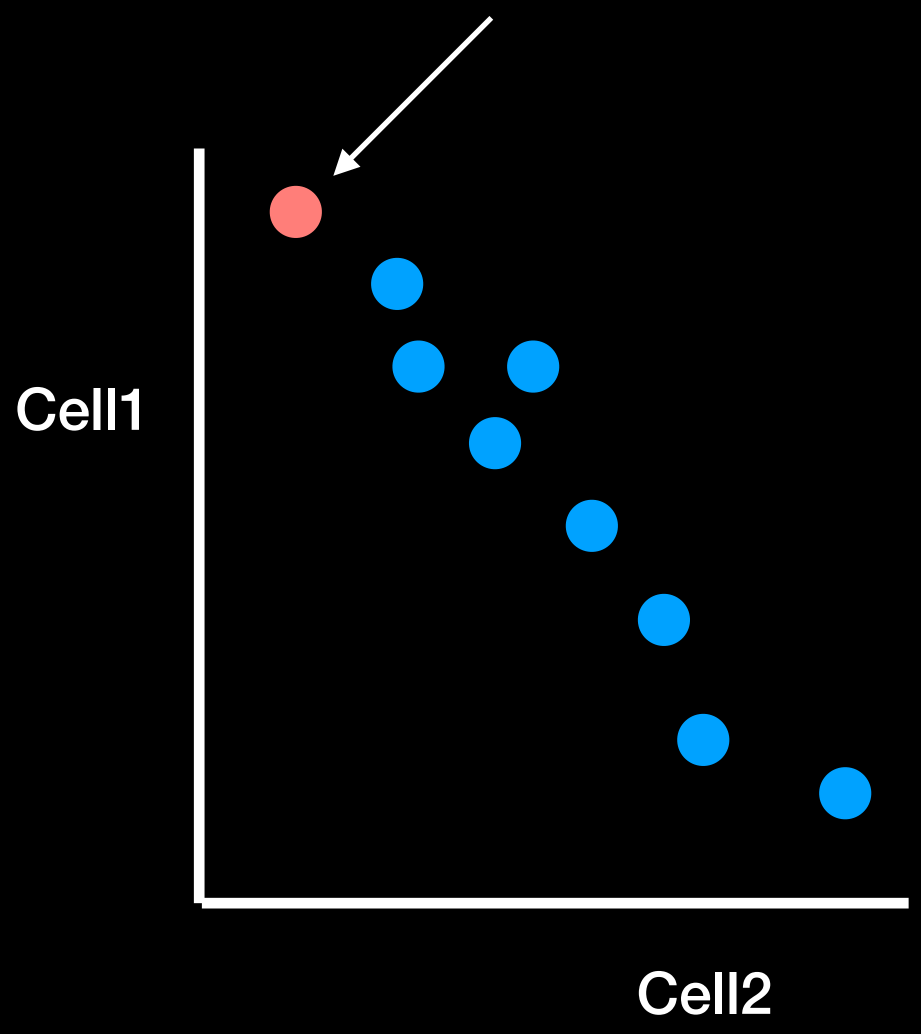
	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

We have just 2 cells so we can plot the measurements for each gene



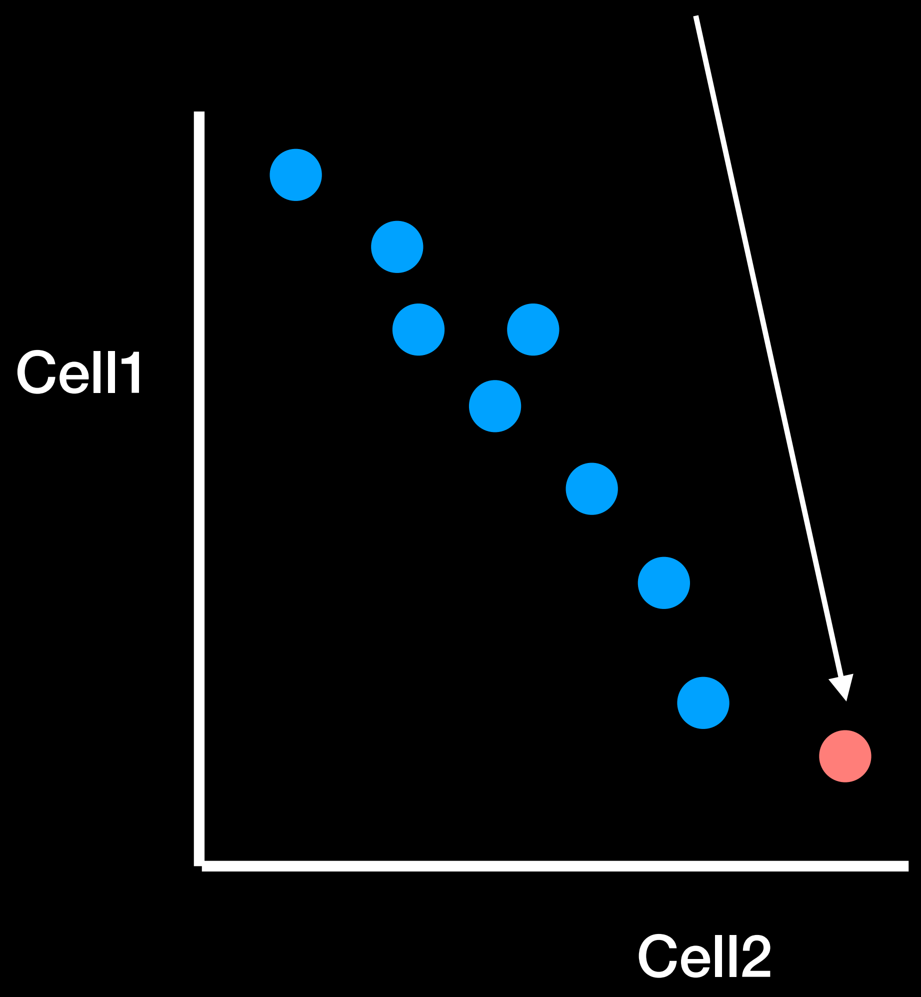
	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

This gene (Gene1) is highly transcribed in Cell1 and lowly transcribed in Cell2…



	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

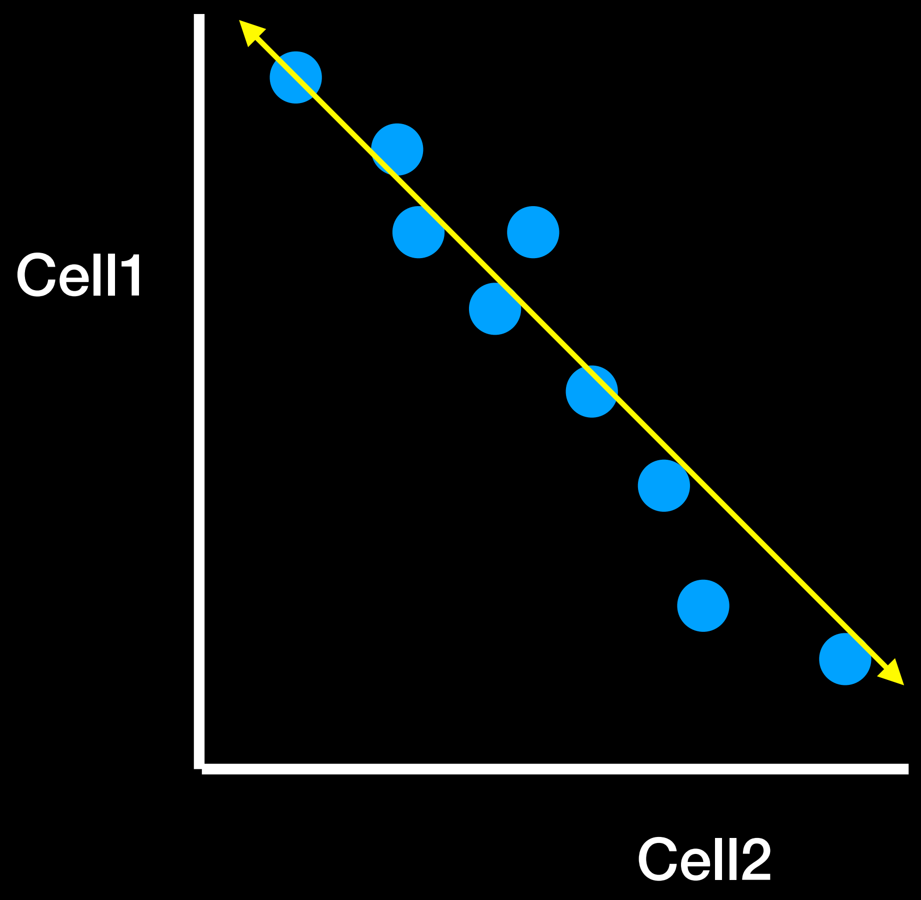
This gene (Gene9) is lowly transcribed in Cell1 and highly transcribed in Cell2…



	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

In generel, Cell1 and Cell2 have an inverse correlation.

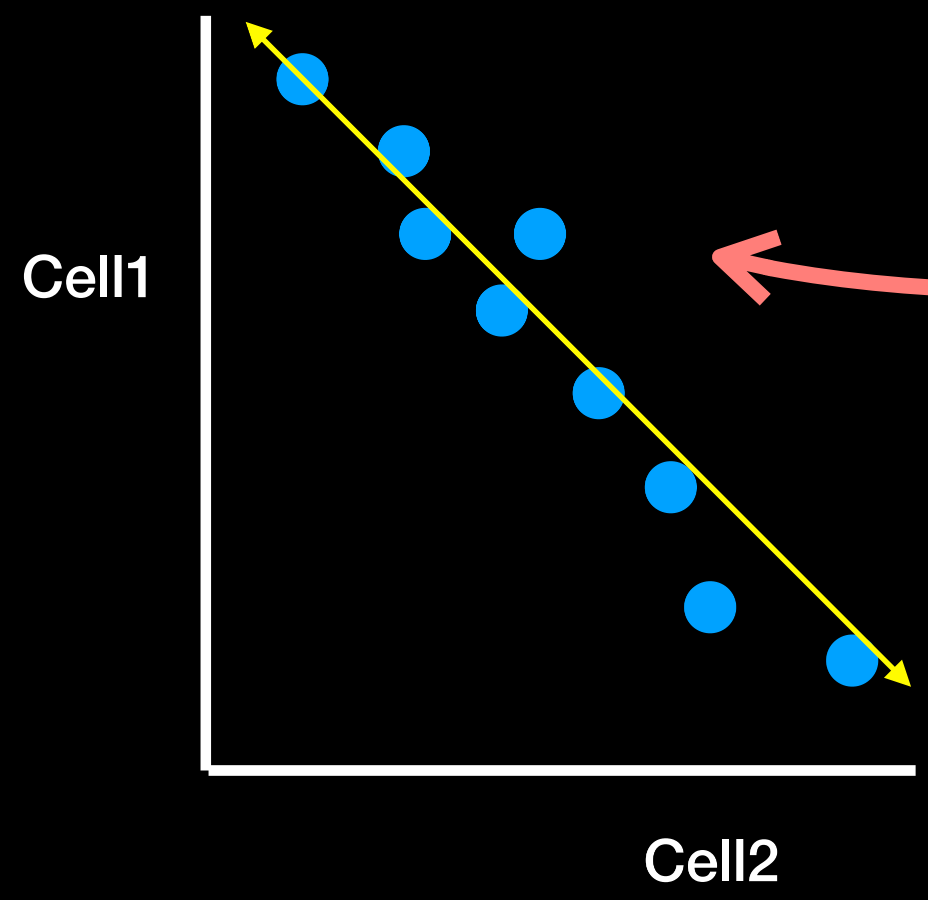
This suggests that they may be two different types of cells as they are using different genes



	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

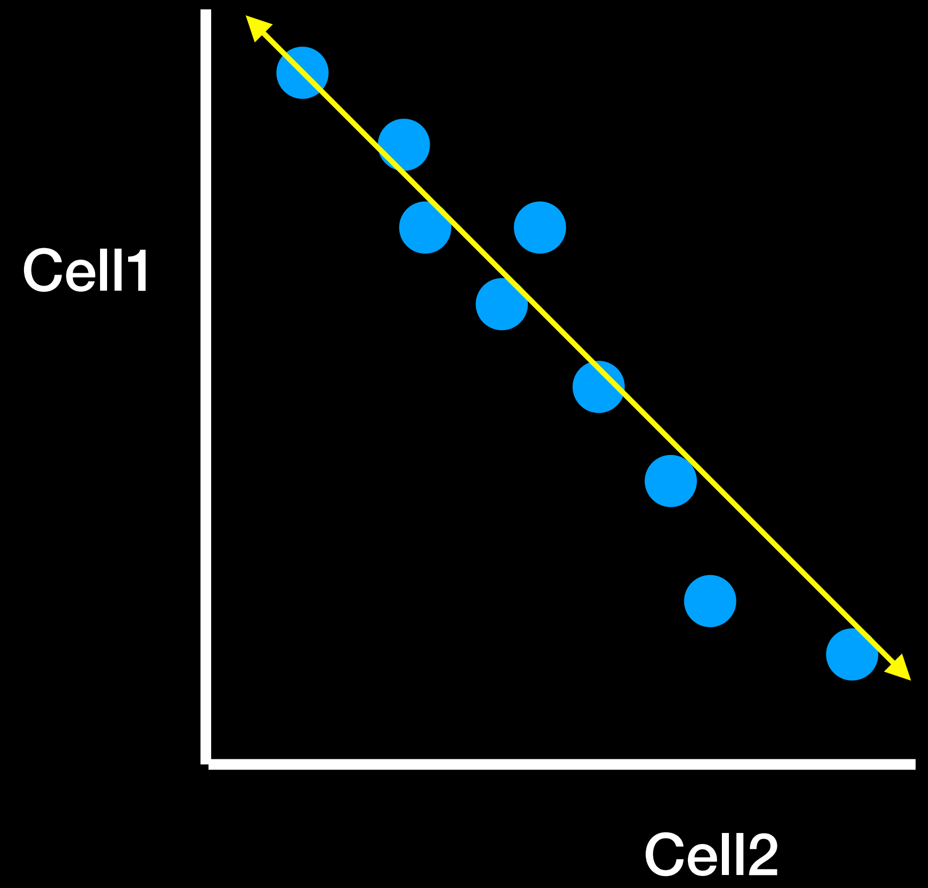
Now lets imagine there are three cells

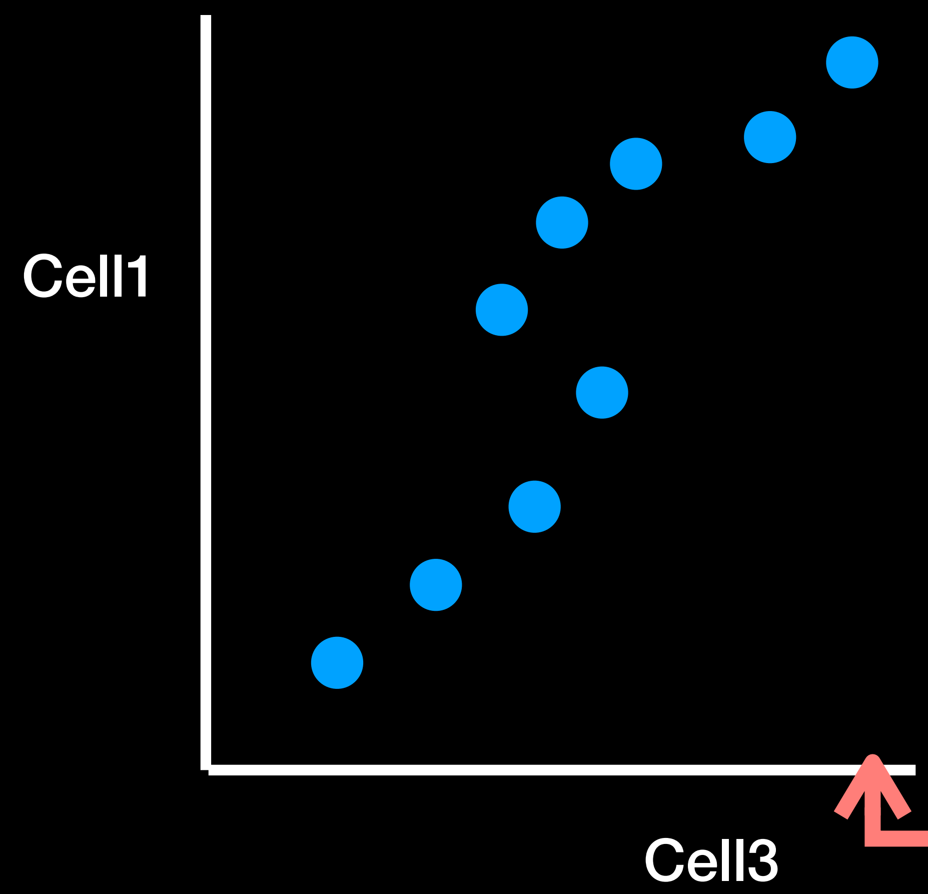
	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6



We have already seen how we can plot the first two cells to see how closely related they are

	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

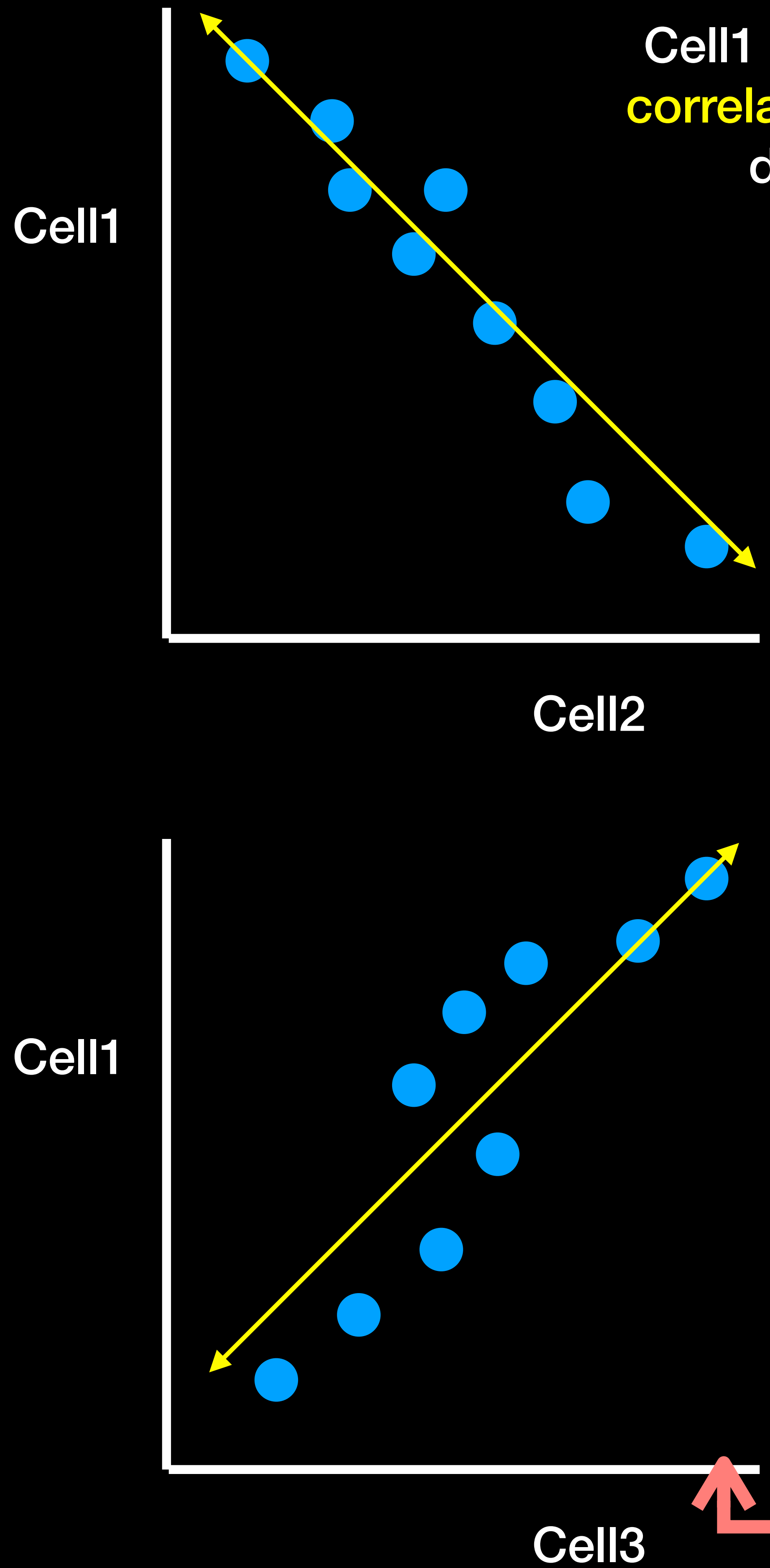




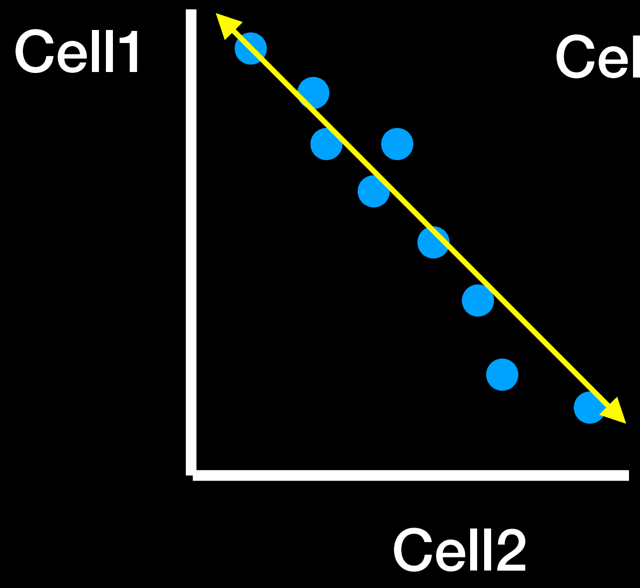
Now we can also compare Cell1 to Cell3

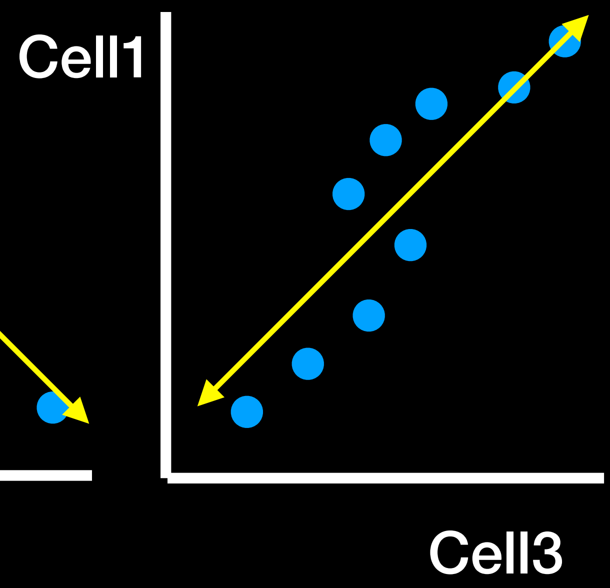
	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

Cell1 and Cell3 are positively correlated suggesting they are doing similar things



	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

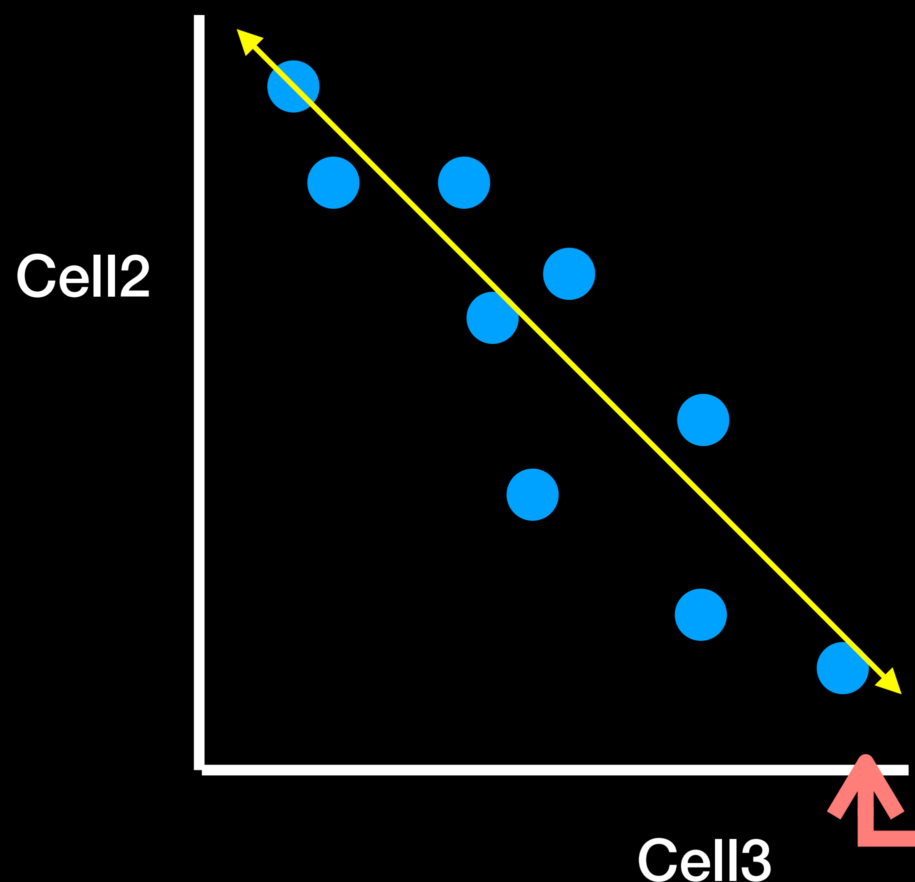




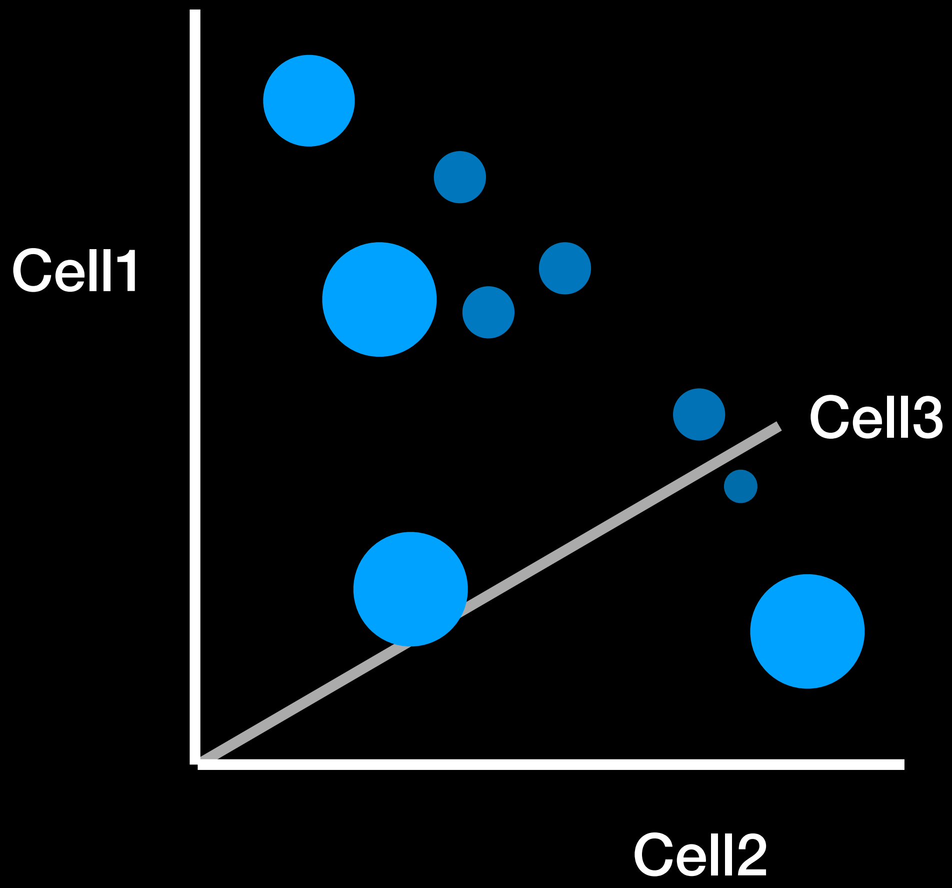
We can also compare Cell2 to Cell3…

	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

The inverse correlation suggests that Cell2 is doing something different from Cell3



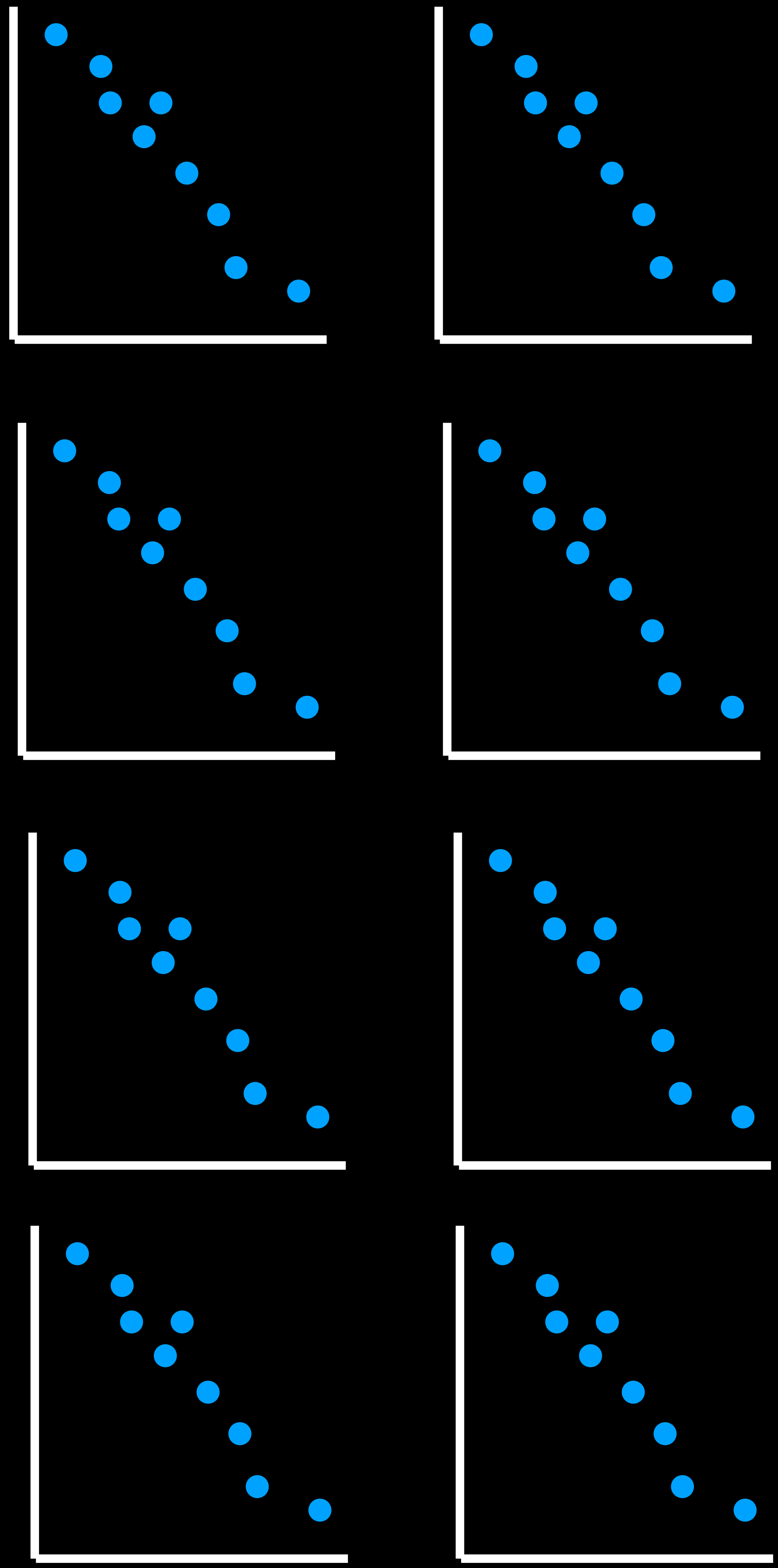
Alternatively, we could try to plot all 3 cells at once on a 3-dimensional graph.



	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

But what if we have 4 or more Cells?

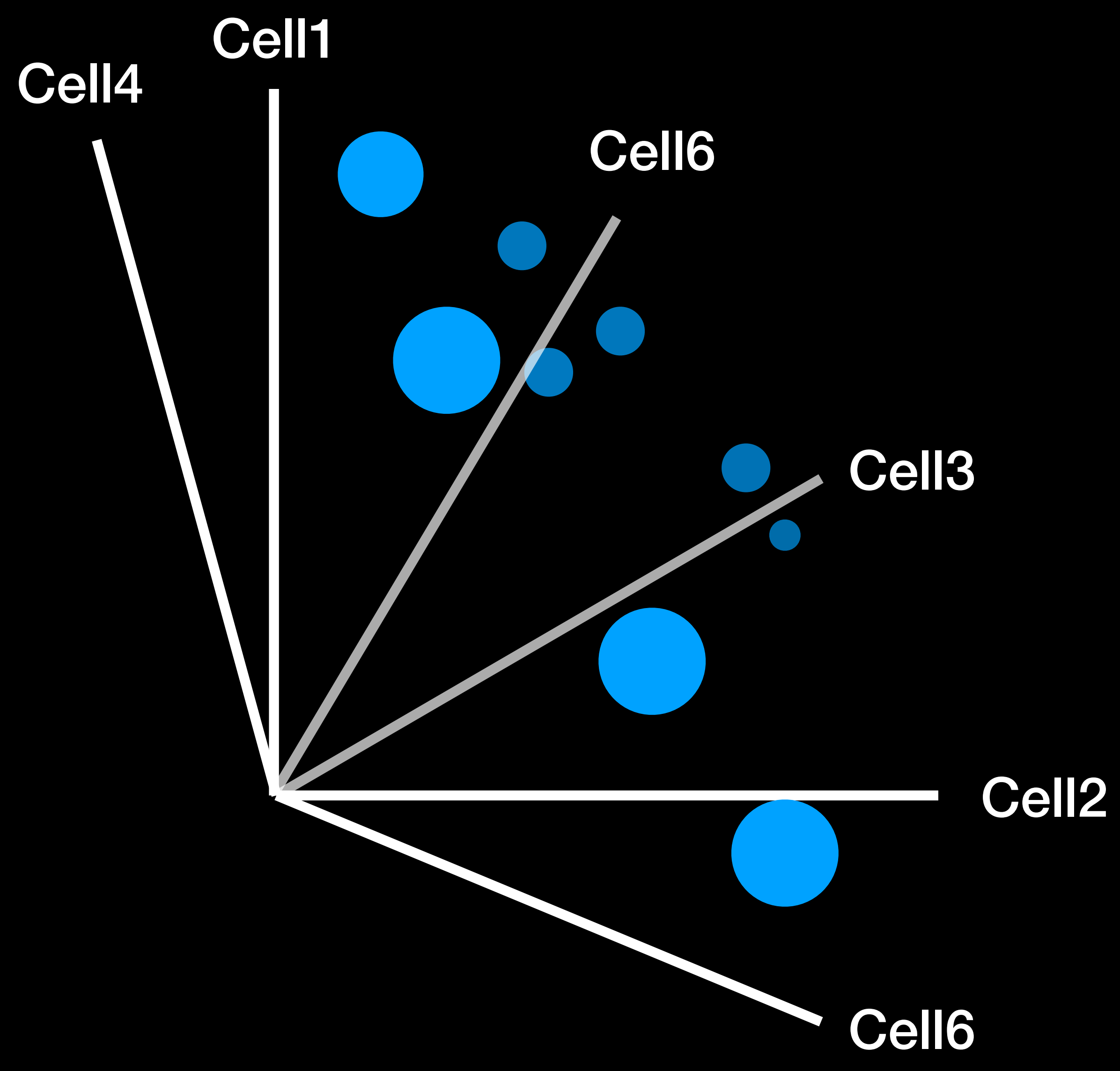
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...



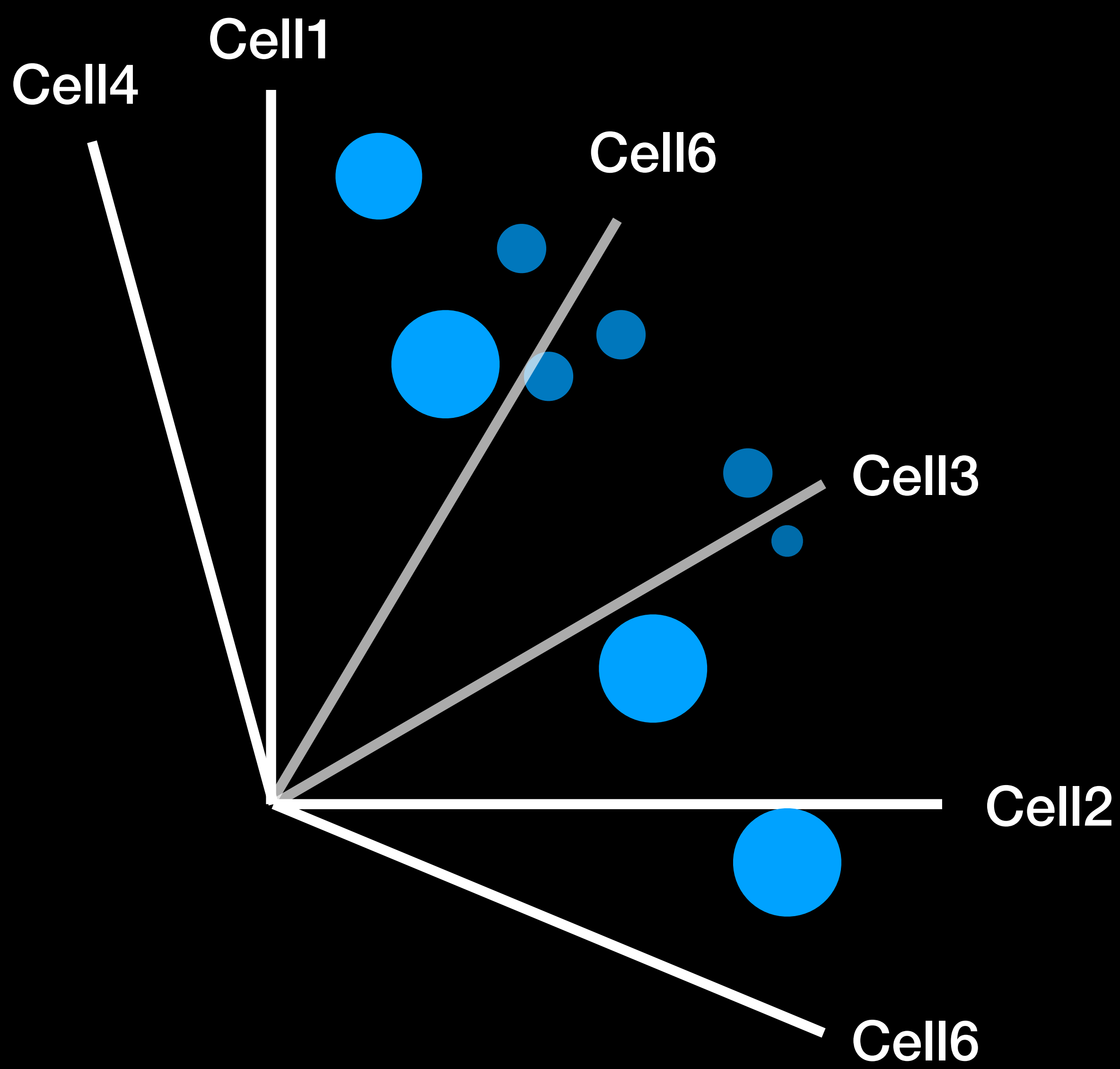
Draw lots of 2 cell plots and try to make sense of them all?

	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

Or draw some crazy graph that has an axis for each cell and makes or brains hurt!

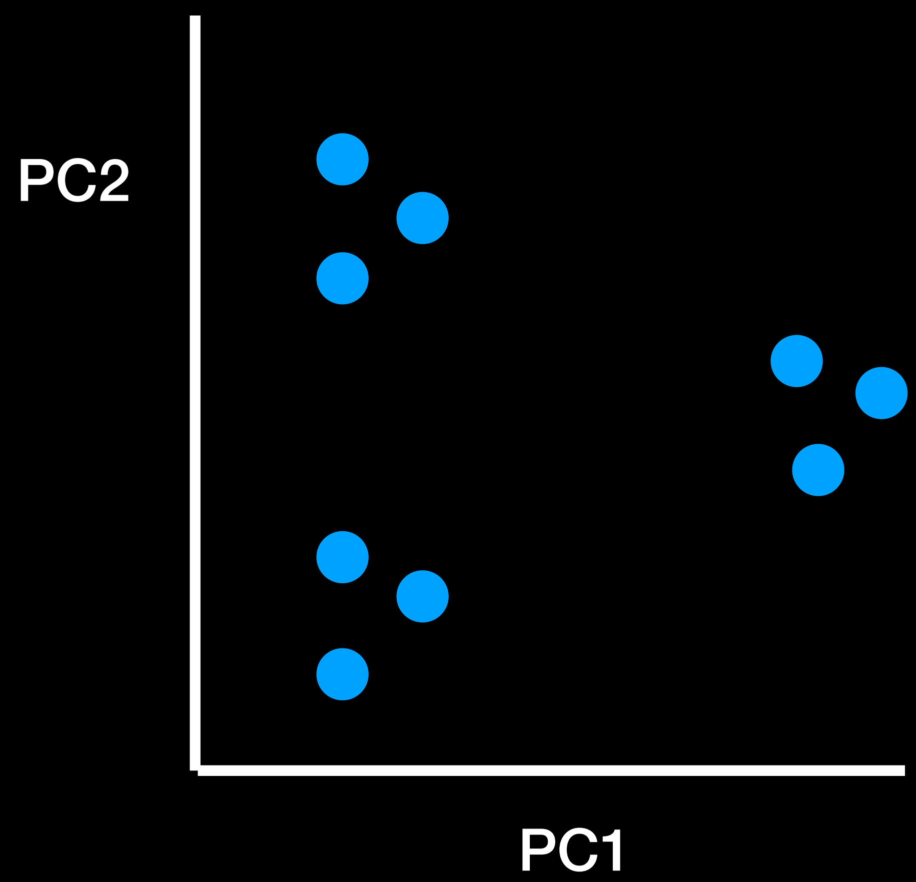


	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...



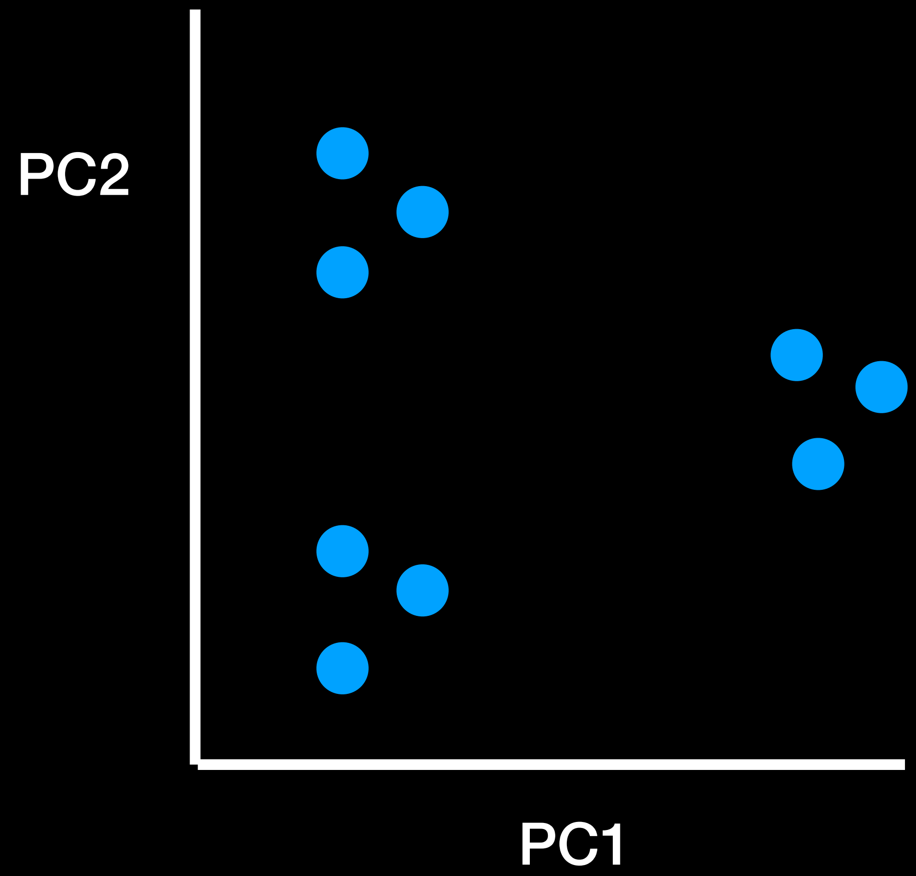
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

Enter Principal Component Analysis
(PCA)



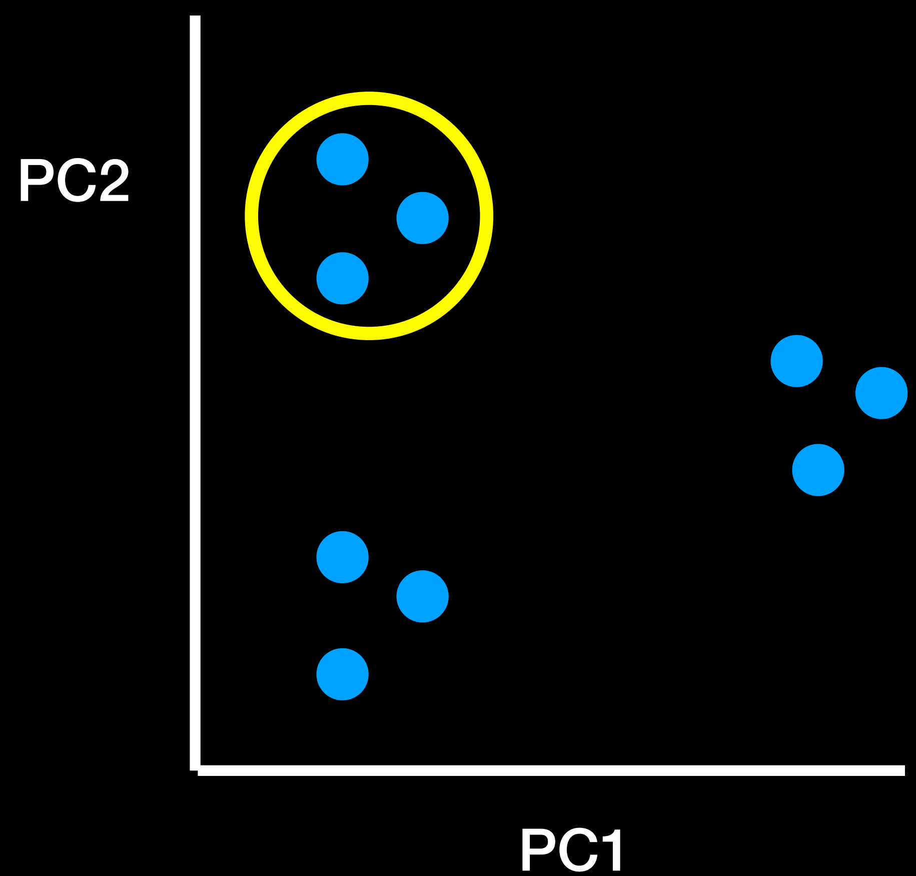
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

PCA converts the correlations (or lack there of) among all cells into a representation we can more readily interpret (e.g. a 2D graph!)



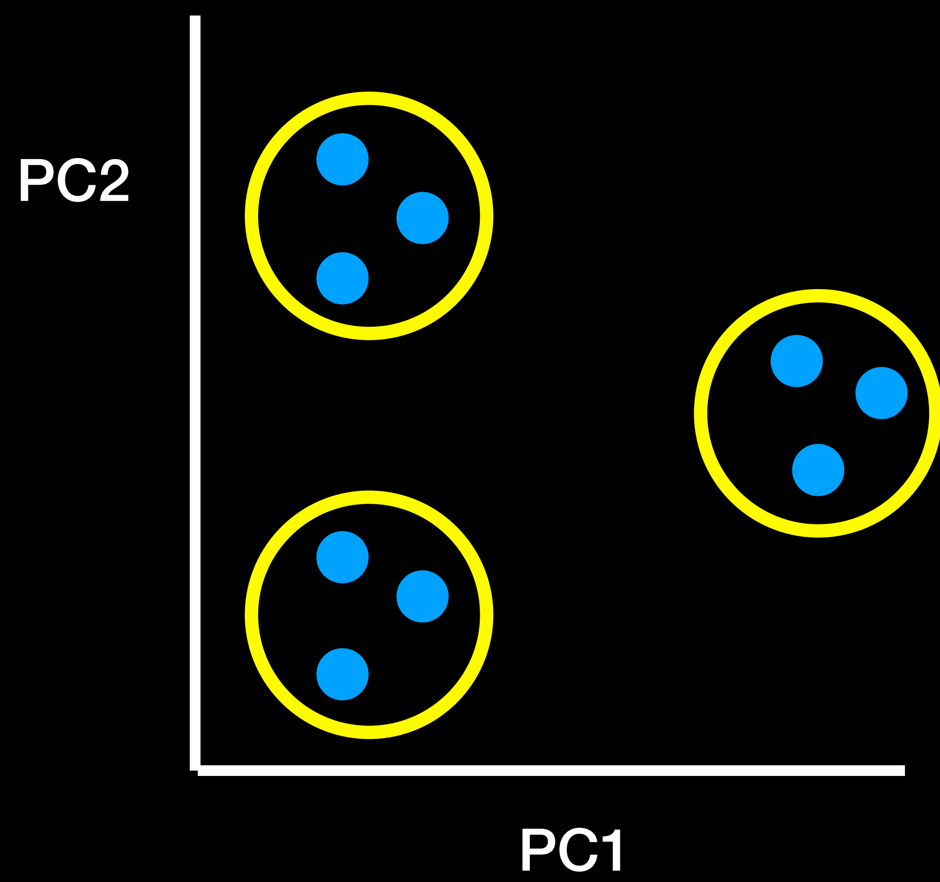
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

Cells that are highly correlated cluster together



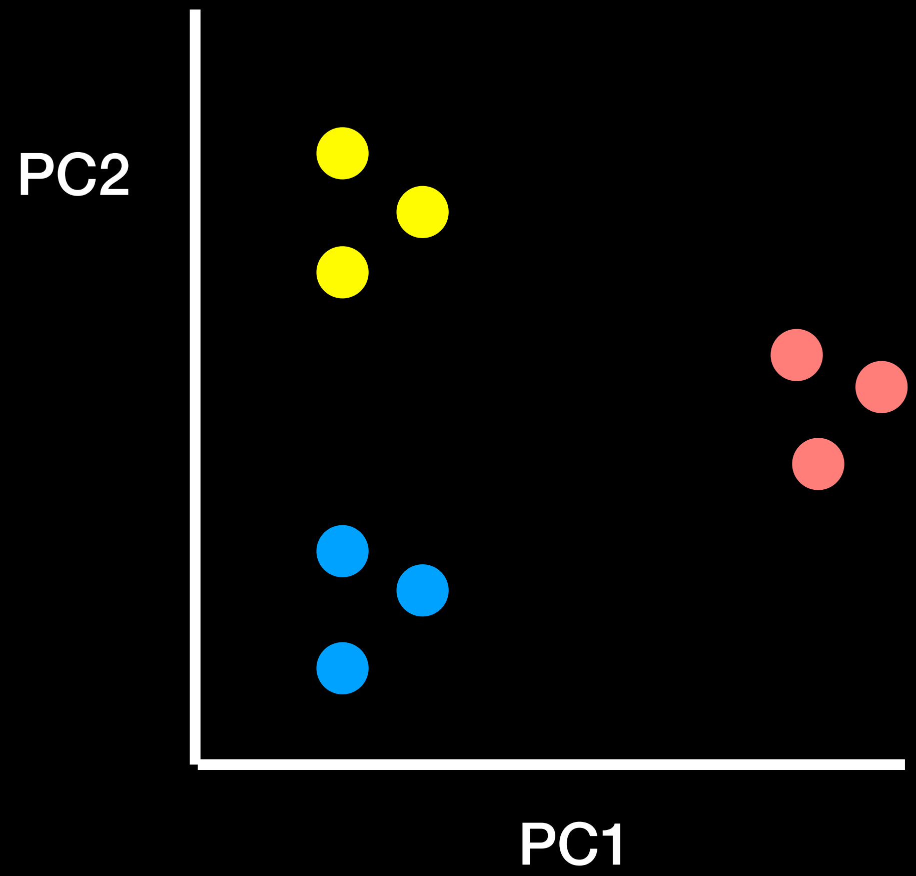
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

Cells that are highly correlated
cluster together

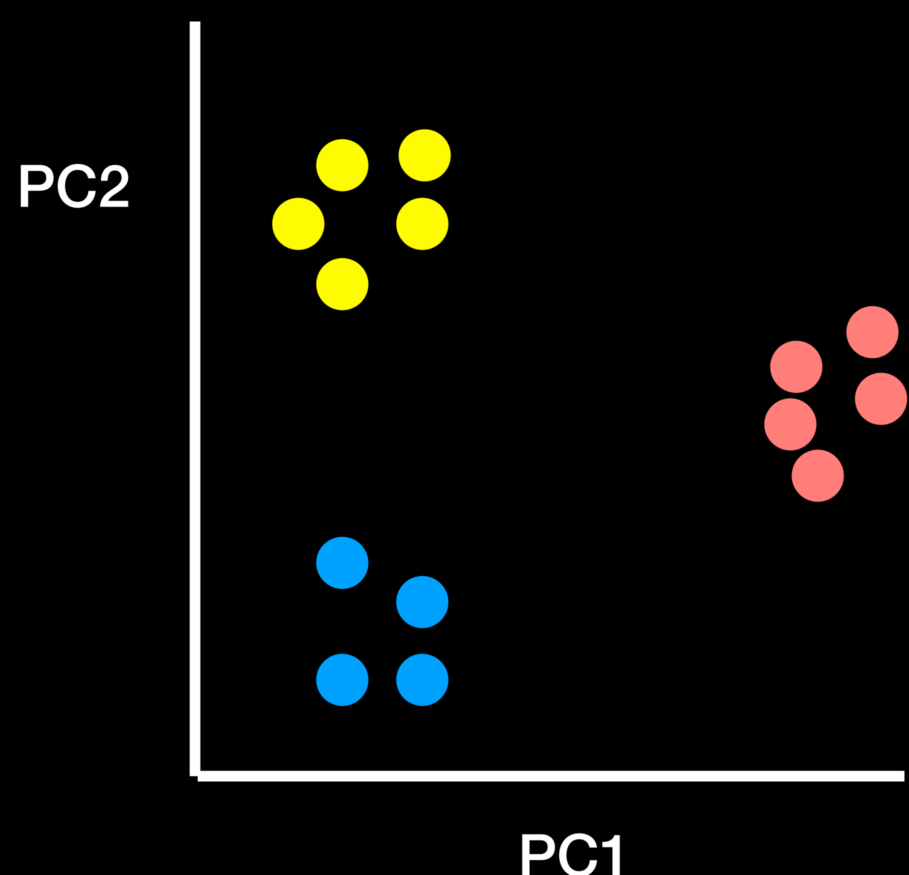


	Cell1	Cell2	Cell3	Cell4	…
Gene1	3	0.25	2.8	0.1	…
Gene2	2.9	0.8	2.2	1.8	…
Gene3	2.2	1	1.5	3.2	…
Gene4	2	1.4	2	0.3	…
Gene5	1.3	1.6	1.6	0	…
Gene6	1.5	2	2.1	3	…
Gene7	1.1	2.2	1.2	2.8	…
Gene8	1	2.7	0.9	0.3	…
Gene9	0.4	3	0.6	0.1	…

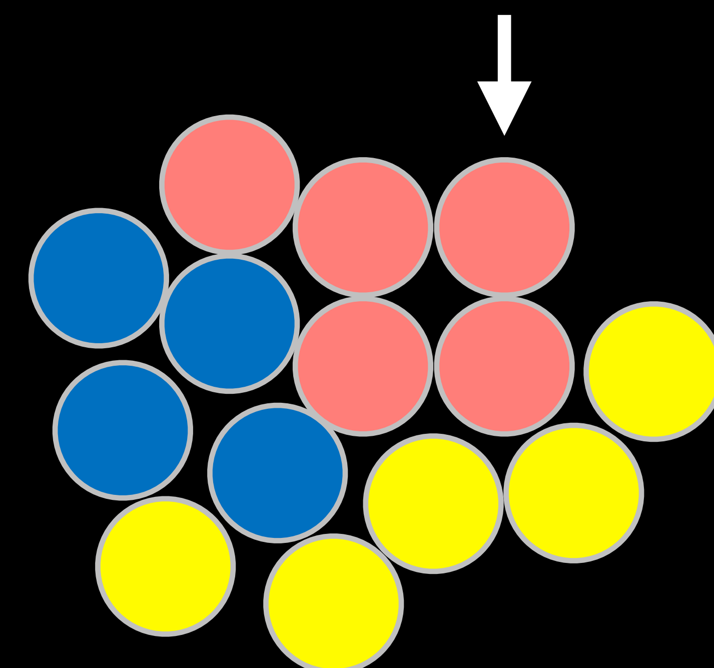
To make the clusters easier to see we can color code them…

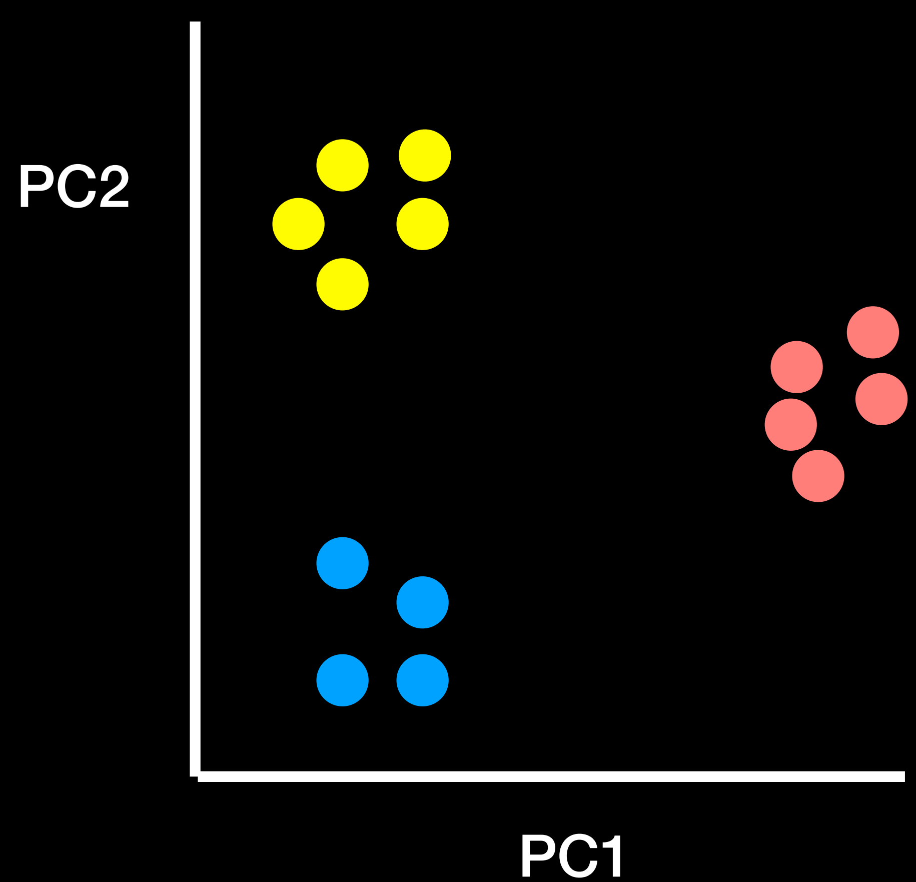


	Cell1	Cell2	Cell3	Cell4	…
Gene1	3	0.25	2.8	0.1	…
Gene2	2.9	0.8	2.2	1.8	…
Gene3	2.2	1	1.5	3.2	…
Gene4	2	1.4	2	0.3	…
Gene5	1.3	1.6	1.6	0	…
Gene6	1.5	2	2.1	3	…
Gene7	1.1	2.2	1.2	2.8	…
Gene8	1	2.7	0.9	0.3	…
Gene9	0.4	3	0.6	0.1	…

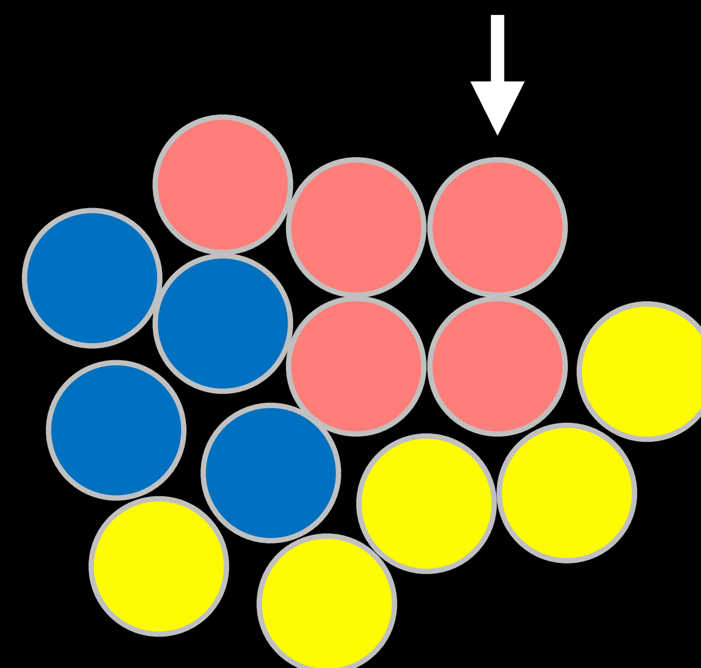


Once we have identified the clusters from our PCA results, we can go back to or original cells…

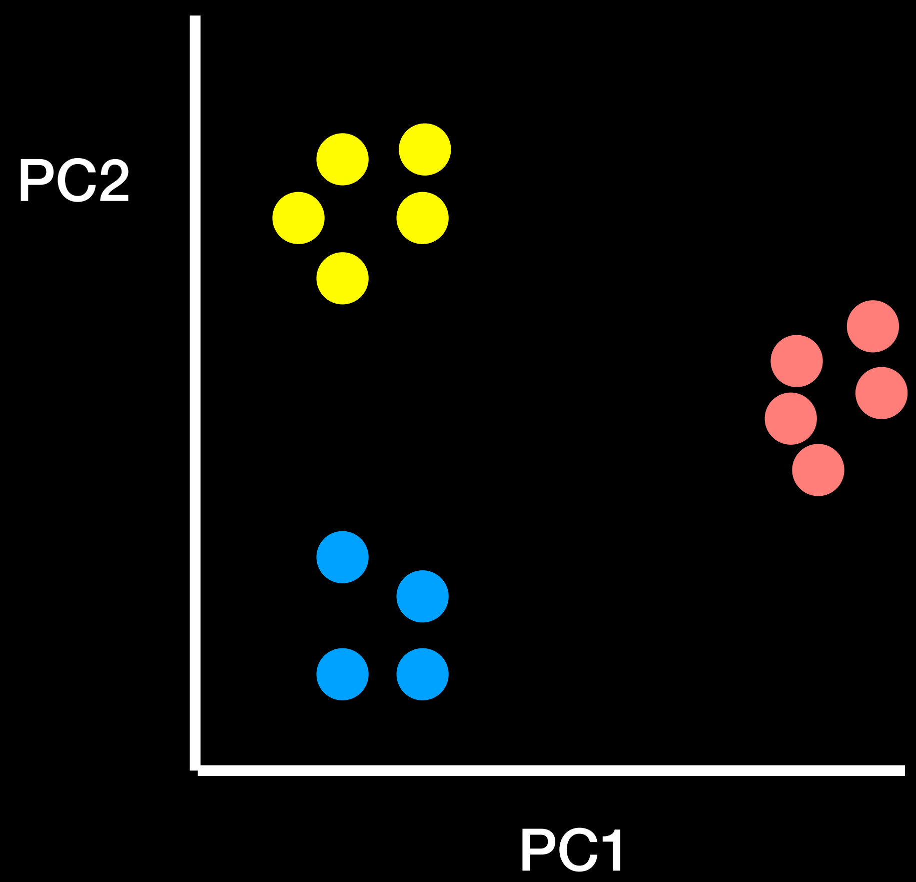




Once we have identified the clusters from our PCA results, we can go back to or original cells…



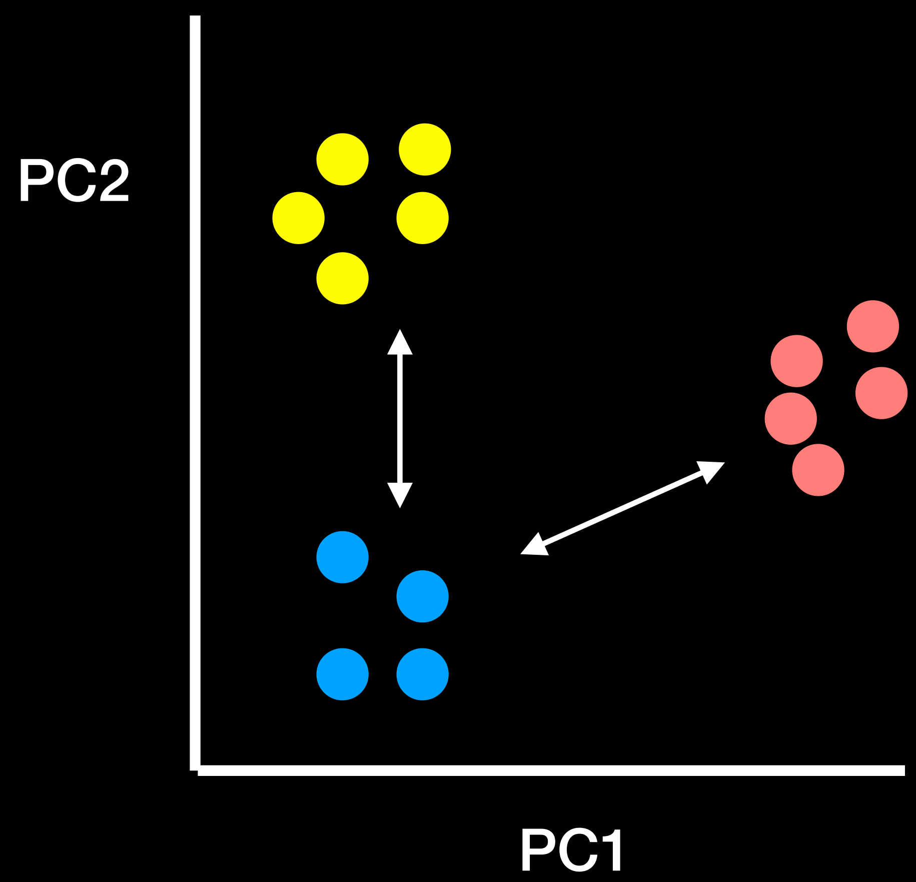
…and see they represent three different types of cells doing three different things with their genes!



Some key points:

The PCs (i.e. new plot axis) are ranked by their importance

So PC1 is more important than PC2 which in turn is more important than PC3 etc.

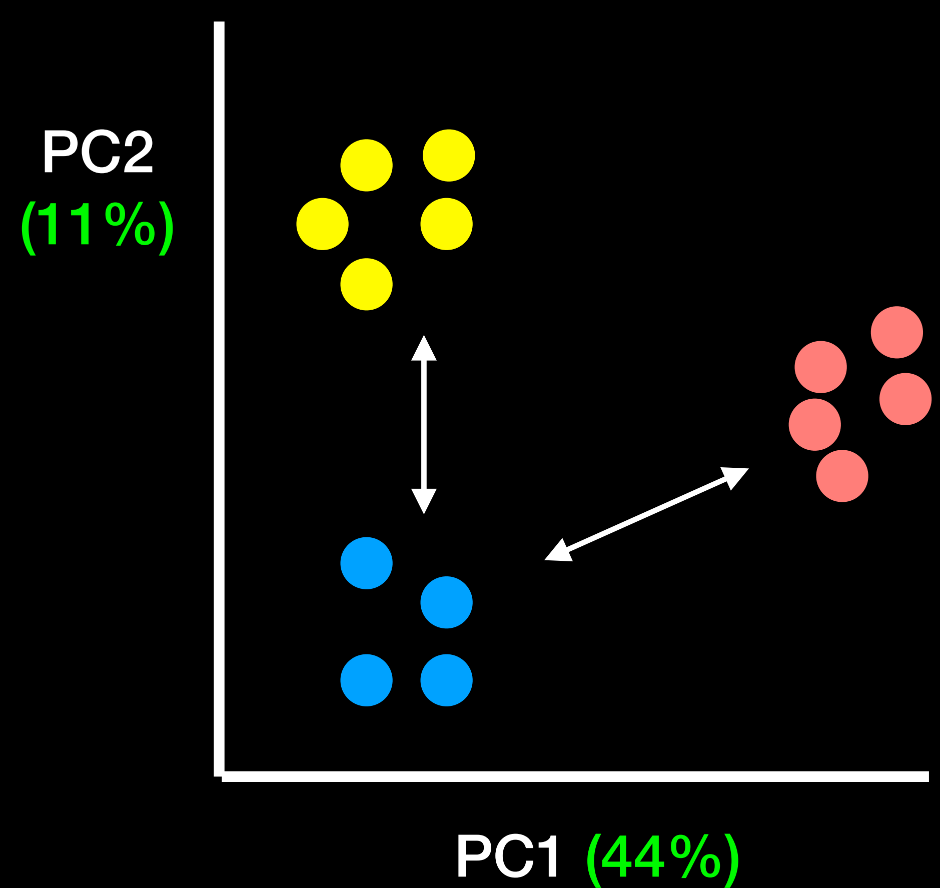


<u>Some key points</u>:

The PCs (i.e. new plot axis) are ranked by their importance

So PC1 is more important than PC2 which in turn is more important than PC3 etc.

So the red and blue cluster are more dissimilar than the yellow and blue clusters



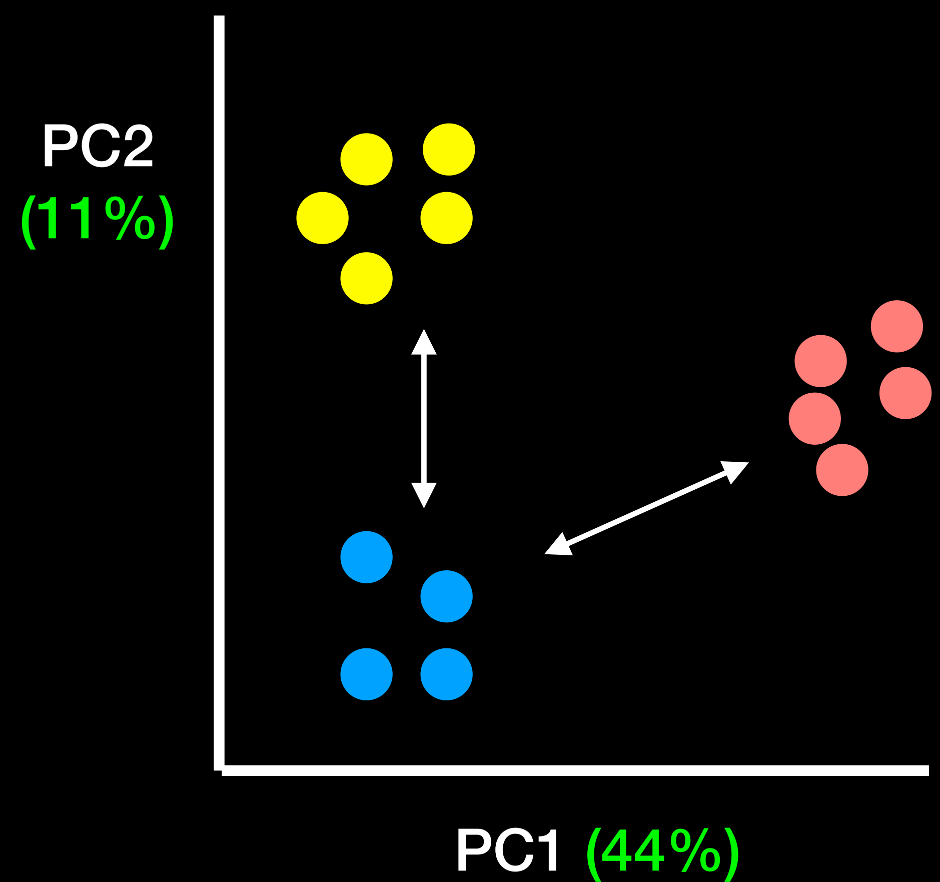
Some key points:

The PCs (i.e. new plot axis) are ranked by their importance

So PC1 is more important than PC2 which in turn is more important than PC3 etc.

So the red and blue cluster are more dissimilar than the yellow and blue clusters

The PCs (i.e. new plot axis) are ranked by the amount of variance in the original data (i.e. gene expression values) that they "capture"



Some key points:

The PCs (i.e. new plot axis) are ranked by their importance

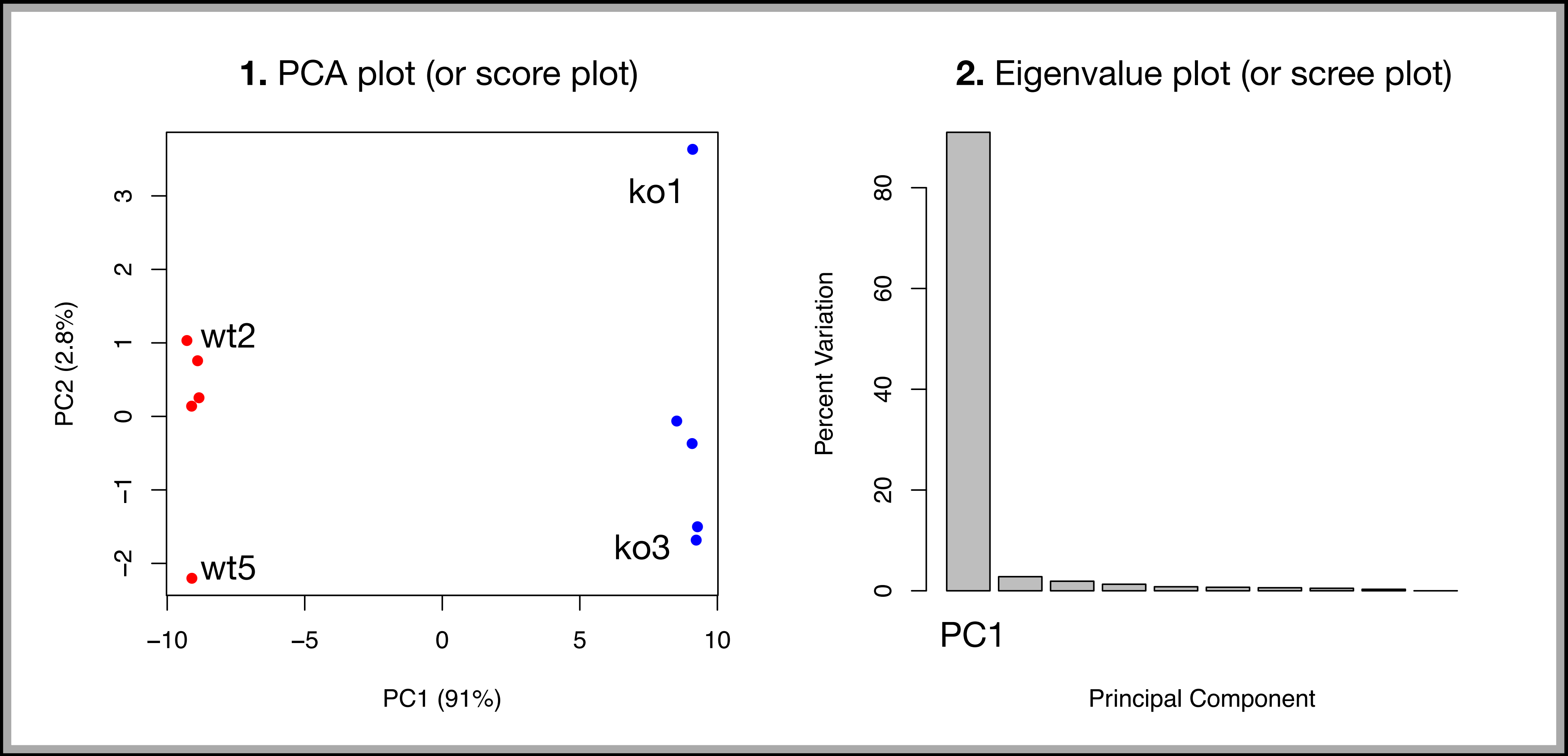
So PC1 is more important than PC2 which in turn is more important than PC3 etc.

So the red and blue cluster are more dissimilar than the yellow and blue clusters

The PCs (i.e. new plot axis) are ranked by the amount of variance in the original data (i.e. gene expression values) that they "capture"

In this example PC1 'captures' 4x more of the original variance than PC2 (44/11 = 4)

- We actually get two main things out of a typical PCA
    - The new axis (called PCs or Eigenvectors) and
    - Eigenvalues that detail the amount of variance captured by each PC



Bonus: PC Loadings

- Another cool thing we can get out of PCA is a quantitive report on how the original variables contributed to each PC
  - In other words, which were the most important genes that lead to the observed clustering in PC-space
  - These are often called the loadings and we can plot them to see which are the most important genes for the observed separation as well as outputting ranked lists of genes that act to discriminate the samples

```
   gene64     gene39
0.1047968  0.1047629
```

```
     gene7      gene65
-0.1047629 -0.1047443
```

Hands-on time!

https://bioboot.github.io/bimm143_W19/class-material/pca/

Do it Yourself!

Outline: How to do PCA in R

- How to use the **prcomp()** function to do PCA.
- How to draw and interpret PCA plots
- How to determine how much variation each principal component accounts for and the the “intrinsic dimensionality” useful for further analysis
- How to examine the **loadings** (or loading scores) to determine what variables have the largest effect on the graph and are thus important for discriminating samples.

- First lets read our example data to work with.

```
## You can also download this file from the class website!
mydata <- read.csv("https://tinyurl.com/expression-CSV",
                   row.names=1)


head(mydata)
```

```
        wt1 wt2 wt3 wt4 wt5 ko1 ko2 ko3 ko4 ko5
  gene1 147 171 160 175 187  63  57  58  55  59
  gene2 151 134 148 126 134 838 831 894 847 830
  gene3 702 672 653 681 701 593 579 644 596 610
  gene4 319 297 310 296 304 754 807 734 750 774
  gene5 168 147 162 142 152 787 811 814 869 784
```

- **NOTE**: the samples are columns, and the genes are rows!

- Now we have our data we call **prcomp()** to do PCA
  - NOTE: **prcomp()** expects the samples to be rows and genes to be columns so we need to first transpose the matrix with the **t()** function!

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)
```

- Now we have our data we call **prcomp()** to do PCA
  - NOTE: **prcomp()** expects the samples to be rows and genes to be columns so we need to first transpose the matrix with the **t()** function!

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## See what is returned by the prcomp() function
attributes(pca)

# $names
#[1] "sdev"     "rotation" "center"   "scale"    "x"
#
# $class
#[1] "prcomp"
```

- The returned pca$x here contains the principal components (PCs) for drawing our first graph.
  - Here we will take the first two columns in pca$x (corresponding to PC1 and PC2) to draw a 2-D plot

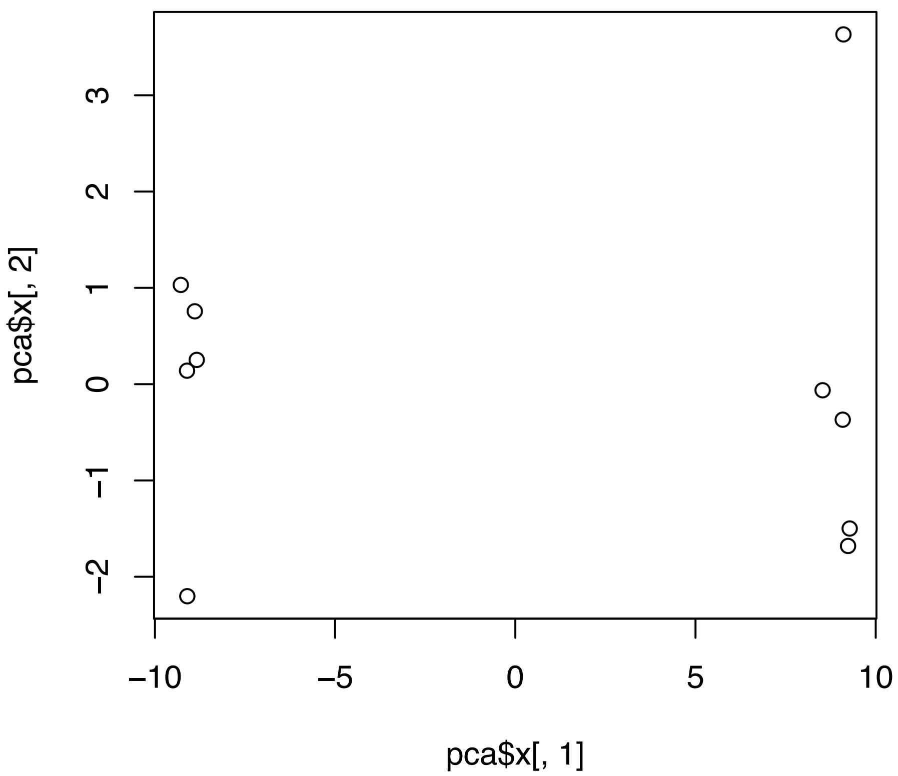
```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## See what is returned by the prcomp() function
attributes(pca)

# $names
#[1] "sdev"     "rotation" "center"   "scale"    "x"
#
# $class
#[1] "prcomp"
```

- The returned pca$x here contains the principal components (PCs) for drawing our first graph.
    - Here we will take the first two columns in pca$x (corresponding to PC1 and PC2) to draw a 2-D plot

```r
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## A basic PC1 vs PC2 2-D plot
plot(pca$x[,1], pca$x[,2])
```

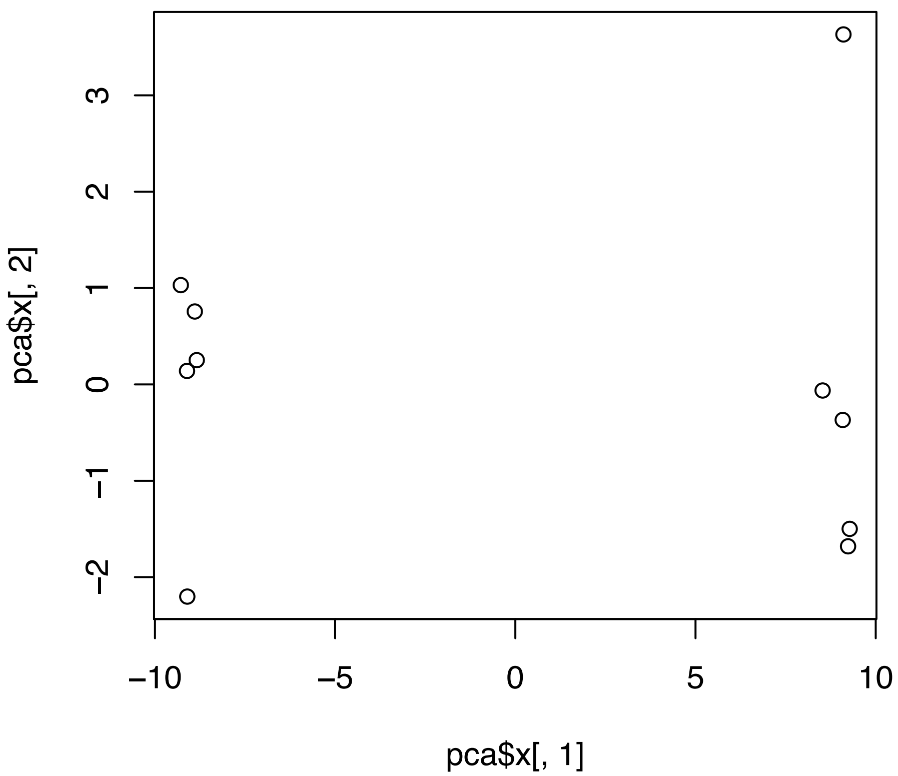
- The returned pca$x here contains the principal components (PCs) for drawing our first graph.
  - Here we will take the first two columns in pca$x (corresponding to PC1 and PC2) to draw a 2-D plot

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## A basic PC1 vs PC2 2-D plot
plot(pca$x[,1], pca$x[,2])
```



- Looks interesting with a nice separation of samples into two groups of 5 samples each
    - Now we can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## A basic PC1 vs PC2 2-D plot
plot(pca$x[,1], pca$x[,2])

## Variance captured per PC
pca.var <- pca$sdev^2
```



- Looks interesting with a nice separation of samples into two groups of 5 samples each
  - Now we can use the square of pca$sdev , which stands for “standard deviation”, to calculate how much variation in the original data each PC accounts for

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## A basic PC1 vs PC2 2-D plot
plot(pca$x[,1], pca$x[,2])

## Precent variance is often more informative to look at
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
```

- Looks interesting with a nice separation of samples into two groups of 5 samples each
  - Now we can use the square of pca$sdev , which stands for “standard deviation”, to calculate how much variation in the original data each PC accounts for

```
## lets do PCA
pca <- prcomp(t(mydata), scale=TRUE)

## A basic PC1 vs PC2 2-D plot
plot(pca$x[,1], pca$x[,2])

## Precent variance is often more informative to look at
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)

pca.var.per
```

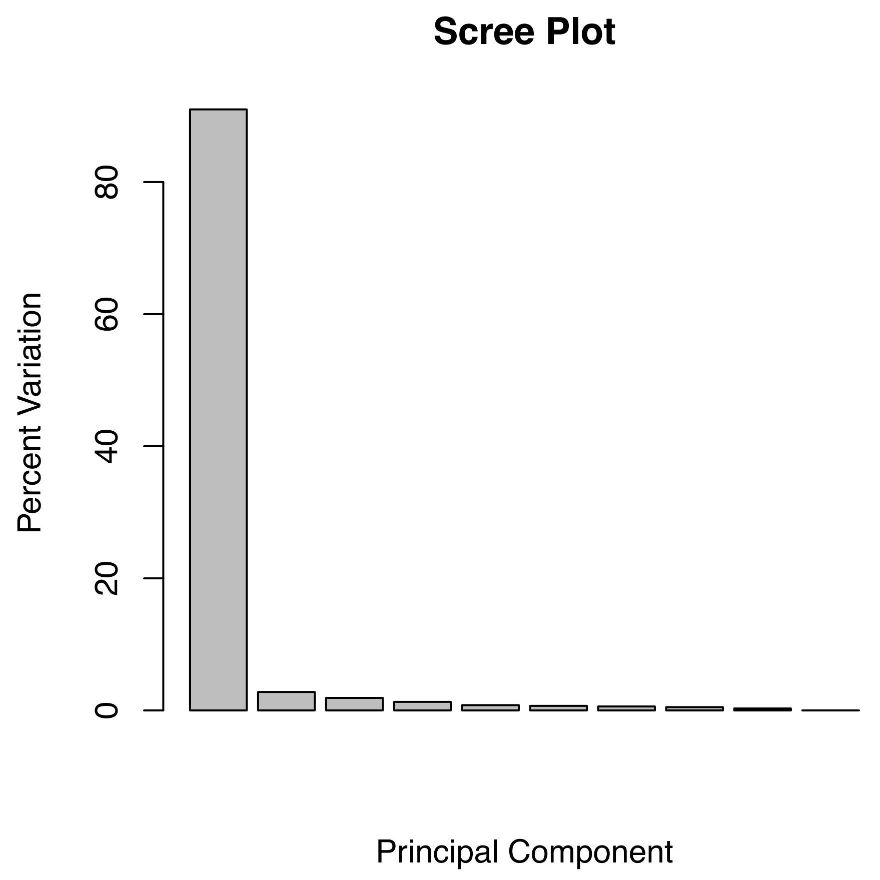
```
 [1] 91.0  2.8  1.9  1.3  0.8  0.7  0.6  0.5  0.3  0.0
```

- Looks interesting with a nice separation of samples into two groups of 5 samples each
  - Now we can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for

```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)

barplot(pca.var.per, main="Scree Plot",
        xlab="Principal Component", ylab="Percent Variation")
```

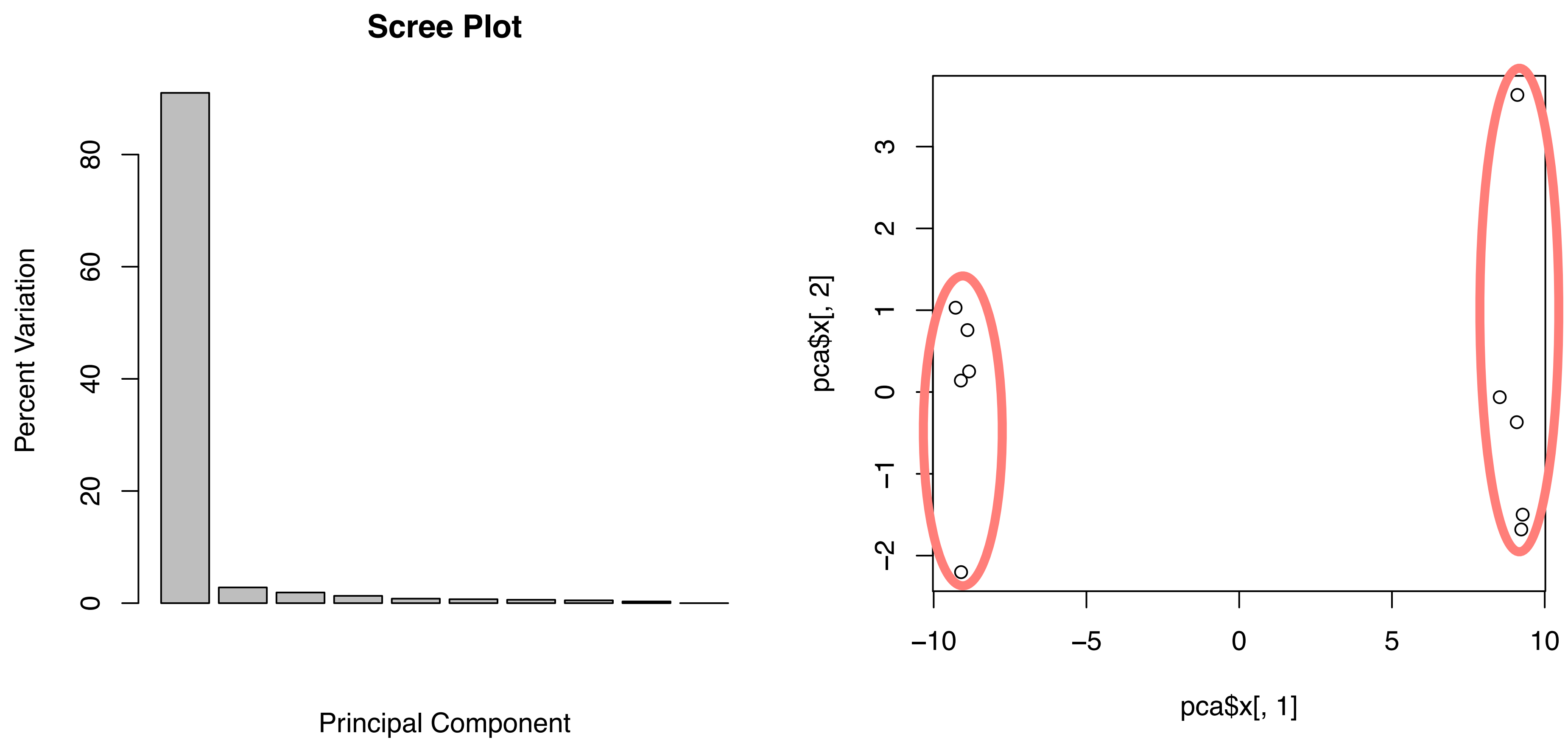
- From the "scree plot" it is clear that **PC1** accounts for almost all of the variation in the data!

```r
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)

barplot(pca.var.per, main="Scree Plot",
        xlab="Principal Component", ylab="Percent Variation")
```



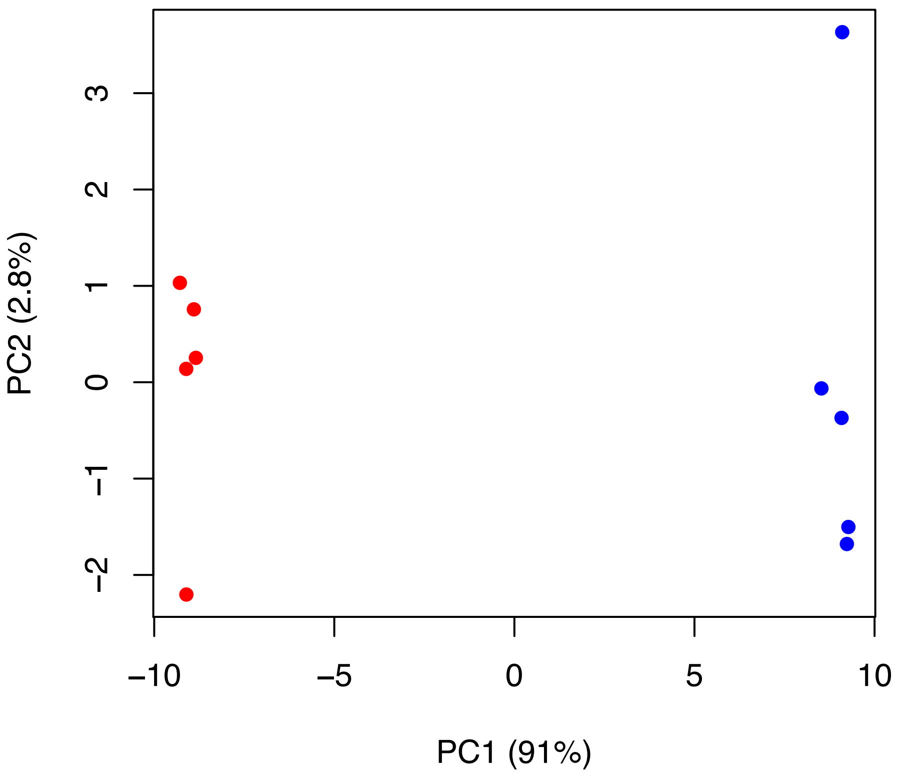
- Which means there are big differences between these two groups that are separated along the PC1 axis…

```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)

barplot(pca.var.per, main="Scree Plot",
        xlab="Principal Component", ylab="Percent Variation")
```



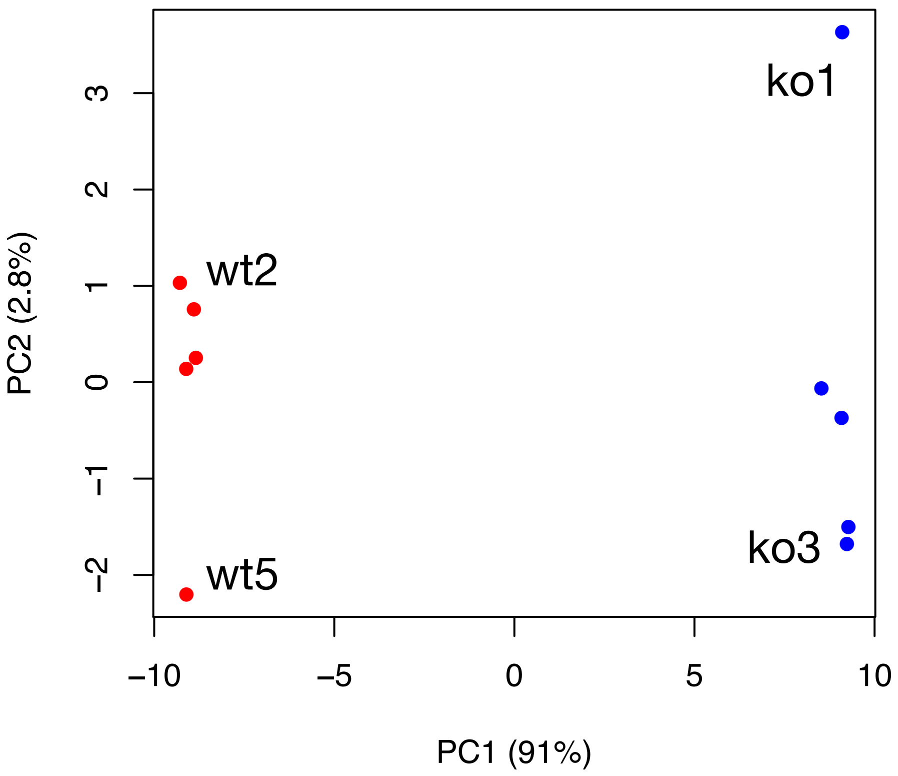
- Lets make our plot a bit more useful…

```r
## A vector of colors for wt and ko samples
colvec <- as.factor( substr( colnames(mydata), 1, 2) )

plot(pca$x[,1], pca$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%)"))
```



- And add some labels…

```
plot(pca$x[,1], pca$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%)"))

## IN THE CONSOLE! Click to identify which sample is which
identify(pca$x[,1], pca$x[,2], labels=colnames(mydata))

## Press ESC to exit…
```



Do it Yourself!

Your turn!

Perform a PCA on the UK foods dataset

https://bioboot.github.io/bggn213_W19/class-material/lab-8-bggn213.html

Input: read, View/head,
PCA: prcomp,
Plots: PCA plot
scree plot,
loadings plot.

[Muddy Point Feedback Link]

- Lets look at how to use the loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot
    - The prcomp() function calls loading scores $rotation

```
## Lets focus on PC1 as it accounts for > 90% of variance
loading_scores <- pca$rotation[,1]
```

- Finally, lets look at how to use the loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot
  - The prcomp() function calls loading scores $rotation

```
## Lets focus on PC1 as it accounts for > 90% of variance
loading_scores <- pca$rotation[,1]

summary(loading_scores)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.104763 -0.104276 -0.068784 -0.005656  0.103926  0.104797

## We are interested in the magnitudes of both plus
##   and minus contributing genes
gene_scores <- abs(loading_scores)
```

- Finally, lets look at how to use the loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot
    - The prcomp() function calls loading scores $rotation

```
loading_scores <- pca$rotation[,1]

gene_scores <- abs(loading_scores)

## Sort by magnitudes from high to low
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)
```

- Finally, lets look at how to use the loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot
    - The prcomp() function calls loading scores $rotation

```
loading_scores <- pca$rotation[,1]

gene_scores <- abs(loading_scores)

## Sort by magnitudes from high to low
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## Find the names of the top 5 genes
top_5_genes <- names(gene_score_ranked[1:5])
```

- Finally, lets look at how to use the loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot
  - The prcomp() function calls loading scores $rotation

```
loading_scores <- pca$rotation[,1]

gene_scores <- abs(loading_scores)

## Sort by magnitudes from high to low
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## Find the names of the top 5 genes
top_5_genes <- names(gene_score_ranked[1:5])

## Show the scores (with +/- sign)
pca$rotation[top_5_genes,1]
```

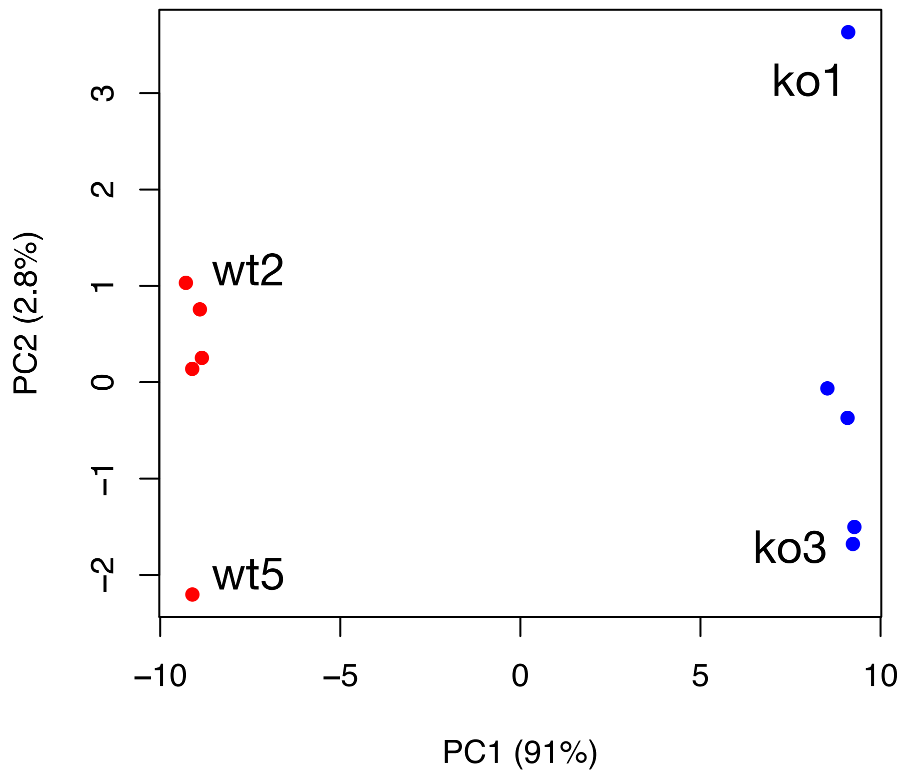
- Here we see genes with the largest positive loading scores that effectively ‘push’ the “ko” samples to the right positive side of the plot.
- And the genes with high negative scores that push “wt” samples to the left side of the plot.

```
loading_scores <- pca$rotation[,1]

gene_scores <- abs(loading_scores)

## Sort by magnitudes from high to low
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)

## Find the names of the top 5 genes
top_5_genes <- names(gene_score_ranked[1:5])

## Show the scores (with +/- sign)
pca$rotation[top_5_genes,1]
    gene64     gene39      gene7     gene60     gene65
 0.1047968  0.1047629 -0.1047629  0.1047601 -0.1047443
```

- Here we see genes with the largest positive loading scores that effectively 'push' the "ko" samples to the right positive side of the plot.
- And the genes with high negative scores that push "wt" samples to the left side of the plot.

```
pca$rotation[top_5_genes,1]
```

```
    gene64     gene39
 0.1047968  0.1047629

     gene7     gene65
-0.1047629 -0.1047443
```



PCA Summary

- PCA is classic "multivariate statistical technique" used to reduce the dimensionality of a complex data set to a more manageable number (typically 2D or 3D)
- For a matrix of *m* genes x *n* samples, we mean center (i.e. subtract the sample mean from each sample column), optionally rescale the values for each sample column, then calculate a new covariance matrix of size *n* x *n*
- We finally diagonalize the covariance matrix to yield our *n* Eigenvectors (called principal components or PCs) and *n* Eigenvalues.
- The top PCs (with largest Eigenvalues) retain the essential features of the original data and represent a useful subspace for further analysis (e.g. visualization, clustering, feature extraction, outlier detection etc…)

Main PCA objectives include:

- To reduce dimensionality
- To visualize multidimensional data
- To choose the most useful variables (features)
- To identify groupings of objects (e.g. genes/samples)
- To identify outliers

[Muddy Point Feedback Link]

Reference Slides

Practical issues with PCA

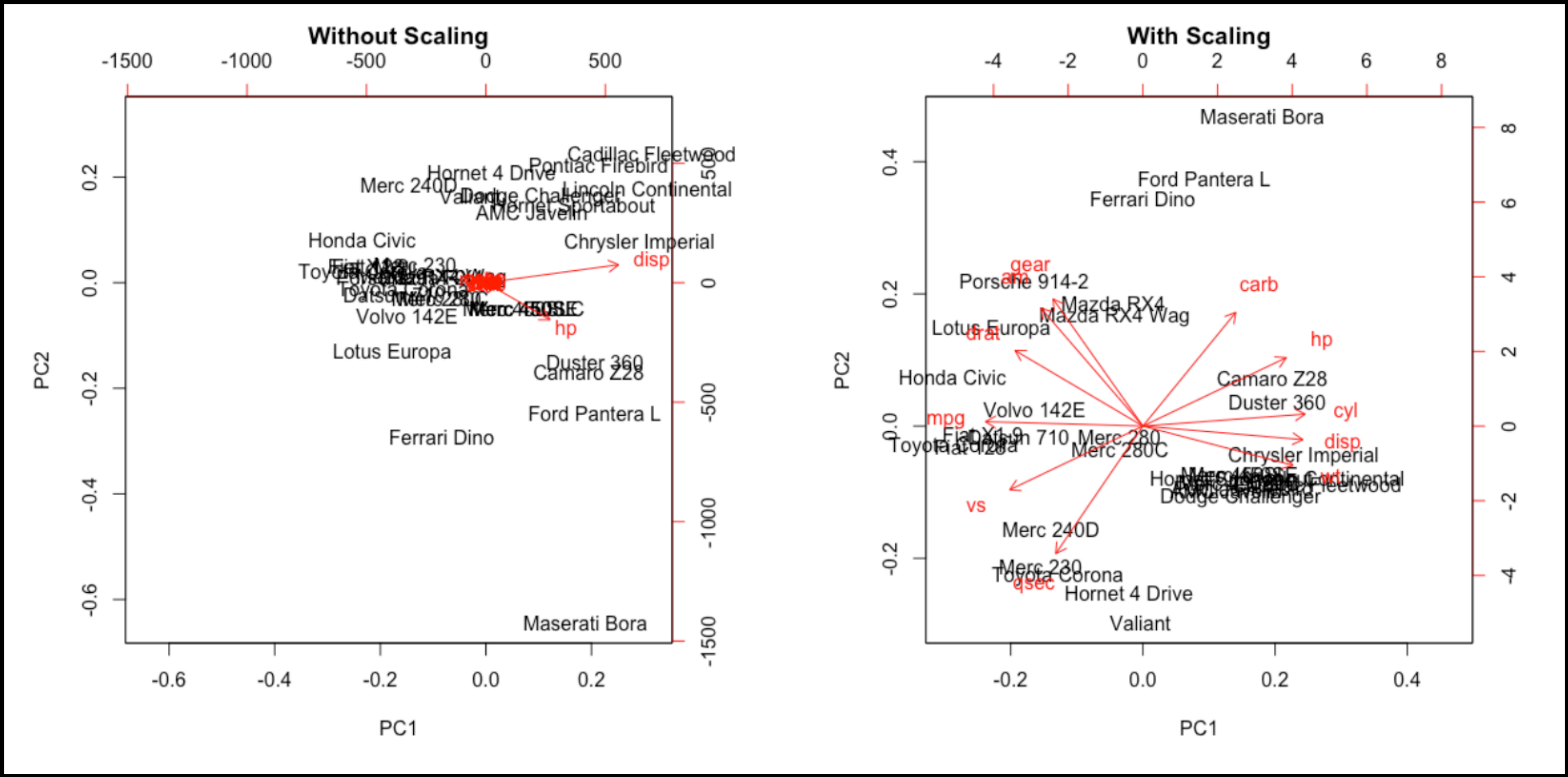
- Scaling the data
- Missing values:

Scaling

```
> data(mtcars)
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

# Means and standard deviations vary a lot
> round(colMeans(mtcars), 2)
   mpg    cyl   disp     hp   drat     wt   qsec     vs     am   gear   carb
 20.09   6.19 230.72 146.69   3.60   3.22  17.85   0.44   0.41   3.69   2.81
> round(apply(mtcars, 2, sd), 2)
   mpg    cyl   disp     hp   drat     wt   qsec     vs     am   gear   carb
  6.03   1.79 123.94  68.56   0.53   0.98   1.79   0.50   0.50   0.74   1.62
```

Scaling

```
prcomp(x, center=TRUE, scale=FALSE)
prcomp(x, center=TRUE, scale=TRUE)
```



Practical issues with PCA

- Scaling the data
- Missing values:
  - ➡ **Drop observations with missing values**
  - ➡ Impute / estimate missing values