Exploring relationships among biological variables and observations

Jeffrey Chang and Philip Moos

6/21/2018

Introduction

In the last presentation, we learned that an unsupervised analysis can be used to explore large data sets (with many genes and/or many samples). Every data set has its own unique personality, and whenever I am confronted with a new one, I always do an unsupervised analysis first to get to know it. These initial analyses will oftentimes reveal idiosyncracies in the data (such as *batch effects*) that may confound the interpretation of the results.

Today, we will learn how to cluster gene expression data visualize it with heatmaps. These are basic, yet powerful, tools that every genomics researcher should have in their toolbox. They can be used to tease out the strongest signals in the data, which can tell you something about the biology driving the disease.

Loading the data

We will apply unsupervised methods to the TCGA (The Cancer Genome Atlas) breast cancer data set. We have retrieved, cleaned up, and saved this data as R files. We begin by loading the previously saved data into R variables that we can work with [1].

```
load(file.path(data_dir, "tcga_brca_expr_norm_df.RData"))
load(file.path(data_dir, "tcga_brca_clinical_df.RData"))
load(file.path(data_dir, "tcga_brca_cdr_clinical_df.RData"))

brca_expr_norm_df.orig <- brca_expr_norm_df
brca_clinical_df.orig <- brca_clinical_df
brca_expr_norm_df.orig <- brca_expr_norm_df
```

Getting to know your data

Let's take an initial look at the data that we loaded. These data are stored in R data frames, which are kind of like matrices in that they have rows and columns. To see how big the data is (how many rows and columns there are), we'll check the dimensions with the `dim` function, that we talked about last week.

```
dim(brca_expr_norm_df)
```

```
## [1] 18351  1084
```

```
dim(brca_clinical_df)
```

```
## [1] 1083   45
```

```
dim(brca_cdr_clinical_df)
```

```
## [1] 1082   22
```

That's a lot of data! Each of the data frames has at least a thousand rows or columns.

To get a peek into how the data are organized, we can look names of the columns. Since we saw that these data frames are rather big, I'm just going to look at the names of the first 5 columns. That should be enough to give us a flavor of how the data is organized.

```
names(brca_expr_norm_df)[1:5]
```

```
## [1] "gene_id"      "TCGA-4H-AAAK" "TCGA-A1-A0SM" "TCGA-A2-A0CO"
## [5] "TCGA-A2-A0CQ"
```

```
names(brca_clinical_df)[1:5]
```

```
## [1] "bcr_patient_uuid"    "bcr_patient_barcode" "acronym"
## [4] "gender"              "vital_status"
```

```
names(brca_cdr_clinical_df)[1:5]
```

```
## [1] "bcr_patient_barcode"
## [2] "type"
## [3] "age_at_initial_pathologic_diagnosis"
## [4] "gender"
## [5] "race"
```

This reveals that the first column of `brca_expr_norm_df` contains the `gene_id` , while the remaining columns contain data for the samples. For the other two matrices, the first column `bcr_patient_barcode` contains the name of the samples, while the remaining columns contain clinical covariates.

Let's take a quick look at the actual data (rather than just the column names). These data frames are really big, and we can't possibly look at the whole thing, we'll just look at the first 5 rows and columns.

```
brca_expr_norm_df[1:5, 1:5]
```

	gene_id <chr>	TCGA-4H-AAAK <dbl>	TCGA-A1-A0SM <dbl>	TCGA-A2-A0CO <dbl>	TCGA-A2-A0CQ <dbl>
1	ENSG00000215529	2.9787	0.3157	0.8355	0.3573
2	ENSG00000178605	705.5320	496.2120	804.1780	713.1760
3	ENSG00000172771	53.6170	26.5152	114.4650	176.1500
4	ENSG00000121410	268.8810	307.4940	304.6520	129.7970
5	ENSG00000148584	0.4255	0.0000	0.0000	0.0000

5 rows

```
brca_clinical_df[1:5, 1:5]
```

	bcr_patient_uuid <chr>	bcr_patient_barcode <chr>	acron… <chr>	gen… <chr>	vital_sta <chr>
1	6E7D5EC6-A469-467C-B748-237353C23416	TCGA-3C-AAAU	BRCA	FEMALE	Alive
2	55262FCB-1B01-4480-B322-36570430C917	TCGA-3C-AALI	BRCA	FEMALE	Alive
3	427D0648-3F77-4FFC-B52C-89855426D647	TCGA-3C-AALJ	BRCA	FEMALE	Alive
4	C31900A4-5DCD-4022-97AC-638E86E889E4	TCGA-3C-AALK	BRCA	FEMALE	Alive
5	6623FC5E-00BE-4476-967A-CBD55F676EA6	TCGA-4H-AAAK	BRCA	FEMALE	Alive

5 rows

```
brca_cdr_clinical_df[1:5, 1:5]
```

	bcr_patient_barcode <chr>	t… <chr>	age_at_initial_pathologic_diagnosis <dbl>	gen… <chr>	race <chr>
1	TCGA-3C-AAAU	BRCA	55	FEMALE	WHITE
2	TCGA-3C-AALI	BRCA	50	FEMALE	BLACK OR AF
3	TCGA-3C-AALJ	BRCA	62	FEMALE	BLACK OR AF
4	TCGA-3C-AALK	BRCA	52	FEMALE	BLACK OR AF
5	TCGA-4H-AAAK	BRCA	50	FEMALE	WHITE

5 rows

By now, you should have a good idea of what is in each of these data frames. brca_expr_norm_df contains the gene expression values (where each row is a gene and each column contains the samples (except for the first)), while the other two data frames contain clinical covariates [2].

Cleaning the data

Unfortunately, we aren't quite ready to start analyzing the data yet. There is still some housekeeping to do. First, let's make sure each of these data frames contain data from the same samples.

```
x1 <- names(brca_expr_norm_df)[2:ncol(brca_expr_norm_df)]
x2 <- brca_clinical_df[["bcr_patient_barcode"]]
x3 <- brca_cdr_clinical_df[["bcr_patient_barcode"]]
sample.names <- intersect(intersect(x1, x2), x3)
sample.names <- sort(sample.names)
if(is.null(sample.names)) stop("No common samples")
if(length(sample.names) < 1000) stop("low number of common samples")
print(sample.names[1:10])
```

```
##  [1] "TCGA-3C-AAAU" "TCGA-3C-AALI" "TCGA-3C-AALJ" "TCGA-3C-AALK"
##  [5] "TCGA-4H-AAAK" "TCGA-5L-AAT0" "TCGA-5T-A9QA" "TCGA-A1-A0SB"
##  [9] "TCGA-A1-A0SD" "TCGA-A1-A0SE"
```

As we noted above, the column names of the `brca_expr_norm_df` data frame are the sample names (except for the first column, which contains a gene ID). In contrast, in the `brca_clinical_df` and `brca_cdr_clinical_df` data frames, each row is a sample. The names of the samples were retrieved by indexing with the `brc_patient_barcode` column name.

The code above makes a list of the samples that are common to each of the matrices. It does some basic checks to make sure we found some common sample names. Once we found the common ones, we'll re-organize each of the data frames to make sure 1) they contain the exact same samples, no more, no less, and 2) the samples are in the same order. This will save us a lot of trouble later.

```r
I <- match(sample.names, names(brca_expr_norm_df))
if(any(is.na(I))) stop("Missing samples")
if(min(I) < 2) stop("missing gene id column")
I <- c(1, I)
brca_expr_norm_df <- brca_expr_norm_df[,I]

I <- match(sample.names, brca_clinical_df[["bcr_patient_barcode"]])
if(any(is.na(I))) stop("Missing samples")
brca_clinical_df <- brca_clinical_df[I,]
if(!all(sample.names == brca_clinical_df[["bcr_patient_barcode"]]))
  stop("Not aligned")

I <- match(sample.names, brca_cdr_clinical_df[["bcr_patient_barcode"]])
if(any(is.na(I))) stop("Missing samples")
brca_cdr_clinical_df <- brca_cdr_clinical_df[I,]
if(!all(sample.names == brca_cdr_clinical_df[["bcr_patient_barcode"]]))
  stop("Not aligned")
```

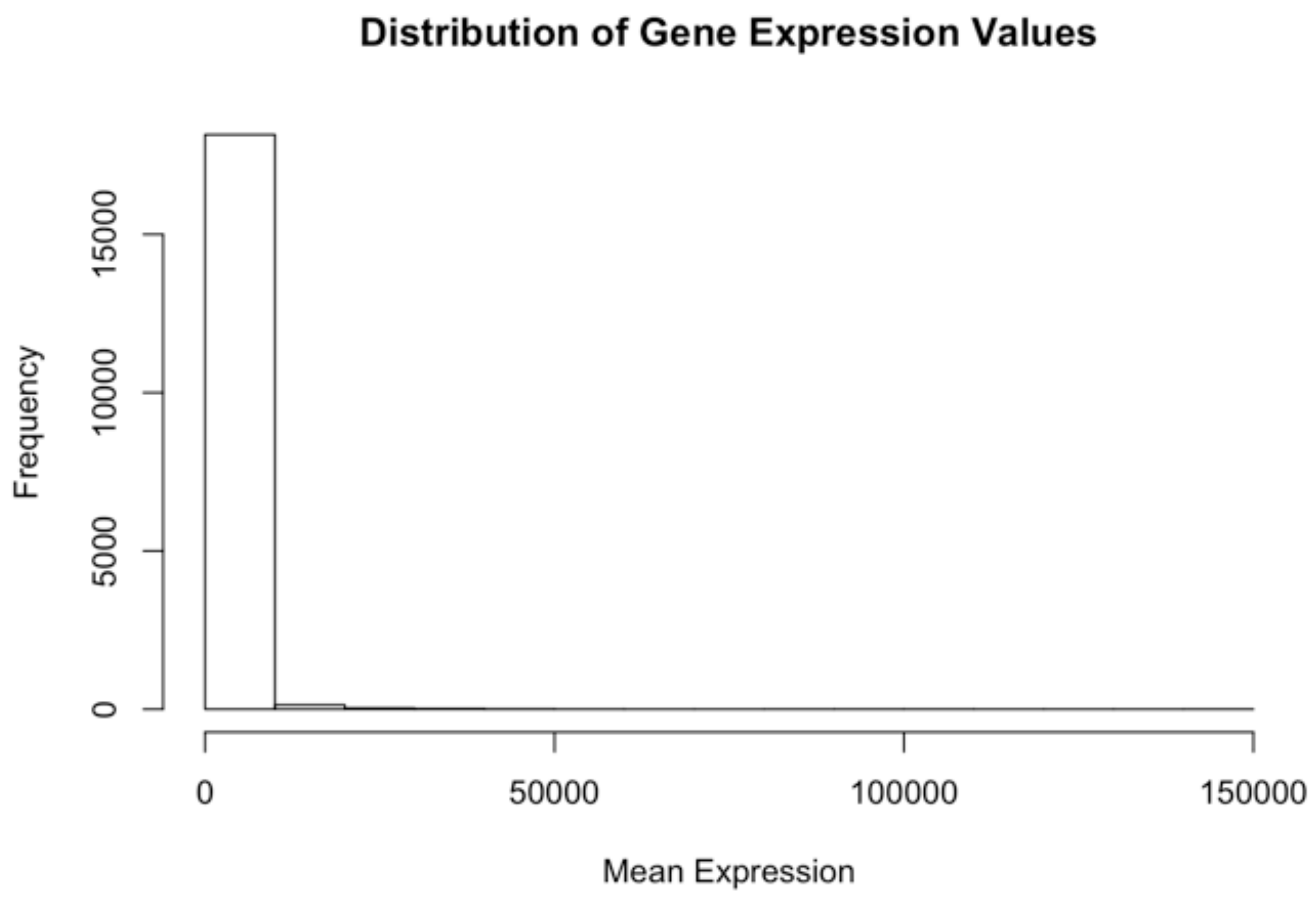
Finally, we'll pull out just the gene expression values from the `brca_expr_norm_df` data frame. It contains the gene names in the first column, and the gene expression values in the following columns. Let's get rid of the gene names, and then convert the gene expression values to a matrix, which is more compatible with mathematical operations.

```r
gene.name <- brca_expr_norm_df[,1]
X <- as.matrix(brca_expr_norm_df[,2:ncol(brca_expr_norm_df)])
```

Genes: Average expression values

OK! We've got the data loaded, cleaned, and ready to be analyzed. What should we look at first? We can start by looking at the average expression level of the genes. In other words, are the genes expressed at a low level or a high level? We'll determine this by making a histogram of the gene expression.

```r
mean.expr <- apply(X, 1, mean)
hist(mean.expr, main="Distribution of Gene Expression Values",
  xlab="Mean Expression")
```



Woah! Looking at the X-axis, we see the expression values go from 0 to 150,000. However, most of the values are close to 0, and there's almost nothing anywhere else. We can confirm that there are actually large gene expression values by checking:

```
print(sprintf("Num values > 100,000: %g", sum(X > 100000)))
```

```
## [1] "Num values > 100,000: 4638"
```

```
print(sprintf("Total values: %d", length(X)))
```

```
## [1] "Total values: 19855782"
```

So there are some really big numbers, but they comprise only a small portion of the matrix. In other words, this distribution has a long right tail. Most genes have low-ish expression, while a small minority of the genes have very high expression. Unfortunately, many statistical tests do not work well with this kind of distribution. Instead, it is very common to assume a Gaussian or normal distribution [3].

Because gene expression data has a long right tail, it is common to transform these to a more normal-ish distribution by doing a log transformation, that is, taking the log of each expression value [4]. To make sure it worked, we'll take a look at the resulting data.

```
X.log <- log(X, 2)
print(X[1:5,1:5])
```

```
##      TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## [1,]       0.3447       4.8940       2.7199       2.0687       2.9787
## [2,]     458.3960     683.5240     657.2980     786.5120     705.5320
## [3,]      40.6697      29.9076      22.6655      54.1994      53.6170
## [4,]     197.0900     237.3840     423.2370     191.0180     268.8810
## [5,]       0.0000       0.0000       0.9066       0.0000       0.4255
```

```
print(X.log[1:5,1:5])
```

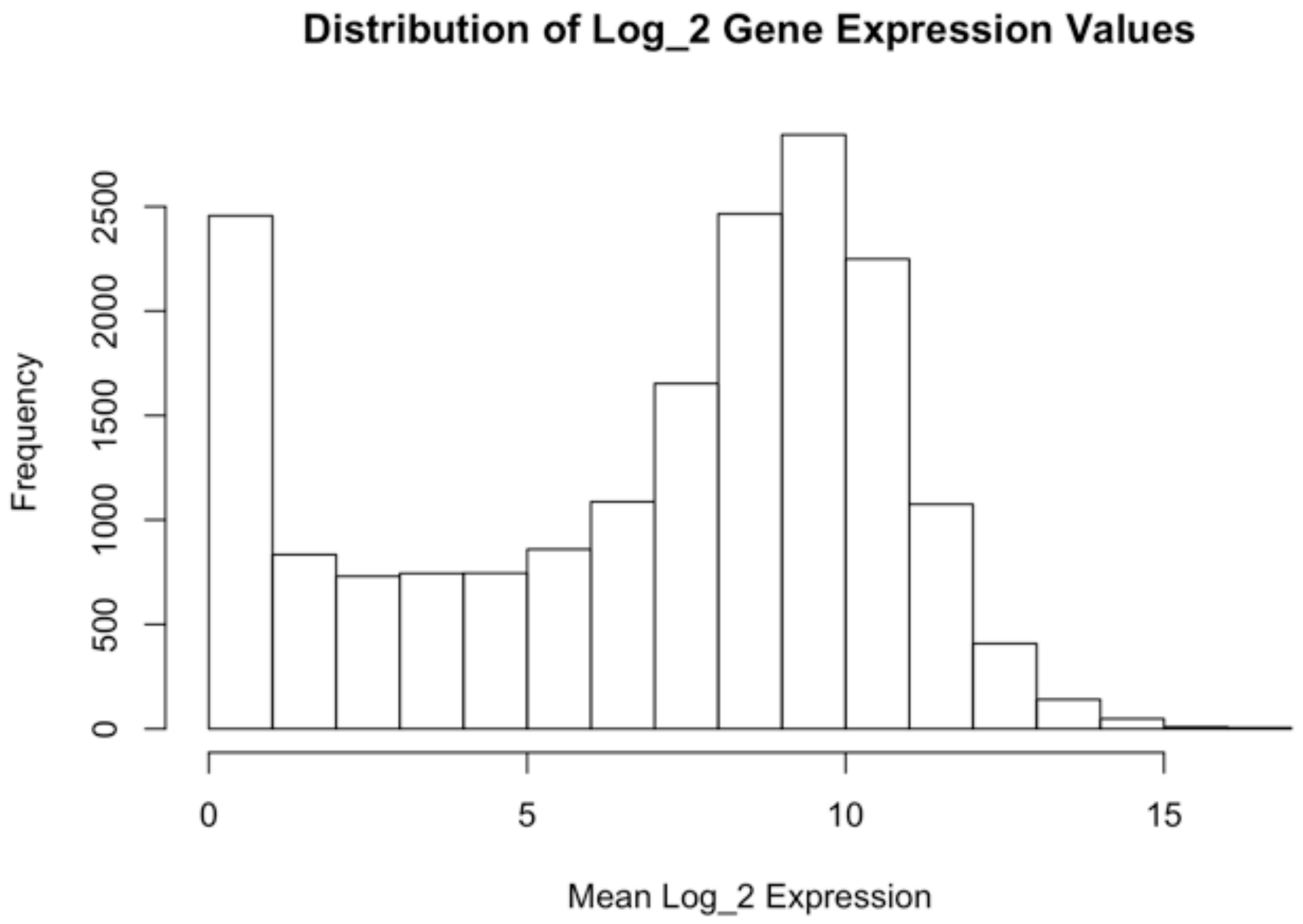
```
##      TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## [1,]    -1.536587     2.291014    1.4435536     1.048724     1.574683
## [2,]     8.840451     9.416848    9.3604038     9.619325     9.462568
## [3,]     5.345882     4.902440    4.5024261     5.760205     5.744619
## [4,]     7.622711     7.891079    8.7253219     7.577565     8.070824
## [5,]         -Inf         -Inf   -0.1414619         -Inf    -1.232769
```

Hey, what are those -Inf values? This happens everywhere the original value is 0, because the log of 0 is -infinity. To avoid this problem, we can add a *pseudo-count* of 1. If we do this, the minimum expression value will be 1, which can be logged without generating infinite numbers.

```
X.log <- log(X+1, 2)
```

Now, taking another look at the distribution:

```
mean.expr <- apply(X.log, 1, mean)
hist(mean.expr, main="Distribution of Log_2 Gene Expression Values",
  xlab="Mean Log_2 Expression")
```

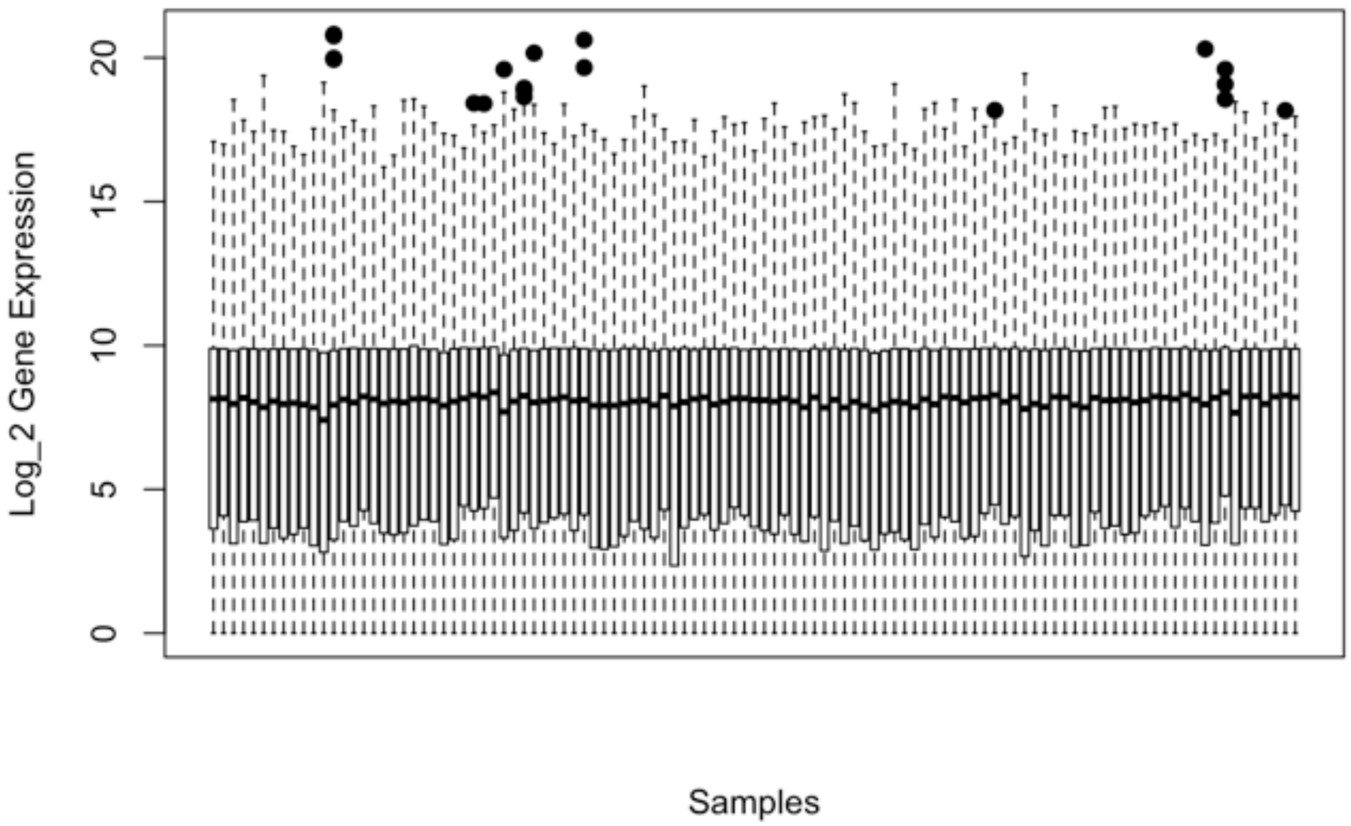


This isn’t quite normal (especially at the lower range), but it is much closer, and importantly, close enough for some exploratory analyses.

Samples: Distributions of expression values

In addition to the genes, we can also look at the distribution of the expression values of the samples. This is commonly done using a boxplot.

```
I <- seq(1, ncol(X.log), 10)
boxplot(X.log[,I], xaxt="n", xlab="Samples", ylab="Log_2 Gene Expression",
  pch=19, cex=1)
```



Here, each of the columns is a sample, and the Y-axis shows the distribution of the expression of genes in each sample. (Here, for simplicity, I've only plotted 1 out of every 10 samples, as indicated in the `I` variable.) When we look at these plots, we hope to see a similar distribution across all samples. While the expression of any individual gene will vary from sample to sample, as a whole, the distribution should be similar. If we see differences (i.e. one set of samples have much different expression values than another), we will need to investigate why that is so, whether it is due to a difference in the biology (e.g. different tissues can exhibit different expression), or a technical artifact (rather common, unfortunately). Here, the data looks pretty clean. You can also try plotting out the entire data set to see.

Making a heatmap

Let's start by taking a look at what the data looks like using a heatmap. We won't be able to make a heatmap of the entire data set–there's just too many genes.

```
print(dim(X.log))
```

```
## [1] 18351  1082
```

To reduce the number of genes, we will try to select the *most important* ones for an exploratory analysis. There are many different ways to do this, and your choice here can lead to profoundly different views of the data. One common approach to select genes in an unbiased way is to find the ones with the *biggest differences* across the samples, or in statistics language, the ones with the highest variance [5]. Here, we'll find the 500 genes with the highest variance.

```
NUM.GENES <- 500
v <- apply(X, 1, var)
O <- order(v, decreasing=TRUE)
X.log.sub <- X.log[O[1:NUM.GENES],]
```

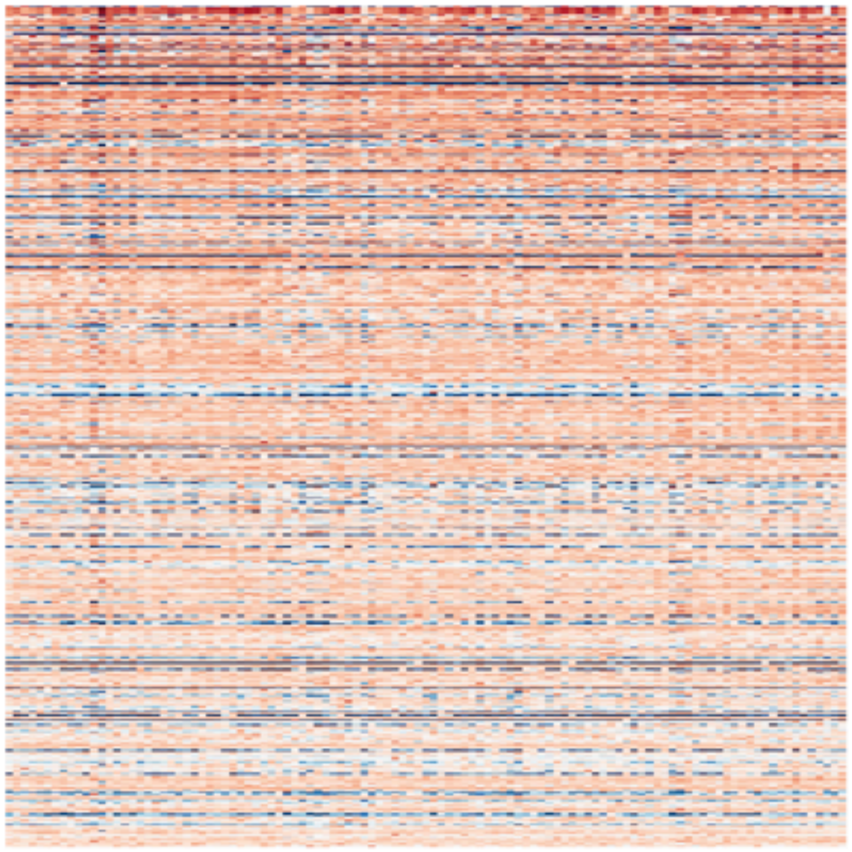
Now let's draw the heatmap. Because the default colors in R are horrible, I'm going to add some code first to generate a nicer looking color palette [6].

```
matrix2color <- function(cmatrix, pos) {
  # pos is [0, 1].  Returns r, g, b where each one is from [0, 1].
  if(is.nan(pos))  # this can happen if someone calls matlab.colors(1)
    pos <- 0.5
  breaks <- cmatrix[,1]
  i1 <- sum(pos >= breaks)
  x <- cmatrix[i1,2:4]
  if(i1 < nrow(cmatrix)) {
    i2 <- i1 + 1
    delta <- (pos - cmatrix[i1,1]) / (cmatrix[i2,1]-cmatrix[i1,1])
    x <- cmatrix[i1,2:4] + delta*(cmatrix[i2,2:4]-cmatrix[i1,2:4])
  }
  x/255
}

matrix2rgb <- function(cmatrix, pos) {
  # pos is [0, 1].  Returns color, e.g. "#003300".
  x <- matrix2color(cmatrix, pos)
  rgb(x[1], x[2], x[3], maxColorValue=1)
}

brewer.rdbu.div.colors <- function(n) {
  if(n <= 0) stop("n must be greater than 0")
  cmatrix <- t(matrix(c(
    c(0.00, 6, 46, 95),
    c(0.10, 33, 102, 173),
    c(0.20, 67, 147, 197),
    c(0.30, 146, 198, 223),
    c(0.40, 210, 230, 240),
    c(0.50, 248, 248, 248),
    c(0.60, 253, 220, 200),
    c(0.70, 245, 165, 130),
    c(0.80, 215, 96, 78),
    c(0.90, 177, 20, 42),
    c(1.00, 103, 0, 28)),
    nrow=4))
  sapply((0:(n-1))/(n-1), function(x) matrix2rgb(cmatrix, x))
}
```

The code to generate the heatmap is below. The R heatmap function [7] plots the matrix from the bottom-up (the last row in the matrix is plotted at the top), which is the opposite of what I want. I want the first row of the matrix to be at the top of the heatmap. Thus, I will first reverse the rows of the matrix.

Often, when I'm doing exploratory analysis, I'll work with just a subset of the data. It makes things go faster, and the plots are easier to interpret. To pull out 10% of the samples, let's store in the `I.sample` vector the indexes of every 10th sample. We'll use this subset for now, but for the final analysis, we'll certainly want to use the whole data set.

```
I.sample <- seq(1, ncol(X), 10)
rev <- X.log.sub[nrow(X.log.sub):1, I.sample]
heatmap(
  rev, Rowv=NA, Colv=NA, scale="none", labRow="",
  labCol="", col=brewer.rdbu.div.colors(64))
```



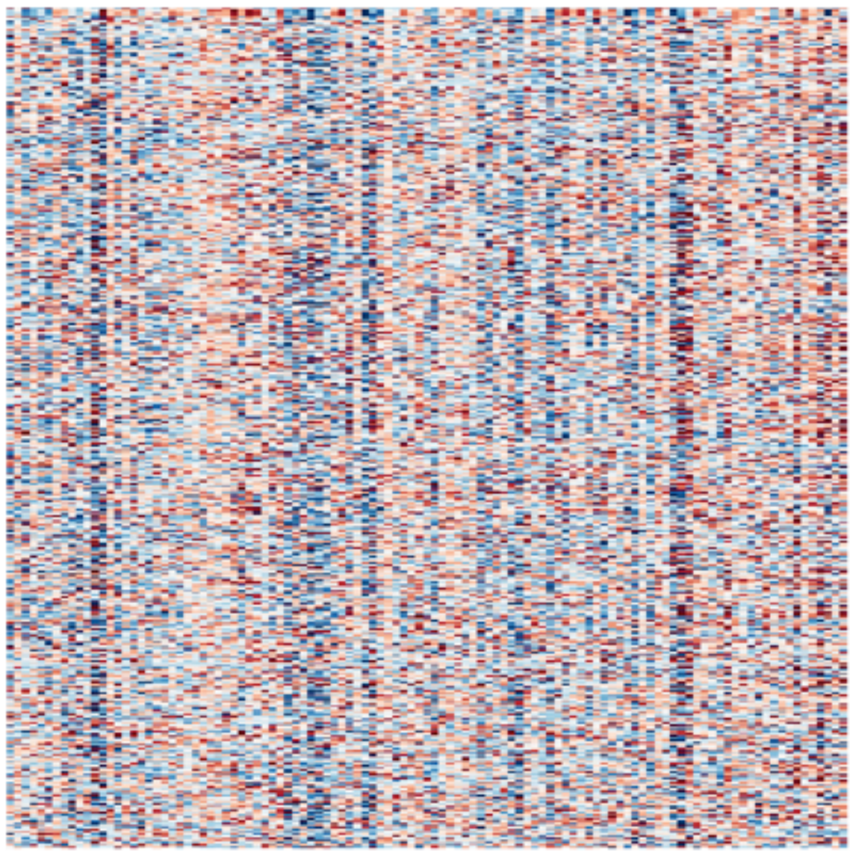
Each row is a gene, and each column is a sample. It doesn't quite look right, though. There are a bunch of horizontal stripes. This happens because some of the genes are expressed higher than others. While this is interesting, what we really want to see are the patterns of expression across the samples. In other words, we want to know, for each gene, whether it is higher in one group of samples versus the other. So we are more interested in the relative expression of the genes, rather than the absolute expression.

To get the relative gene expression, we will first normalize each of the genes. A common way to do this is to change each gene such that the mean expression is 0, and the variance is 1. The gene expression values normalized this way can be interpreted as z-scores, or the number of standard deviations arount the mean. The absolute gene expression values will be changed, but the relative expression (whether it is higher or lower in a particular sample) will be preserved.

```
x <- X.log.sub
means <- apply(x, 1, mean)
x <- sweep(x, 1, means)
x <- t(apply(x, 1, function(x) {
  V.0 <- var(x); M.0 <- mean(x); (x-M.0)*sqrt(1/V.0) + M.0 }))
X.norm <- x
```

And now, let’s plot the normalized gene expression. We’ll limit the values to a range of -2 and +2 standard deviations so that the outliers do not skew the colors.

```
rev <- X.norm[nrow(X.norm):1, I.sample]
rev <- pmax(pmin(rev, 2), -2)
heatmap(
  rev, Rowv=NA, Colv=NA, scale="none", labRow="",
  labCol="", col=brewer.rdbu.div.colors(64), zlim=c(-2, 2))
```



This heatmap reveals a lot of variation in the gene expression, but it’s hard to see the patterns this way. Thus, we need to do some clustering.

Clustering reveals patterns in the data

Here, we will do hierarchical agglomerative clustering (or just hierarchical clustering). As we discussed before, there are a number of distance metrics, or ways to calculate the distance between two data points. Some common ones that might be appropriate for gene expression data are: pearson, euclidean, manhattan, minkowski

You can get a description of them in R using:

```
?dist
```

Also, for hierarchical clustering, there are different ways to combine data points into clusters: average, complete, single

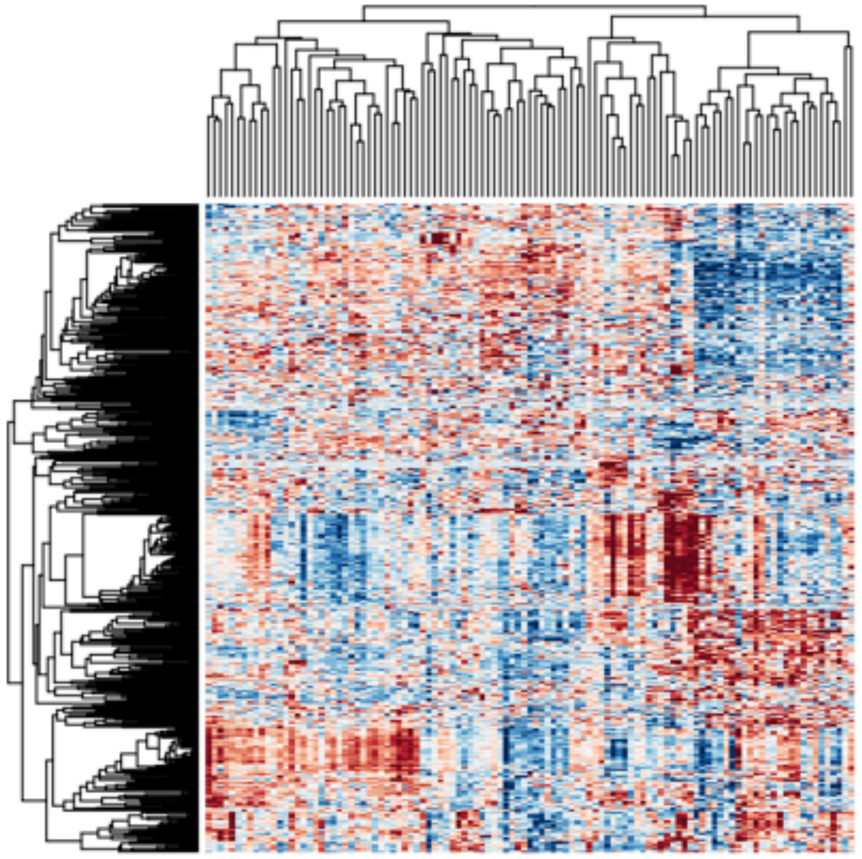
See:

```
?hclust
```

```
dist.method <- "pearson"
clust.method <- "average"

rev <- X.norm[nrow(X.norm):1, I.sample]
if(dist.method == "pearson") {
  row.dist <- as.dist(1-cor(t(rev), method=dist.method))
  col.dist <- as.dist(1-cor(rev, method=dist.method))
} else {
  row.dist <- dist(rev, method=dist.method)
  col.dist <- dist(t(rev), method=dist.method)
}
rc <- hclust(row.dist, method=clust.method)
cc <- hclust(col.dist, method=clust.method)
```

Now, we will generate the clustered heatmaps.

```
rev <- pmax(pmin(rev, 2), -2)
heatmap(
  rev, Rowv=as.dendrogram(rc), Colv=as.dendrogram(cc), scale="none",
  labRow="", labCol="", col=brewer.rdbu.div.colors(64), zlim=c(-2, 2))
```



Clusters and the underlying biology

From this heatmap, we can see that the tumors split up into different groups. Is there any biological significance to these clusters? Are these groups only seen in the gene expression patterns, or is it related to any known biology?

As you previously learned, breast cancer is split up into ER+ and ER- tumors, depending on whether the tumor expresses high levels of receptors for estrogen. This is really important to know because it tells you what's driving the disease, and also how to treat it. To see whether ER status is associated with these clusters, let's label the samples with a "+" for ER+ breast cancers, and "." for negative. We'll leave blank the tumors that are missing this information [8].

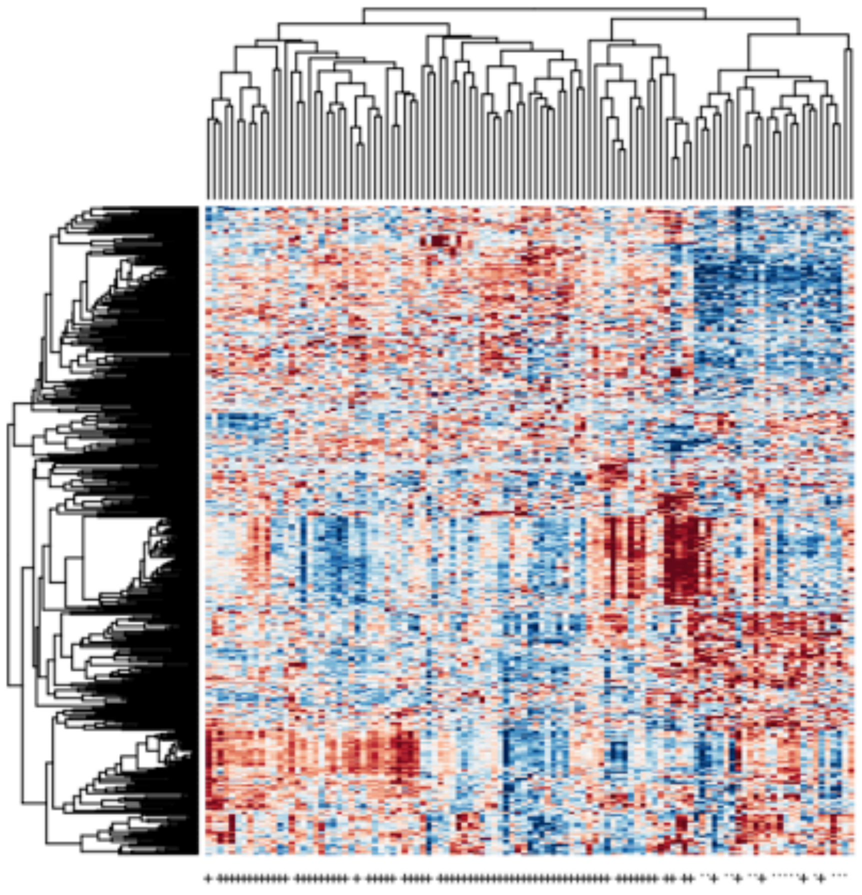
```
clin <- brca_clinical_df[I.sample,]
if(!all(clin[["bcr_patient_barcode"]] == colnames(rev))) stop("unaligned")
if(!all(clin[["bcr_patient_barcode"]] == cc$labels)) stop("unaligned")

x <- clin[["breast_carcinoma_estrogen_receptor_status"]]
print(sort(unique(x)))
```

```
## [1] "[Not Evaluated]" "Negative"        "Positive"
```

```
er <- rep("", length(x))
er[x == "Positive"] <- "+"
er[x == "Negative"] <- "."
print(sum(er == ""))
```

```
## [1] 6
```

```
heatmap(
  rev, Rowv=as.dendrogram(rc), Colv=as.dendrogram(cc), scale="none",
  labRow="", labCol=er, col=brewer.rdbu.div.colors(64),
  zlim=c(-2, 2))
```



In this plot, I see about four clusters of samples (although we can debate about this). Three of them have a lot of ER+ tumors (marked by the row of pluses on the bottom), while the fourth (on the very right) is nearly all ER-. It's not a perfect split, however. Let's calculate the p-value to determine the statistical significance. I'm not going to go through the statistics in detail, but briefly, we will split these samples into 4 clusters. Then, we will set up a 2-way contingency table and calculate the statistical significance with a chi-square test.

```
NUM.CLUSTERS <- 4
if(!all(clin[["bcr_patient_barcode"]] == cc$labels)) stop("unaligned")
cluster <- cutree(cc, k=NUM.CLUSTERS)

outcome <- "breast_carcinoma_estrogen_receptor_status"
values <- sort(unique(clin[[outcome]]))
uniq.clust <- sort(unique(cluster))
counts <- matrix(0, nrow=length(values), ncol=length(uniq.clust))
for(i in 1:length(values)) {
  for(j in 1:length(uniq.clust)) {
    I1 <- clin[[outcome]] == values[i]
    I2 <- cluster == uniq.clust[j]
    counts[i, j] <- sum(I1&I2)
  }
}
print(chisq.test(counts))
```

```
## Warning in chisq.test(counts): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  counts
## X-squared = 58.276, df = 6, p-value = 1.007e-10
```

In fact, the association between ER status and cluster is statistically significant [9].

---

t-SNE: T-distributed Stochastic Neighbor Embedding

Another method to look at the data is *t-SNE*. This method has recently become popular because it works well with single cell data. It is a way to organize high dimensional data (the tumor samples here are high dimensional because each gene is a dimension) into low dimensions (i.e. into a two-dimensional scatterplot) in a way that best preserves the distances seen in high dimension. While we're not looking at single cell data here, it works well on bulk expression data too.

A t-SNE analysis has one (major) parameter that you must specify, the perplexity. It determines the size of the neighborhood the algorithm uses when finding clusters. The way to fit this parameter is just to try different methods and see what the clustering looks like. Usually a good perplexity is within 2-50, and I've found that around 15-25 usually works well for single cell expression data. You can try playing around with this to see how it affects the plot.

Let's see how we can use t-SNE to cluster the data set above. We'll use the entire data set here (rather than the 10% used above) since high numbers of tumor samples are easy to visualize in a scatterplot. We use Pearson correlations to find the distance between data points.

This should run for a minute or two.

```
library(tsne)

PERPLEXITY <- 25
D <- as.dist(1-cor(X.norm, method="pearson"))
coords <- tsne(D, perplexity=PERPLEXITY, max_iter=500)
```

```
## sigma summary: Min. : 0.137888770509641 |1st Qu. : 0.189562449986705 |Median : 0.2
03783682204654 |Mean : 0.206686302960807 |3rd Qu. : 0.220642275977256 |Max. : 0.29615
9058725407 |
```

```
## Epoch: Iteration #100 error is: 17.143648105191
```

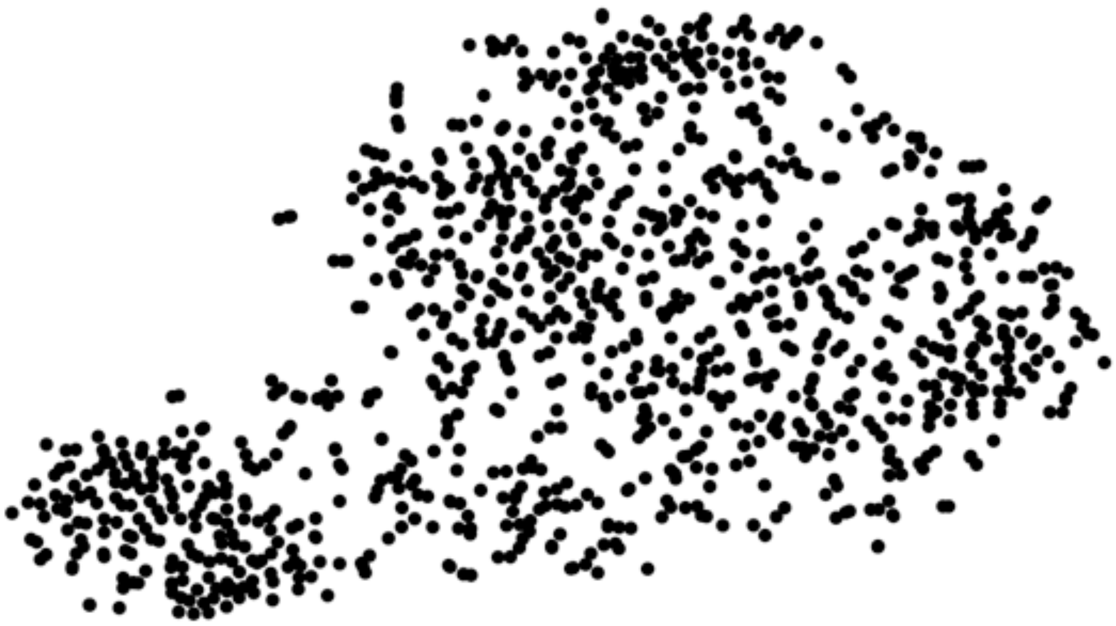
```
## Epoch: Iteration #200 error is: 1.47309072005143
```

```
## Epoch: Iteration #300 error is: 1.38586325923263
```

```
## Epoch: Iteration #400 error is: 1.3578165703913
```

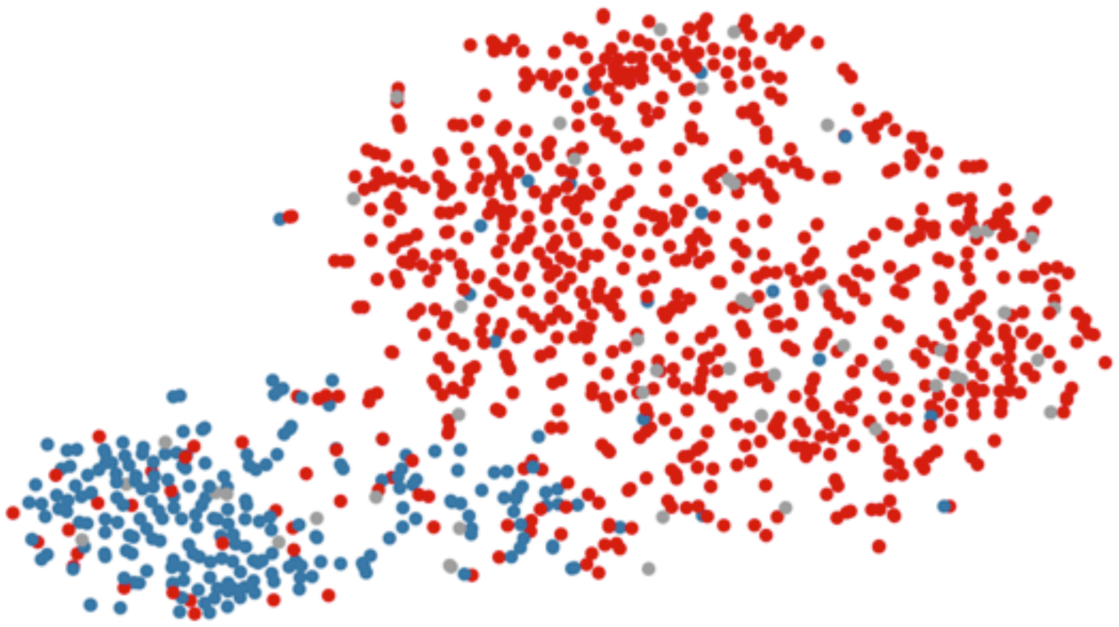
```
## Epoch: Iteration #500 error is: 1.34873730721994
```

```
plot(coords[,1], coords[,2], pch=19, cex=0.75, xlab="", ylab="", axes=FALSE)
```



Here, each point is a tumor sample that is clustered according to their gene expression profiles. Let's see whether the t-SNE clusters here are associated with ER status. We'll color [10] each point according to the ER status. ER+ will be red, and ER- will be blue.

```
clin <- brca_clinical_df
if(!all(clin[["bcr_patient_barcode"]] == colnames(X.norm))) stop("unaligned")

x <- clin[["breast_carcinoma_estrogen_receptor_status"]]
col <- rep("#A0A0A0", length(x))
col[x == "Positive"] <- "#D80713"   # Red
col[x == "Negative"] <- "#3979A8"   # Blue
if(length(col) != nrow(coords)) stop("misaligned")

plot(coords[,1], coords[,2], pch=19, cex=0.75, xlab="", ylab="",
     axes=FALSE, col=col)
```



You can see that the ER positive (red) and negative (blue) tumors are separated relatively cleanly. Thus, the ER status is a signal that is robust across at least two different methods of clustering.

---

mini-DREAM Challenge

QUESTION: How does the choice of distance metric or clustering method affect the association of ER status and the clusters? What is the best set of parameters (distance metric, clustering method, and number of clusters) and p-value you can find?

To answer this question, you can re-run the code under the section *Clustering reveals patterns in the data*. You can try different values for `dist.method`, including `pearson`, `euclidean`, or `manhattan`. For `clust.method`, you can try `average`, `complete`, or `single`. Run the chunks and see what the clustered heatmap looks like.

Finally, there's a chunk that calculates the p-value using a chi-squared test. In that chunk, you may need to change the `NUM.CLUSTERS` parameter based on the number of clusters you wish to break the samples into. There's not a hard and fast rule for how to choose this. You can just look at the data to see how many you think naturally falls out.

Please document your results below.

```
my_distance_metric <- ""
my_cluster_method <- ""
my_num_clusters <- 0
my_p_value <- 1
```

Submitting the prediction

Please run the chunk below to submit the prediction. Make note of the submission ID, and check the mini-DREAM scoreboards to see the results.

```
#library(synapser)
#synLogin(rememberMe = TRUE, silent = TRUE)
#submission <- submit_module_answers(module = 1)
```

Thought questions:

1. A significant problem in science is *overfitting the data*. This is a common phenomenon where a computational method may work really well on the data set that it was developed on, but then doesn't work as well (or at all) on other data sets. This happens because computational methods will pick up on both signals that are reproducible biology as well as signals that are idiosyncrasies to a particular data set (e.g. confounding factors, technical variation). As a method that is calibrated on a specific data set (like we did here), it will become more finely tuned to pick up on features unique to this data set. The magnitude of the problem depends on the relative intensity of the biological signal to the noise. When developing computational methods, you should always be aware that overfitting is occurring. What steps can you take to mitigate the problem, or at least measure how big of an issue it is?
2. In your exploration of different distance metrics or clustering methods, did you see the association with many different parameters, or just a specific one? Is the result robust?
3. What is a p-value? If a p-value cutoff of 5% means that there's a 5% chance that you would see this association randomly, how many significant p-values would you expect to see if you did 100 experiments on random data (e.g. tested 100 clinical covariates, tested 100 sets of parameters, etc.)? About how many tests did you do to answer the mini-DREAM Challenge? What does this mean in terms of the number of false positives you expect to see?
4. Why might an ER+ tumor be clustered with ER- tumors based on gene expression?
5. Are there other clinical covariates that associate with your clusters? (To answer this question, you will need to do more computational analysis to test each covariate in the annotation files). What are the p-values of their association?
6. If another clinical covariate is correlated with ER status, what does that mean biologically? Which one causes the other? Or could they both be the result of another process? How would you distinguish these possibilities?

The most important take-away

The most important thing I would like you to take from this, is to **always look at your data**. As a computational scientist, it is easy to become complacent and place an undue amount of faith in the power of the formulas that we've spent so much time developing. However, the methods are only as good as the underlying data, and biology has a limitless ability to surprise. So, don't fail to spend time looking at your data. What does the underlying distribution look like? Are there unexpected patterns in it that are difficult to explain? Unsupervised methods can help you to reveal confounding factors (very frequent), or discover new biology (sometimes).

Have fun exploring your data!

---

1. Pro tip: loading data files can be slow if the data is really big. When I'm working with the data, I'll first make a *backup* copy of the variable so that if I mess up the data, I can restore it without having to reload the data file.

   Later on, if I need to restore the original data, I will just run these lines:

   ```
   brca_expr_norm_df <- brca_expr_norm_df.orig
   brca_clinical_df <- brca_clinical_df.orig
   brca_expr_norm_df <- brca_expr_norm_df.orig
   ```

   ↩
2. You'll note that these data matrices are *tidy*. When analyzing, you should always strive to make your data tidy: (https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf (https://www.jstatsoft.org/article/view/v059i10/v59i10.pdf)).↩
3. Very little biological data is truly normally distributed. However, the statistics can work if the distribution is *close enough*. What constitutes close enough is very difficult to say. The bottom line is that you need to be aware that your data is probably not normally distributed, and that the results from the statistical methods are therefore not going to be precisely accurate. It is best to validate your findings with other methods (e.g. other statistical tests, other data sets, other types of assays).↩
4. Expression data is typically logged using base 2. This makes it easy to look for 2-fold changes, which corresponds to a difference of 1 after logging. A 2-fold change is commonly considered, as a very rough rule of thumb, to be big enough to be interesting.↩
5. Other unbiased approaches may use other statistical measures, such as selecting ones with highest mean expression, highest interquartile range, etc. Or, you may choose knowledge-based approaches, for example, using genes known to be expressed in breast tissue, or deregulated in cancer, etc.↩
6. The color palette here was developed by Cynthia Brewer, and designed such that the perceived differences between the colors correspond to the difference in the underlying data. For more information, see: http://mkweb.bcgsc.ca/brewer/ (http://mkweb.bcgsc.ca/brewer/)↩
7. You can get help in R, for example, to see what each argument in the heatmap function does, by typing `?heatmap`.↩

8. The first few lines of this code are just checks to make sure the samples in the brca_clinical_df data frame, containing the ER status, are in the same order as the gene expression matrix. If they aren't in the same order, the plot will be incorrect. Since this is critically important, you MUST add code to check these things!↩

9. You will get an alarming warning here that the *Chi-squared approximation may be incorrect*. This is happening because some entries in the contingency table are small, and thus difficult to model accurately. We can ignore this warning for now.↩

10. Colors in R are represented as strings in the format `#RRGGBB` , where RR is a two digit hexadecimal value from 00-FF indicating the shade of red (00 is no red, and FF is the most). Green and blue shades are indicated in the GG and BB characters. As an example, `#D80713` is a red-ish color. There is a heavy red shade (D8), while green (07) and blue (13) are low.↩