Module 1: R genomics analysis environment

Alex Lewis and Diana Murray

July 23, 2020

The knitr R package

knitr() is the R package that generates the report from R Markdown. We can create reports as Word doc, PDF, and HTML files.

An R package bundles together code, data, documentation, and tests, and is easy to download and share with others.

About this activity

You will load and examine R dataframe objects that contain data from over 1,000 breast cancer patients from The Cancer Genome Atlas. The objects include: * clinical measurements on the patients and the patients' tumors * gene expression data

The Cancer Genome Atlas or TCGA characterized over 20,000 cancer samples spanning 33 cancer types with genomics.

In 2012, the TCGA Network reported findings of analyses of primary breast cancers in hundreds of patients with multiple experimental platforms (exome sequencing, genomic DNA copy number arrays, DNA methylation, mRNA arrays, microRNA sequencing, and reverse-phase protein arrays.) Here, we will consider the normalized read count expression data for patient-level breast cancer (brca) data from TCGA.

Loading the data

Throughout the course, We will be loading data from R files with the extension `.RData`. An example is the file `TCGA_brca.RData` which we examine below.

`.RData` files are binary files (i.e., not human readable) that can store multiple R objects, such as vectors, lists, matrices, and data frames.

The R objects in `.Rdata` files are the results of previous R analyses, and the data is in **tidy** form. Tidy data sets have consistent structure and are easy to manipulate, visualize, and model.

We could spend the entire course on learning how to put data in tidy form, but our goal is the understand and analyze the data, which occurs *after* tidying the data.

```r
# We simply `load()` TCGA_brca.RData
# The file.path() function tells `load()` where our data lives
# The argument "verbose = TRUE" makes `load()` tell us what R objects are being loaded

load(file.path(data_dir, "TCGA_brca.RData"),verbose=TRUE)
```

```
## Loading objects:
##   brca_expr_norm_names_df
##   brca_gene_df
##   brca_clin_df
```

The objects were named to be be descriptive. * "df" stands for "dataframe" * "brca" stands for "TCGA breast cancer" * "expr" stands for "gene expression data" * "norm" means that the data is normalized to account for variability in the experiments * "names" reminds us that the data has the gene names included * "clin" is short for "clinical"

We looked at `brca_clin_df` last time.
Today we will also consider `brca_expr_norm_names_df`.

The *Environment* tab gives us basic information on the objects that were loaded.

We will use some of the R functions we learned to examine the data in more detail. * `dim()` tells us the dimensions (# row, # columns) of the objects. * `head()` shows us the top several rows of the objects. * indexing allows us to look at the rows and columns we choose.

```
dim(brca_expr_norm_names_df)  # We should get a result consistent
```

```
## [1] 18351  1083
```

```
                              # with what is in the Environment tab
```

Let's see what's in the rows and columns.

```
brca_expr_norm_names_df[1:5,1:5]     # Inspect the first few rows of the data frame
```

```
##     symbol TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK
## 1   TSPAN6     188.1840     207.7220    1005.4400    1104.6800
## 2     TNMD       0.3447       1.0875      39.8912       1.2412
## 3     DPM1     524.2260     809.1350     967.3620     463.3840
## 4    SCYL3     325.1410    1558.9400     335.8300     549.2550
## 5 C1orf112     124.2940     274.6660     241.6860     218.2250
```

The first column, "symbol" is the names of the genes. The remaining columns are the expression levels of the genes from different patient samples.

"TCGA-3C-AAAU" is the anonymized identifier for one patient's sample.

To do analysis with the expression dataframe, we have to remove the "symbol" column so that the matrix contains only the numerical gene expression data. But we want to keep the information from the "symbol" column. R allows us to do this by assigning the symbols as the row names for our matrix.

```
brca_expr_mat <- as.matrix(brca_expr_norm_names_df[,-1])  # We remove the first column,
                                                          # use the function as.matrix().
                                                          # and assign the result to a new object.

# Now, we'll use the function row.names() to keep the gene symbols.
row.names(brca_expr_mat) <- brca_expr_norm_names_df$symbol
```

Let's use indexing to check our the first 5 rows and columns of our matrix.

```
brca_expr_mat[1:5,1:5]
```

```
##          TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## TSPAN6       188.1840     207.7220    1005.4400    1104.6800     942.5530
## TNMD           0.3447       1.0875      39.8912       1.2412       4.6809
## DPM1         524.2260     809.1350     967.3620     463.3840     486.3830
## SCYL3        325.1410    1558.9400     335.8300     549.2550     416.2850
## C1orf112      124.2940      274.6660      241.6860      218.2250      179.4600
```

Let's look at the average value of the expression for the first two genes, TSPAN6 and TNMD.

TSPAN6 codes for the protein Tetraspanin-6. This protein helps signal events that play a role in cell growth and motility.

TNMD codes for the protein Tenomodulin, which is important for the formation of tendons. TNMD is highly expressed in tendons, but lowly expressed in other parts of the body.

```r
# mean() takes the average of a set of values

avg_TSPAN6_expr <- mean(brca_expr_mat[1,])    # We are taking the average of expression values
                                              # for the first row of the matrix

avg_NFYA_expr <- mean(brca_expr_mat[10,])       # We are taking the average of expression values
                                              # for the second row of the matrix


print(paste("Average expression of TSPAN6: ", round(avg_TSPAN6_expr,0)))
```

```
## [1] "Average expression of TSPAN6:  996"
```

```r
print(paste("Average expression of NFYA: ", round(avg_NFYA_expr,0)))
```

```
## [1] "Average expression of NFYA:  1326"
```

```r
# The function paste() allows us to print text and numbers together.
# The function round() lets us choose how many decimal digits we want to show.
```

It would stink to have to do this for all 18,351 genes.

But we can use the function `apply()` to apply a function like `mean()` to all rows or columns of our matrix.

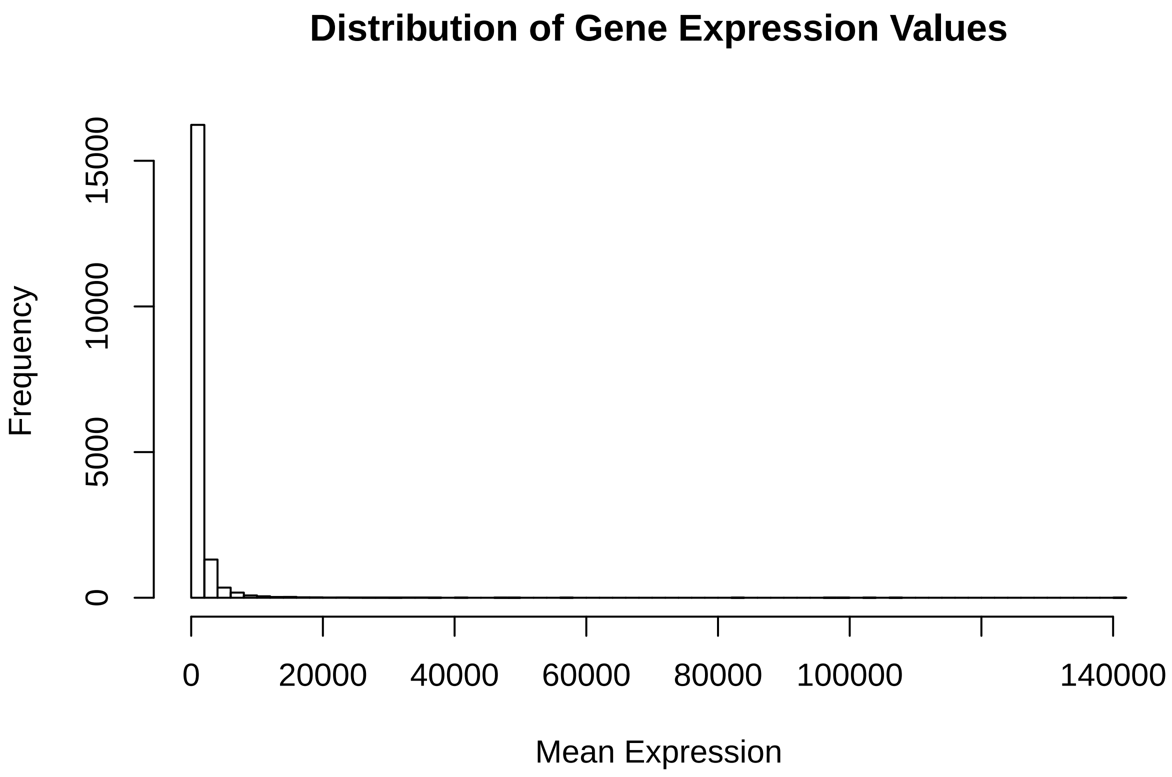
```r
mean.expr <- apply(brca_expr_mat, 1, mean)  # We "apply" the function mean() to all rows.
                                            # The first argument is the matrix.
                                            # The second argument is 1 for "rows".
                                            # The third argument is the function name, `mean()`

as.table(round(mean.expr[1:25],0))
```

```
##   TSPAN6     TNMD     DPM1    SCYL3 C1orf112      FGR      CFH    FUCA2
##      996       24      699      624      320      187     1246     1432
##     GCLC     NFYA    STPG1   NIPAL3    LAS1L    ENPP4   SEMA3F     CFTR
##      869     1326      215      985     1143      642     2180       19
##   ANKIB1  CYP51A1    KRIT1    RAD52      BAD     LAP3     CD99   HS3ST1
##     1614     1945      738      130      853     2646     2751       91
##     AOC1
##       84
```

It's nice we can calculate all the means at once, but it's not very helpful in comparing the mean of the different genes.

Let's try plotting the means as a histogram.

```r
hist(mean.expr, breaks=100, main="Distribution of Gene Expression Values",
  xlab="Mean Expression")
```



```
# A histogram is a graphical display of data using bars of different heights.
# Each bar groups numbers into ranges. Taller bars show that more data falls in that range.
# The first argument is the data, the second is how bars we want to show,
# the third is a title for the plot, and the fourth is the name of the x-axis.
```

Wow! Most of the average expression values are relatively small (less than 200), but there must be a few really large values.

Let's see how many genes have average expression greater than 10,000.

```
print(paste("Total values: ", length(mean.expr)))
```

```
## [1] "Total values:  18351"
```

```
print(paste("Num values > 10000: ", sum(mean.expr > 10000)))
```

```
## [1] "Num values > 10000:  201"
```

```
print(paste("Pencentage of mean values > 10000: ", round(sum(mean.expr>10000)/length(mean.expr)*100,1)))
```

```
## [1] "Pencentage of mean values > 10000:  1.1"
```

So there are a few really big numbers. The curve in the histogram has a long right tail.
Nearly all genes have *low* expression, and very few genes have *high* expression.

Unfortunately, many analysis functions in R do not work well with this kind of distribution.

Instead, it is very common to assume the data follow a normal (or bell) curve.

In practive,very little real life biological data truly follow a normal curve. It has been established that most analysis works quite well if the data are *close enough* to a normal curve.

We can make our data more like normal data by doing a *log transform*. If we take the log of all the epxression values, we put the data more on equally footing.

Expression data is typically logged using base 2.

Log base 2 of 8 is 3 because 2 * 2 * 2 = 8.

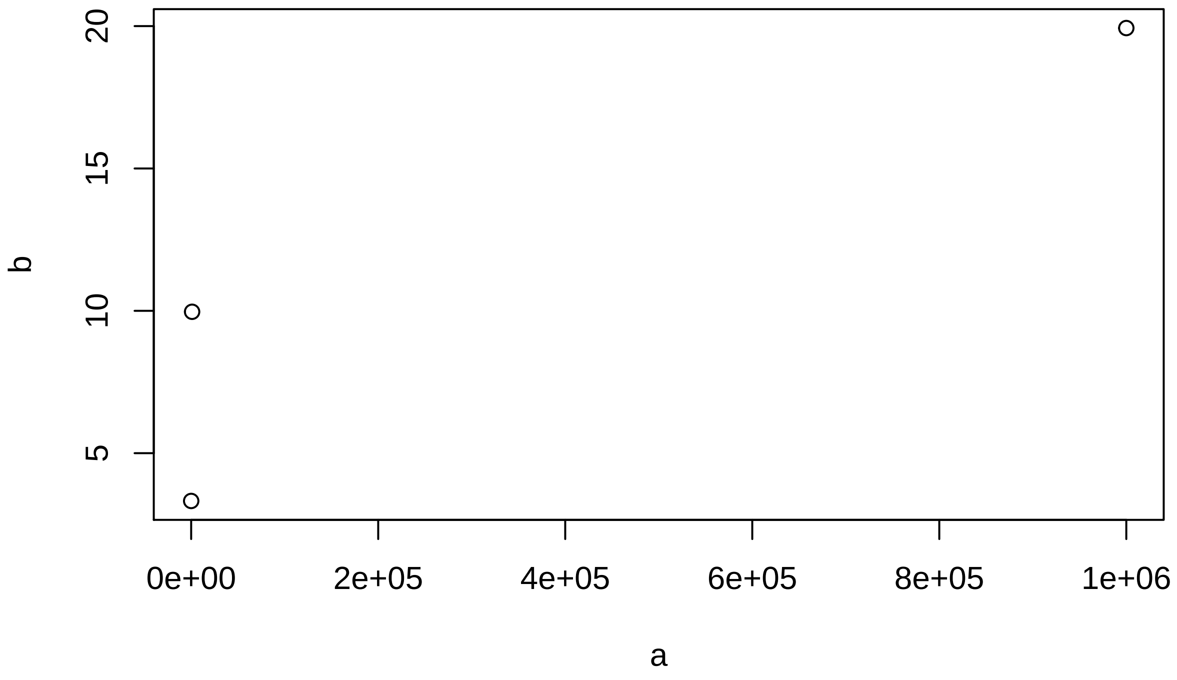
Log base 2 of 4 is 2 because 2 * 2 = 4.

Log base 2 of 2 is 1 because 2 = 2.

So the values (8, 4 , 2) are *transformed* to (3, 2, 1).

Let's consider some larger numbers:

```
a <- c(1000000,1000,10)

b <- log(a,2)

plot(a,b)
```



Notice how the scale for b is much more compressed than for a.

Taking log2 of numbers with a large range make them more comparable.

```
a
```

```
## [1] 1e+06 1e+03 1e+01
```

```
log(a,2)
```

```
## [1] 19.931569  9.965784  3.321928
```

There is one caveat. The log2 of 0 is infinity!

```
log(0,2)
```

```
## [1] -Inf
```

But the log2 of 1 is "well-behaved."

```
log(1,2)
```

```
## [1] 0
```

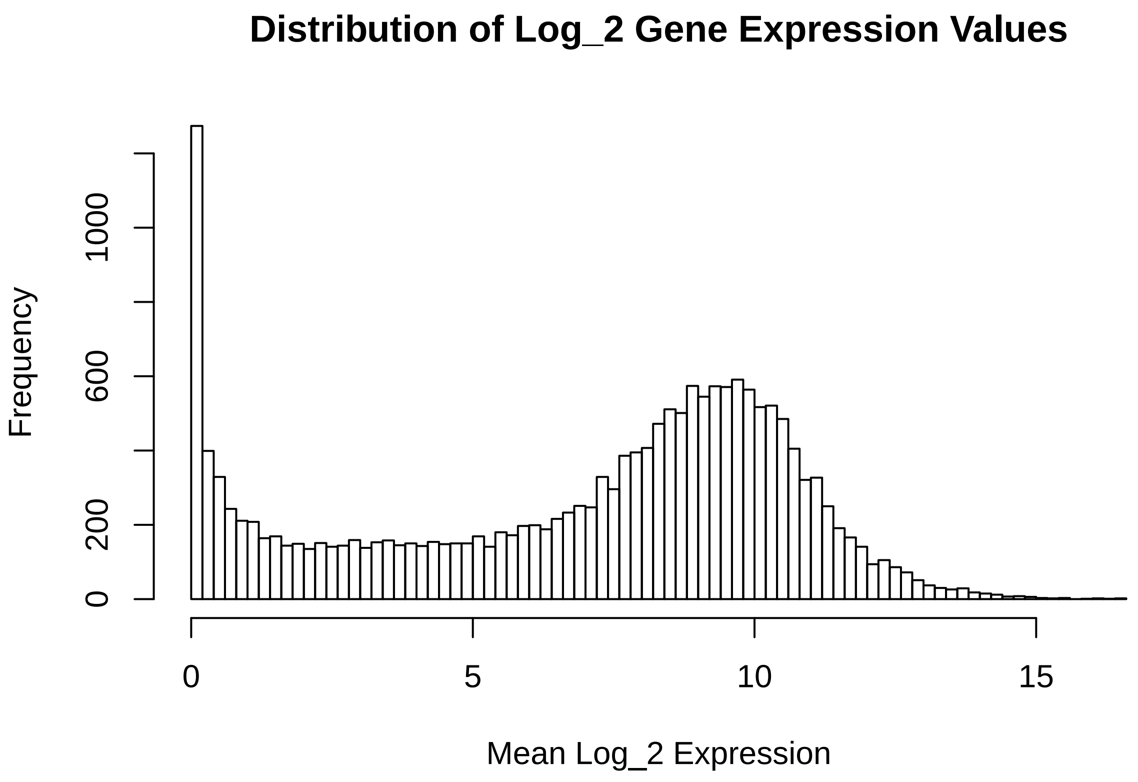
So before taking the log2 of our expression data, we will add 1 to each value to avoid the occurence of "-Inf" values.

```r
brca_expr_mat.log <- log(brca_expr_mat+1, 2)  # We add 1 to each element, then take the log base 2.

#Let's look at some values:
print(brca_expr_mat.log[1:5,1:5])
```

```
##          TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## TSPAN6      7.5636463     7.705439     9.975045    10.110718     9.881960
## TNMD        0.4272843     1.061776     5.353718     1.164271     2.506120
## DPM1        9.0367945     9.662019     9.919403     8.859174     8.928912
## SCYL3       8.3493520    10.607275     8.395877     9.103957     8.704889
## C1orf112    6.9691735     8.106778     7.922947     7.776269     7.495535
```

Let's check out what the curve is for the log-transformed expression values:

```r
mean.expr <- apply(brca_expr_mat.log, 1, mean)  # We "apply" the function mean() to all rows.
                                                # The first argument is the matrix.
                                                # The second argument is 1 for "rows".
                                                # The third argument is the function name, `mean()`

hist(mean.expr, breaks=100, main="Distribution of Log_2 Gene Expression Values",
  xlab="Mean Log_2 Expression")
```



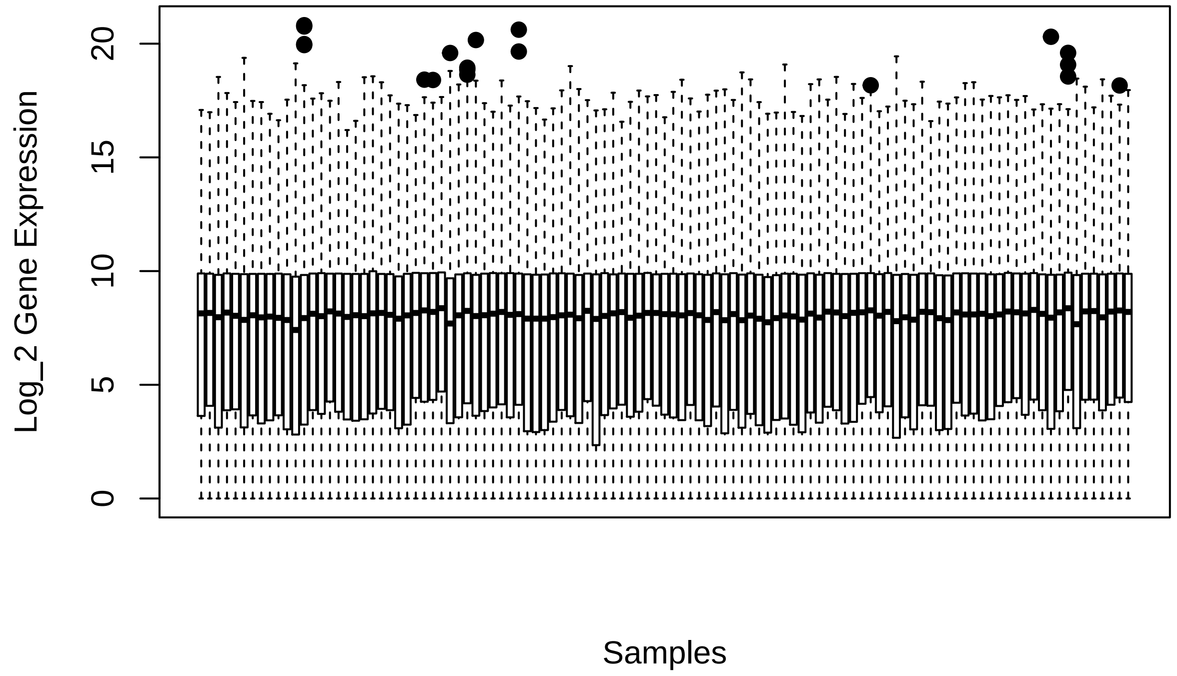
```r
# A histogram is a graphical display of data using bars of different heights.
# Each bar groups numbers into ranges. Taller bars show that more data falls in that range.
# The first argument is the data, the second is how bars we want to show,
# the third is a title for the plot, and the fourth is the name of the x-axis.
```

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.

```r
I <- seq(1, ncol(brca_expr_mat.log), 10)  # We will look at every tenth sample.
boxplot(brca_expr_mat.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.
When we look at these plots, we hope to see a similar distribution of gene expression values 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. some samples have very different expression values), 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 from the experiments (rather common, unfortunately). Here, the data looks pretty clean.

Visualizing the data with heatmaps

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 are just too many genes.

Just to remind ourselves, take another look at the size of the data.

```r
dim(brca_expr_mat.log)
```

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

```r
brca_expr_mat.log[1:5,1:5]
```

```
##          TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## TSPAN6      7.5636463     7.705439     9.975045    10.110718     9.881960
## TNMD        0.4272843     1.061776     5.353718     1.164271     2.506120
## DPM1        9.0367945     9.662019     9.919403     8.859174     8.928912
## SCYL3       8.3493520    10.607275     8.395877     9.103957     8.704889
## C1orf112    6.9691735     8.106778     7.922947     7.776269     7.495535
```

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

We'll consider the 500 genes with the highest variance.

And order out matrix accordingly, so we make a matrix for the top 500 variable genes.

```
NUM.GENES <- 500                                # How many genes we want to consider.
v <- apply(brca_expr_mat, 1, var)     # Here, we use the function apply
                                                # with the function var (for variance).

O <- order(v, decreasing=TRUE)    # The function order will rank the values for us.

var_genes <- v[O]                        # We look at the top ten.


brca.log.sub <- brca_expr_mat.log[O[1:NUM.GENES],]
```

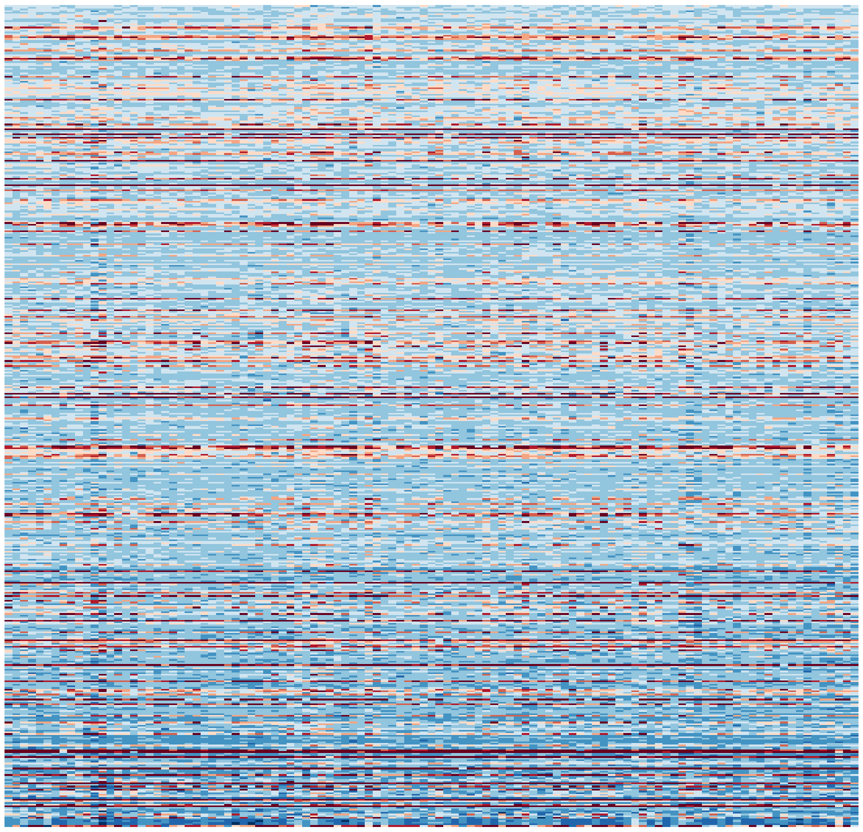
Let's look at it.

```
brca.log.sub[1:5,1:5]
```

```
##        TCGA-3C-AAAU TCGA-3C-AALI TCGA-3C-AALJ TCGA-3C-AALK TCGA-4H-AAAK
## CPB1       16.16135     4.243707     11.10774     6.928501     2.707503
## COL1A1     15.76215    17.451710     17.46421    18.729189    18.659219
## COL1A2     15.08131    16.591093     16.53135    17.853364    18.187168
## FN1        15.49303    16.703795     16.88043    17.056871    17.901097
## COL3A1     14.94762    16.132576     16.08866    17.581760    17.871713
```

When we're doing exploratory analysis, we'll often 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.

```
library("RColorBrewer")              # We call a library that has nice colors

I.sample <- seq(1, ncol(brca.log.sub), 10)      # We take every tenth sample.


heatmap(brca.log.sub[,I.sample],
        Rowv=NA, Colv=NA, scale="none", labRow="",                # These arguments are for plotting
        labCol="", col=brewer.pal(10,"RdBu"), margins = c(1, 0))     # the heatmap.
```



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 absolute gene expression values will be changed, but the relative expression (whether it is higher or lower in a particular sample) will be preserved.

R has an easy way to do this with the function `scale()`.

```
help(scale)
```

Notice that `scale()` will scale the *columns* of our matrix. But we want to scale the genes, which are in the rows!

This is not a problem, because we can use the transpose function `t()` to convert rows to columns and columns to rows.

So we scale the transposed matrix. Then we have to transpose it back!!

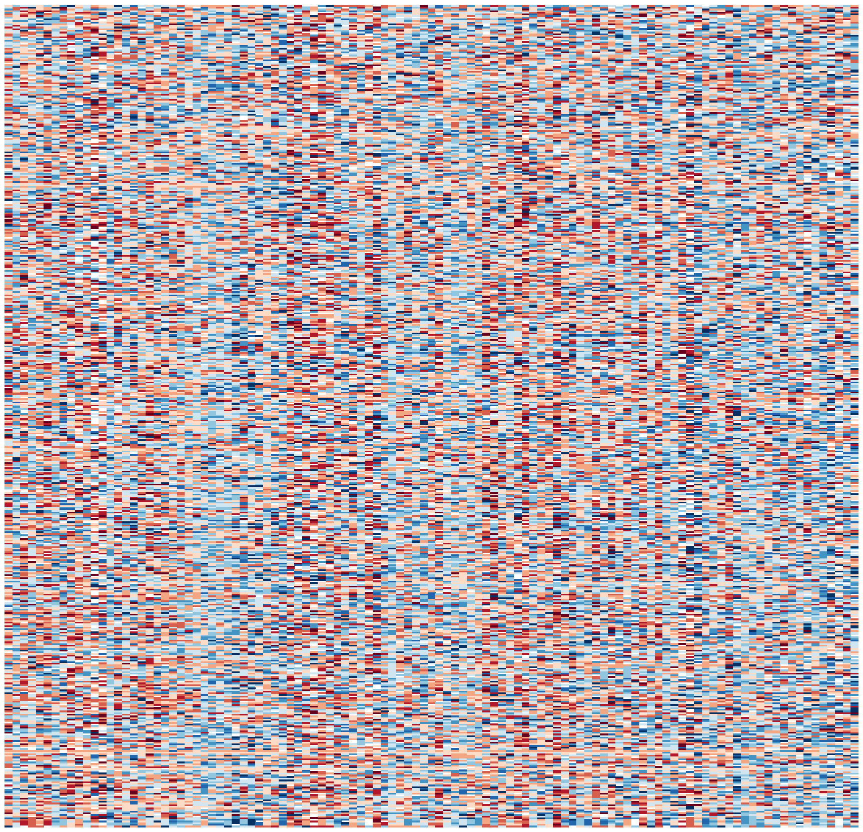
Here, we are using three nested functions: * `t()` to transpose our matrix * `scale()` to scale the transposed matrix * another `t()` to transpose back so that genes are rows again.

```
brca.log.sub.scale <- t(scale(t(brca.log.sub)))
```

What does the heatmap look like now?

```
heatmap(brca.log.sub.scale[,I.sample],
   Rowv=NA, Colv=NA, scale="none", labRow="",      # The rest of the arguments are for plotting.
```

```
    labCol="", col=brewer.pal(10,"RdBu"), zlim=c(-2, 2), margins = c(1, 0))
```



Nice! This is much better. We can more easily see which genes are overexpressed (red) and underexpressed (blue) in a relative manner.

Our next activity will be to organize this matrix so we can see patterns among the * genes (row), and * samples (columns),

and associate the patterns with clinical informations.

Practice

1. Save this file with another name.
2. Work through the activity again and change values such as NUM.GENES and I.sample.
3. How do your choices change what you see?