Data exploration and visualization in R

Code

BGGN-213 Lecture 5:
Barry Grant < http://thegrantlab.org/bggn213/ (http://thegrantlab.org/bggn213/) >
2018-10-12 (09:03:06 PDT on Fri, Oct 12)

Overview

One of the biggest attractions to the R programming language is that built into the language is the functionality required to have complete programmatic control over the plotting of complex figures and graphs. Whilst the use of these functions to generate simple plots is very easy, the generation of more complex multi-layered plots with precise positioning, layout, fonts and colors can be more challenging for new users.

Sections 1 to 4 are exercises designed to test and expand your familiarity with R base plotting. *Section 5* feature a rather detailed guide to plotting with base R and will be useful for answering the questions in the first four sections. My objective with the last section is to provide you with everything you would need to create plots for almost any kind of data you may encounter in the future.

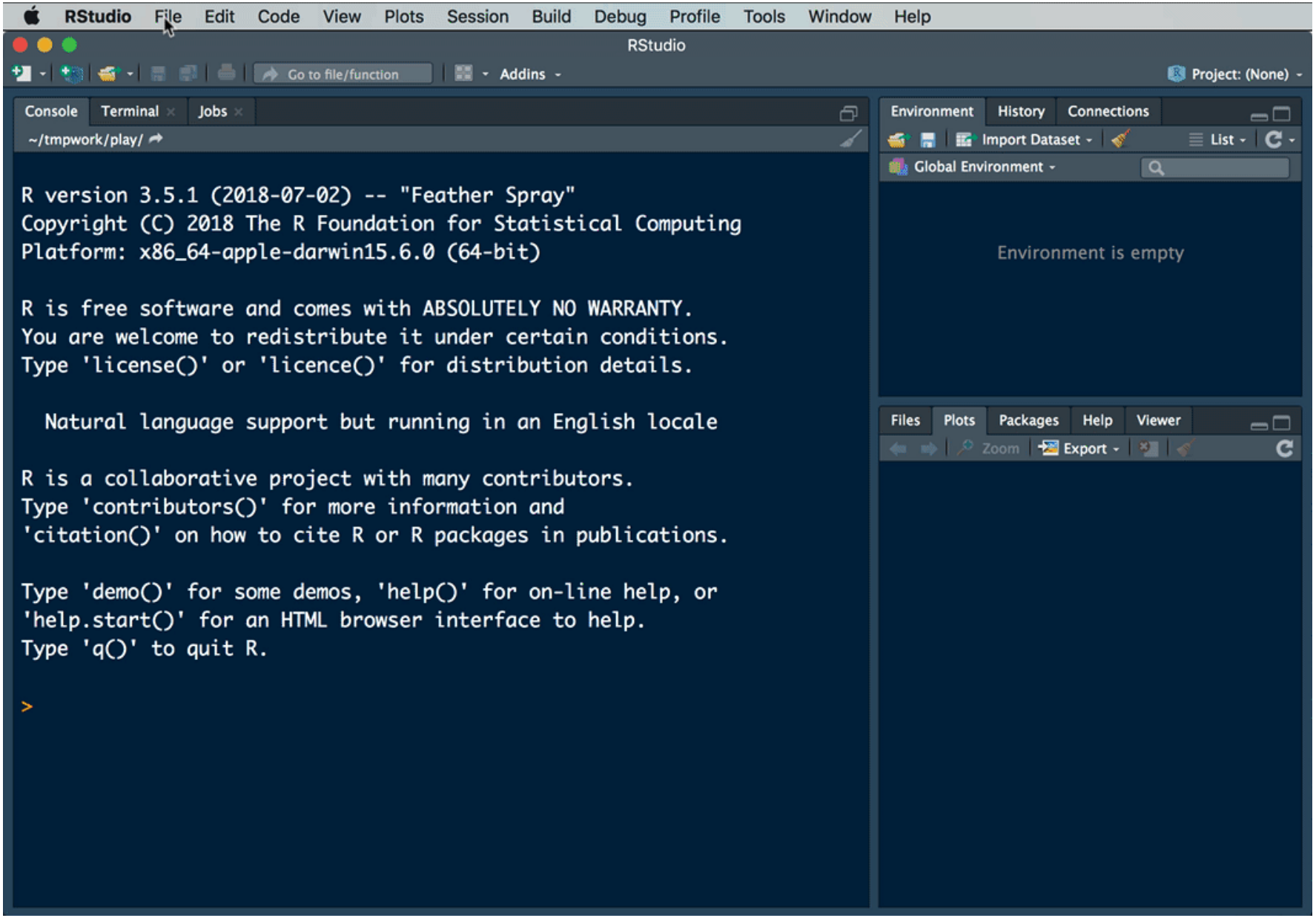
Side-note: While I have attempted to cover all of the major graph types you are likely to use as a newcomer to base R these are obviously not the only way to do plotting in R. There are many other R graphing packages and libraries, which can be imported and each of these has advantages and disadvantages. The most common, and ridiculously popular, external library is **ggplot2**, – we will introduce this add on package from CRAN next day.

Section 1: Getting organized

1A. Creating a Project

1A. We will begin by getting organized. This entails creating a new RStudio *Project*, creating a new R *script*, and downloading the input data we will use For this hands-on session.

- Begin by opening RStudio and creating a new **Project** `File > New Project > New Directory > New Project` make sure you are working in the directory (a.k.a. folder!) where you want to keep all your work for this class organized. For example, for me this is a directory on my Desktop with the class name (see animated figure below). We will create our project as a *sub-directory* called `class05` in this location.

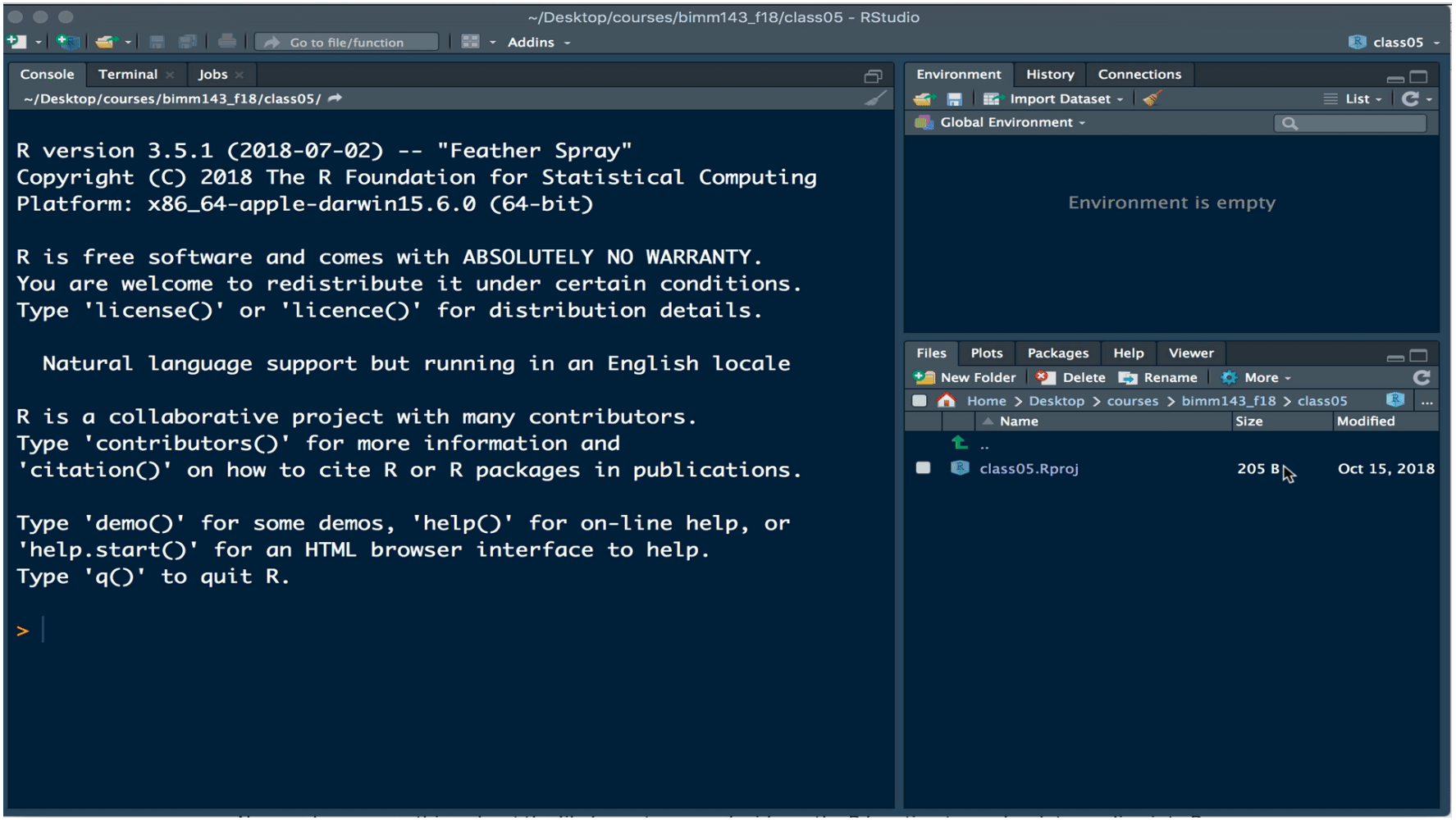


The key step here is to name your project after this class session (i.e. “class05”) and make sure it is a sub-directory of where ever you are organizing all your work for this class (see image below for an example).

1B. Getting data to plot

1B. Your next task is to download the ZIP file (https://bioboot.github.io/bimm143_F18/class-material/bimm143_05_rstats.zip) containing all the input data for this lab to your computer and move it to your RStudio project.

- Download the ZIP file (https://bioboot.github.io/bimm143_F18/class-material/bimm143_05_rstats.zip)
- Note that on most computers this will typically unzip itself after downloading to create a new directory/folder named `bimm143_05_rstats` . Check your *Downloads* directory/folder. However, some computers may require you to double click on the ZIP file to begin the unzipping process.
- Once unzipped move the resulting `bimm143_05_rstats` folder into your R project directory. You can use your Finder window (on Mac) or File Explorer (on Windows) to do this.



Section 2: Customizing plots

2A. Line plot

2A. The file **weight_chart.txt** from the example data you downloaded above contains data for a growth chart for a typical baby over the first 9 months of its life.

Your first task is to get this data into R:

- Find the file `weight_chart.txt` in your RStudio **Files** panel. This should typically be on the right lower side of your RStudio window.
- Click on the `weight_chart.txt` file to get a preview of its contents in RStudio.
- How are the records in the file separated? Is there a comma, space, tab or other character between the data entries? Does the file have a *"Header"* line that contains the names of the variables (this is often the first line of a file)?

Now we know something about the file format we can decide on the R function to use for data reading into R.

- Look at the help page for the `read.table()` function and decide on the function and arguments to use for importing the `weight_chart.txt` file.
- What effect does setting and changing the argument `header=FALSE` to `header=TRUE` in your chosen **read** function?
- Make sure to examine the object you create and store
- Add your working code for reading the file to your R script and make sure to save your file.

Hide

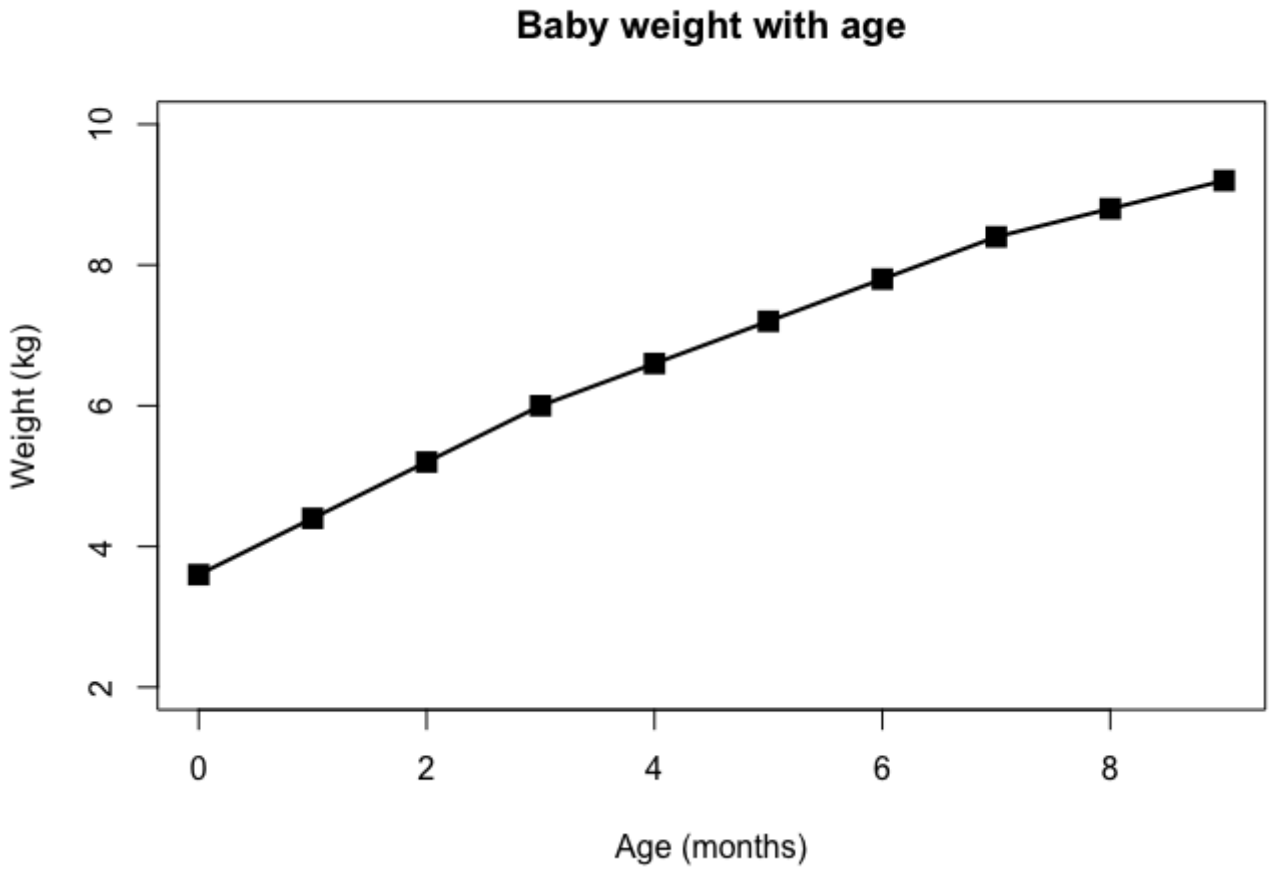
```
weight <- read.table("bimm143_05_rstats/weight_chart.txt", header=TRUE)
```

Use the `plot()` function to draw this as a point and line graph with the following changes:

- Change the point character to be a filled square (`pch=15`)
- Change the plot point size to be 1.5x normal size (`cex=1.5`)
- Change the line thickness to be twice the default size (`lwd=2`)
- Change the y-axis to scale between 2 and 10kg (`ylim=c(2,10)`)
- Change the x-axis title to be Age (months) (`xlab="Age (months)"`)
- Change the y-axis title to be Weight (kg) (`ylab="Weight (kg)"`)
- Add a suitable title to the top of the plot (`main="Some title"`)

Hide

```
plot(weight$Age, weight$Weight, typ="o",
     pch=15, cex=1.5, lwd=2, ylim=c(2,10),
     xlab="Age (months)", ylab="Weight (kg)",
     main="Baby weight with age")
```



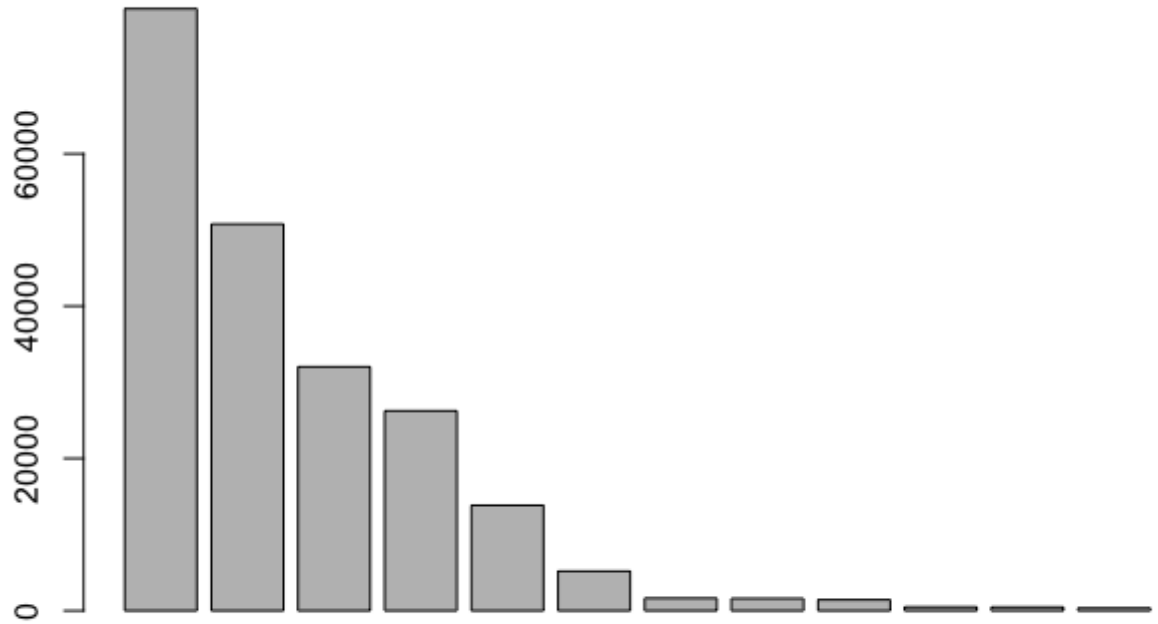
2B. Barplot

2B. The file **feature_counts.txt** contains a summary of the number of features of different types in the mouse GRCm38 genome.

- Read this data into R using the same overall procedure you used in section A above (i.e. check the format of the file to decide on the read function and arguments to use).
- What is the field separator and header format of this file?
- Can you plot this as a **barplot**. Note that the data you need to plot is contained within the **Count** column of the data frame, so you pass only that column as the data.

Hide

```
mouse <- read.table("bimm143_05_rstats/feature_counts.txt", sep="\t", header=TRUE)

barplot(mouse$Count)
```



Once you have the basic plot above make the following changes:

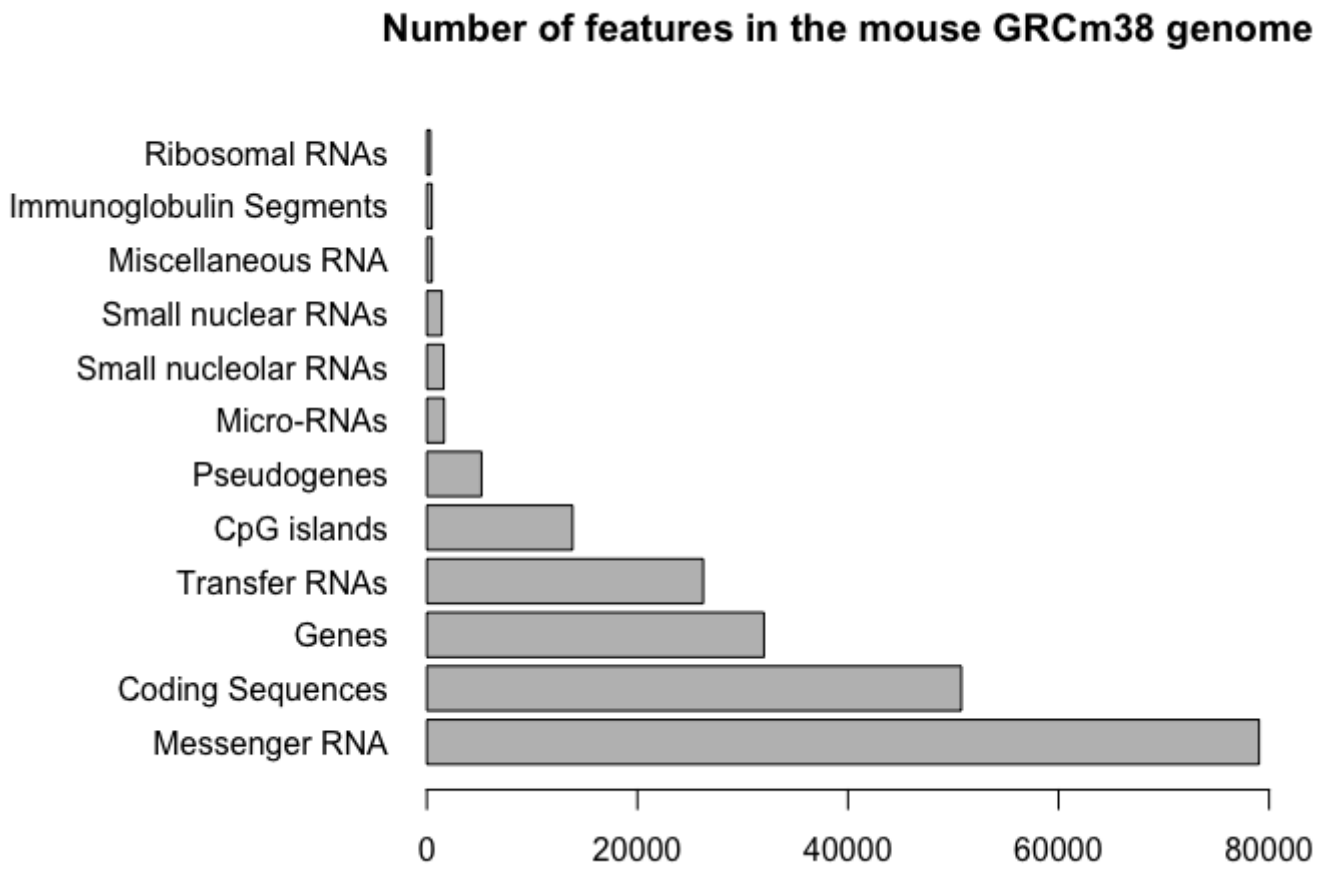
- The bars should be horizontal rather than vertical (`horiz=TRUE`).
- The count axis should be labelled (`ylab="A title"`)
- The feature names should be added to the y axis. (set `names.arg` to the **Feature** column of the data frame)
- The plot should be given a suitable title (`main="Some title"`)
- The text labels should all be horizontal (`las=1`) Note that you can pass this parameter either via par, or as an additional option to barplot.

- The margins should be adjusted to accommodate the labels (par `mar` parameter). You need to supply a 4 element vector for the bottom, left, top and right margin values.

Look at the value of `par()$mar` to see what the default values are so you know where to start. Note that you will have to redraw the barplot after making the changes to par.

Hide

```
par(mar=c(3.1, 11.1, 4.1, 2))
barplot(mouse$Count, names.arg=mouse$Feature,
        horiz=TRUE, ylab="",
        main="Number of features in the mouse GRCm38 genome",
        las=1, xlim=c(0,80000))
```



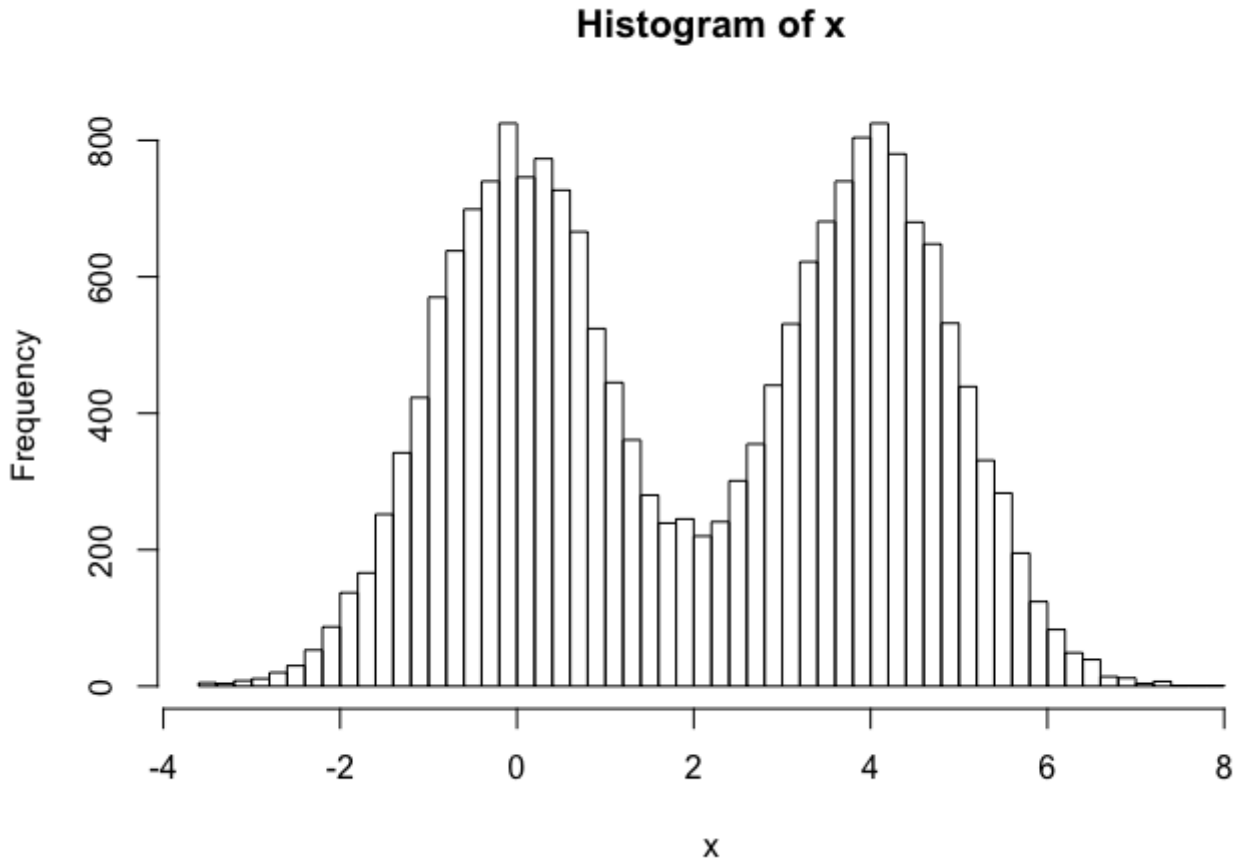
2C. Histograms

[**Extension if you have time**] Use this `hist()` function to plot out the distribution of 10000 points sampled from a standard normal distribution (use the `rnorm()` function) along with another 10000 points sampled from the same distribution but with an offset of 4.

- Example: `c(rnorm(10000),rnorm(10000)+4)`
- Find a suitable number of breaks to make the plot look nicer (`breaks=10` for example)

Hide

```
x <- c(rnorm(10000),rnorm(10000)+4)
hist(x, breaks=80)
```



Section 3: Using color in plots

3A. Providing color vectors

3A. The file **male_female_counts.txt** contains a time series split into male and female count values.

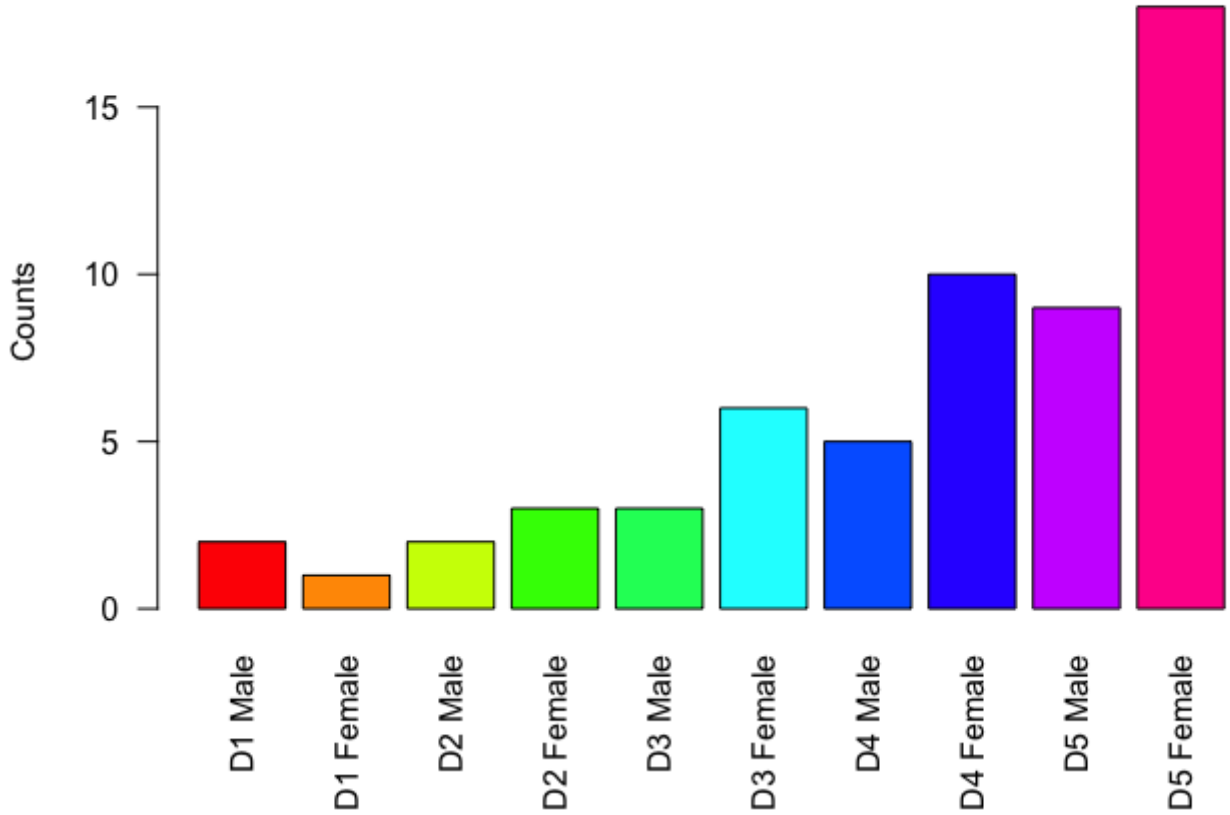
- Plot this as a barplot
- Make all bars different colors using the `rainbow()` function.
- Note that the `rainbow()` function takes a single argument, which is the number of colors to generate, eg `rainbow(10)`. Try making the vector of colors separately before passing it as the `col` argument to barplot (`col=rainbow(10)`).
- Rather than hard coding the number of colors, think how you could use the `nrow()` function to automatically generate the correct number of colors for the size of dataset.
- Re-plot, and make the bars for the males a different color to those for the females. In this case the male and female samples alternate so you can just pass a 2 color vector to the col parameter to achieve this effect. (`col=c("blue2","red2")`) for example.

Hide

```
mf <- read.delim("bimm143_05_rstats/male_female_counts.txt")
```

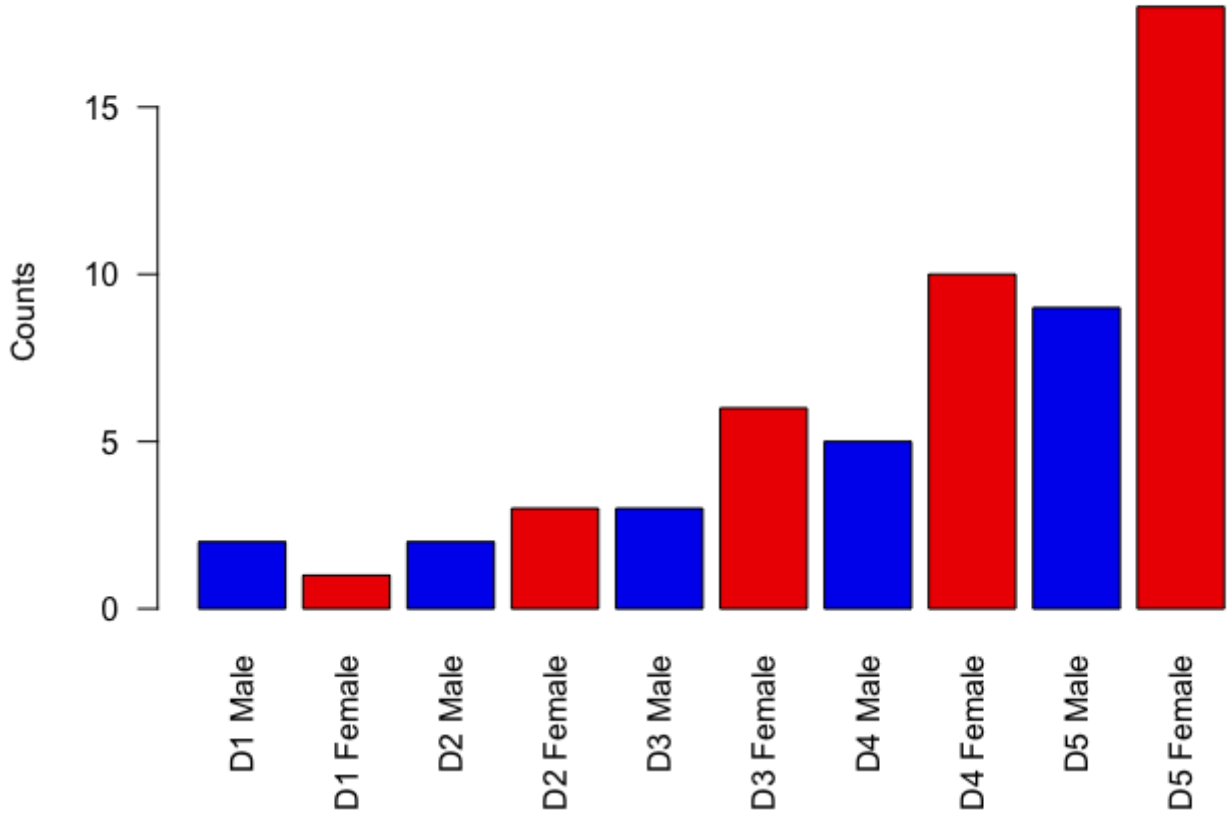
Hide

```
barplot(mf$Count, names.arg=mf$Sample, col=rainbow(nrow(mf)),
        las=2, ylab="Counts")
```



Hide

```
barplot(mf$Count, names.arg=mf$Sample, col=c("blue2","red2"),
        las=2, ylab="Counts")
```



3B. Coloring by value

3B. The file **up_down_expression.txt** contains an expression comparison dataset, but has an extra column which classifies the rows into one of 3 groups (up, down or unchanging).

Plot this as a scatterplot (plot) with the up being red, the down being blue and the unchanging being grey.

- Read in the file to an object named `genes`

Hide

```
genes <- read.delim("bimm143_05_rstats/up_down_expression.txt")
```

- How many genes are detailed in this file (i.e. how many rows are there)?
- To determine how many genes are up, down and unchanging in their expression values between the two conditions we can inspect the `genes$State` column. A useful function for this is the `table()` function, e.g. try `table(genes$State)`.

Hide

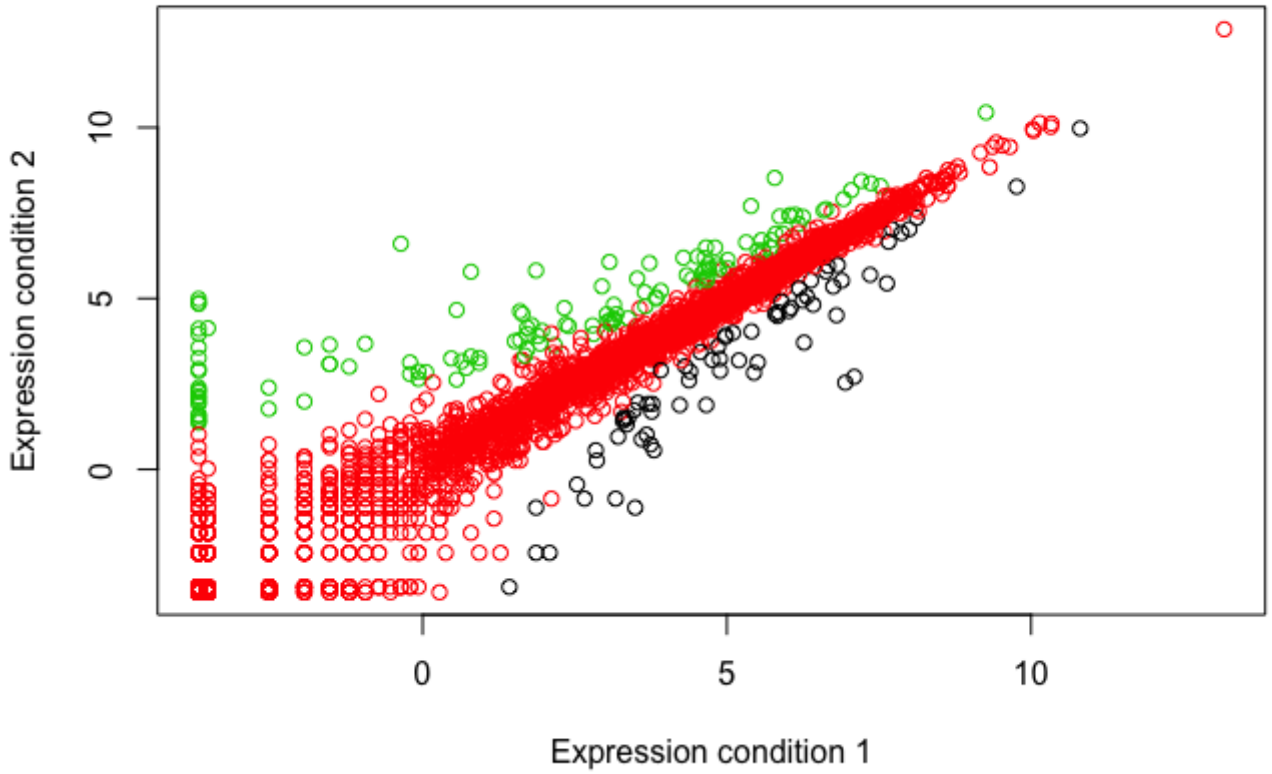
```
table(genes$State)
```

```
##
##       down unchanging         up
##         72       4997        127
```

- For graphing start by just plotting the `Condition1` column against the `Condition2` column in a plot
- Pass the `State` column as the `col` parameter (`col=genes$State` for example). This will set the color according to the state of each point, but the colors will be set automatically from the output of palette.

Hide

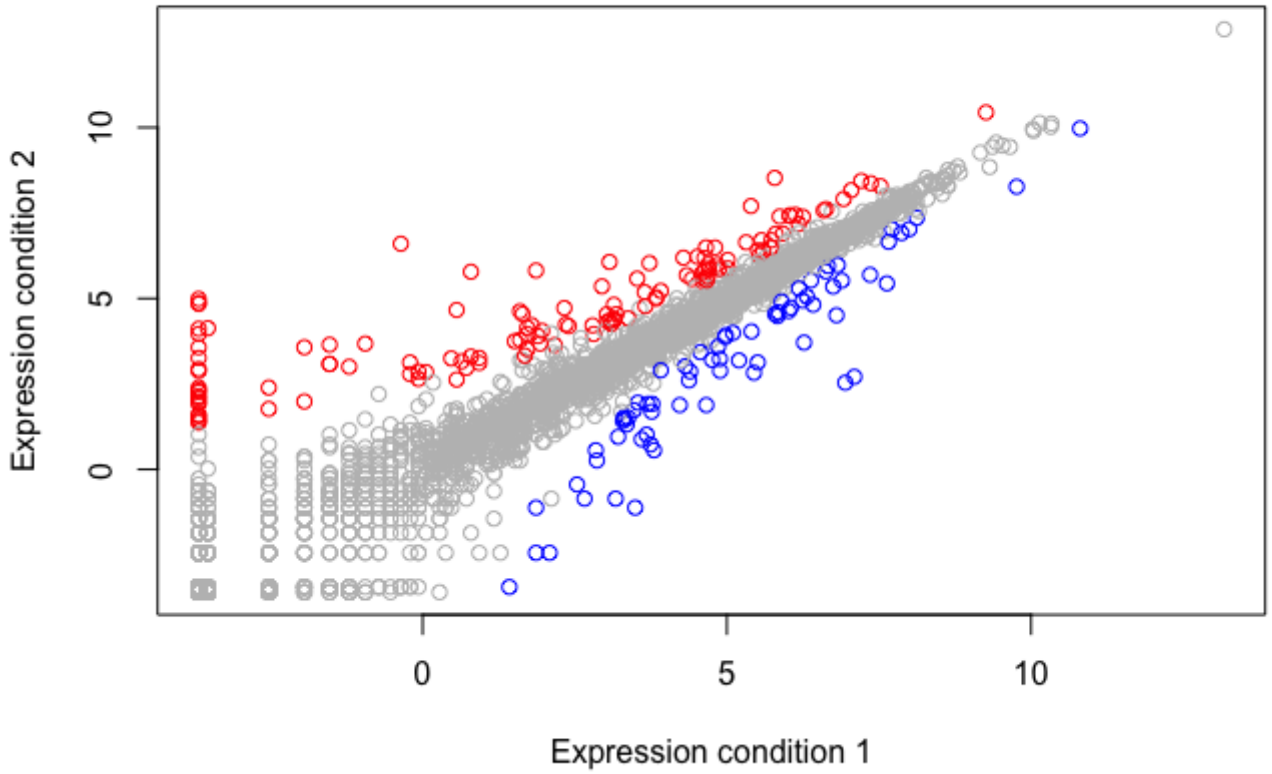
```
plot(genes$Condition1, genes$Condition2, col=genes$State,
     xlab="Expression condition 1", ylab="Expression condition 2")
```



- Run `palette()` to see what colors are there initially and check that you can see how these relate to the colors you get in your plot.
- Run `levels()` on the `State` column and match this with what you saw in `palette()` to see how each color was selected. Work out what colors you would need to put into palette to get the color selection you actually want.
- Use the `palette()` function to set the corresponding colors you want to use (eg `palette(c("red","green","blue"))` – but using the correct colors in the correct order.
- Redraw the plot and check that the colors are now what you wanted.

Hide

```
palette(c("blue","gray","red"))
plot(genes$Condition1, genes$Condition2, col=genes$State, xlab="Expression condition 1", ylab="Express
ion condition 2")
```



3C. Dynamic use of color

The uses of color described above would be what you would use to do categorical coloring, but often we want to use color for a more quantitative purpose. In some graph types you can specify a color function and it will dynamically generate an appropriate color for each data point, but it can be useful to do this more manually in other plot types.

Coloring by point density

One common use of dynamic color is to color a scatterplot by the number of points overlaid in a particular area so that you can get a better impression for where the majority of points fall. R has a built in function `densCols()` which can take in the data for a scatterplot and will calculate an appropriate color vector to use to accurately represent the density. This has a built in color scheme which is just shades of blue, but you can pass in your own color generating function (such as one generated by `colorRampPalette()`) to get whatever coloring you prefer.

The file `expression_methylation.txt` contains data for gene body methylation, promoter methylation and gene expression.

- Read this file into an R object called `meth`.

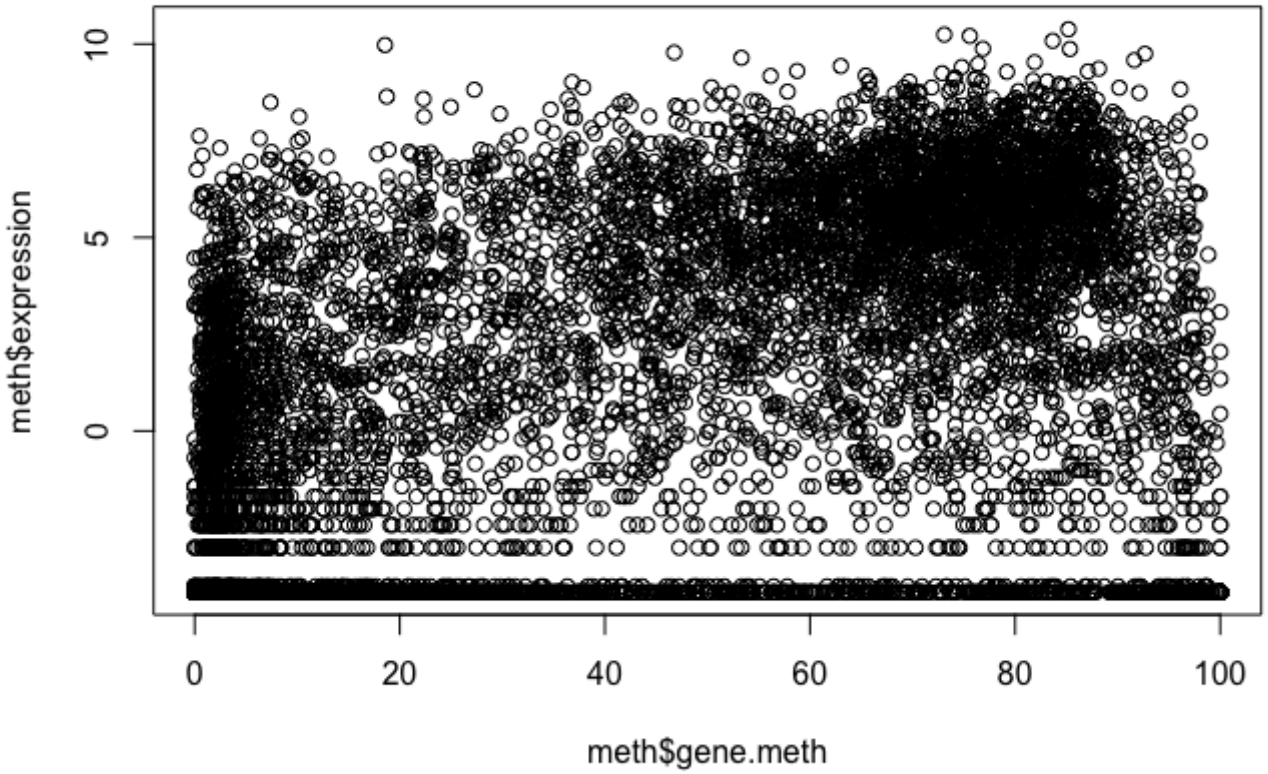
Hide

```
# Lets plot expresion vs gene regulation
meth <- read.delim("bimm143_05_rstats/expression_methylation.txt")
```

- How many genes are in this dataset?
- Draw a scatterplot (plot) of the `gene.meth` column against the `expression` column.

Hide

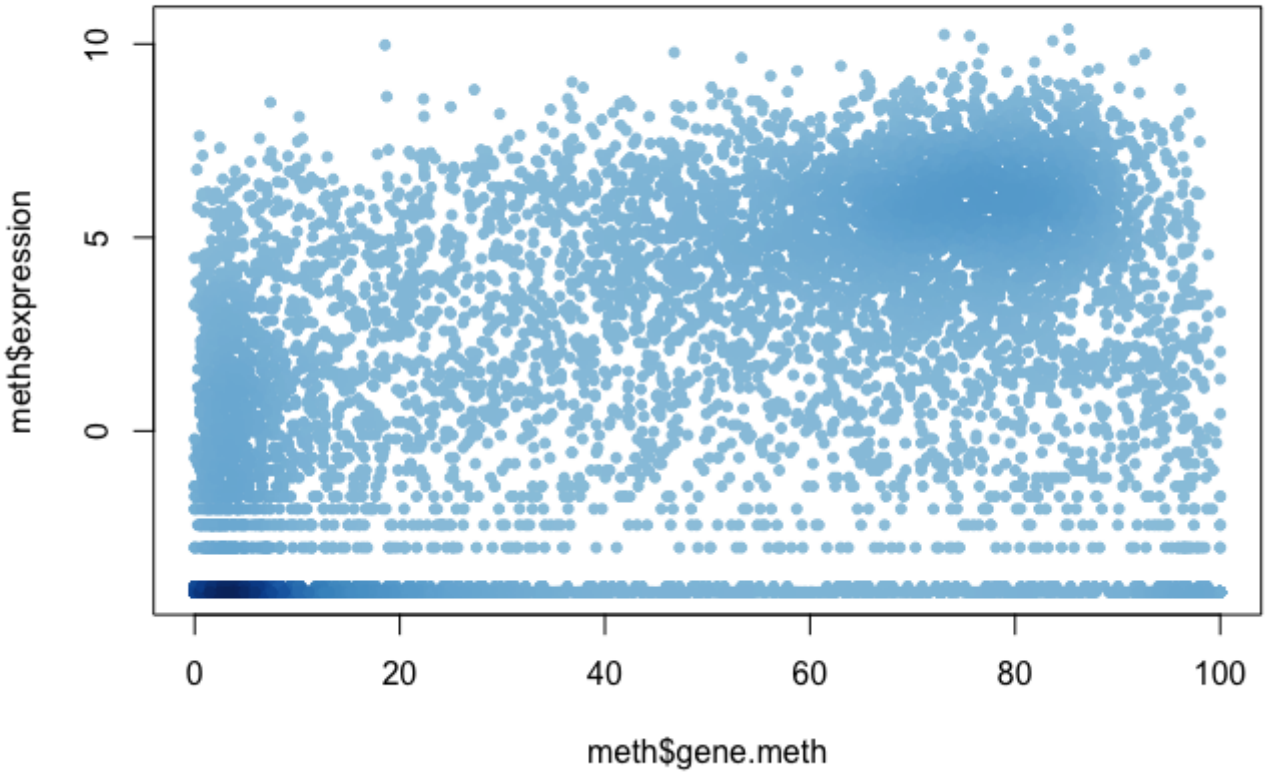
```
plot(meth$gene.meth, meth$expression)
```



- This is a busy plot with lots of data points in top of one another. Let improve it a little by coloring by point density. Use the `densCols()` function to make a new color vector that you can use in a new plot along with solid plotting character (e.g. `pch=20`).

Hide

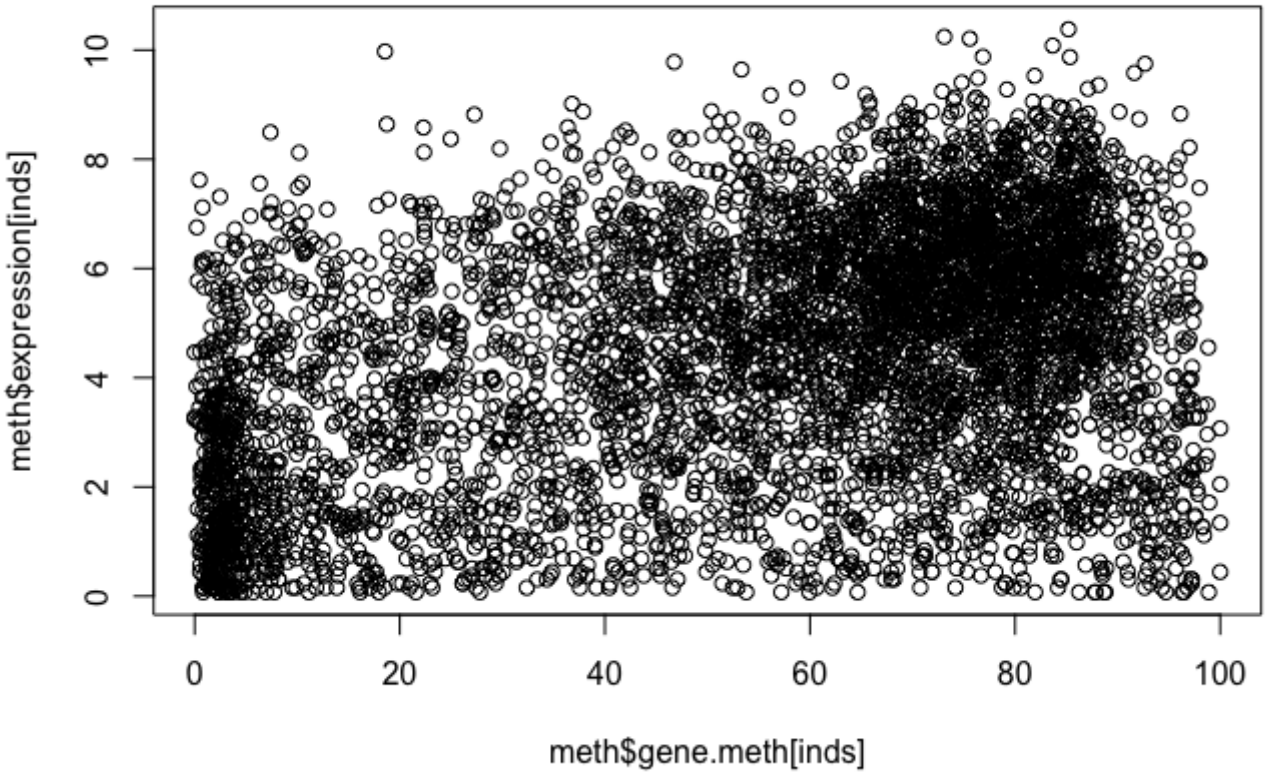
```
dcols <- densCols(meth$gene.meth, meth$expression)

# Plot changing the plot character ('pch') to a solid circle
plot(meth$gene.meth, meth$expression, col = dcols, pch = 20)
```



It looks like most of the data is clustered near the origin. Lets restrict ourselves to the genes that have more than zero `expresion` values

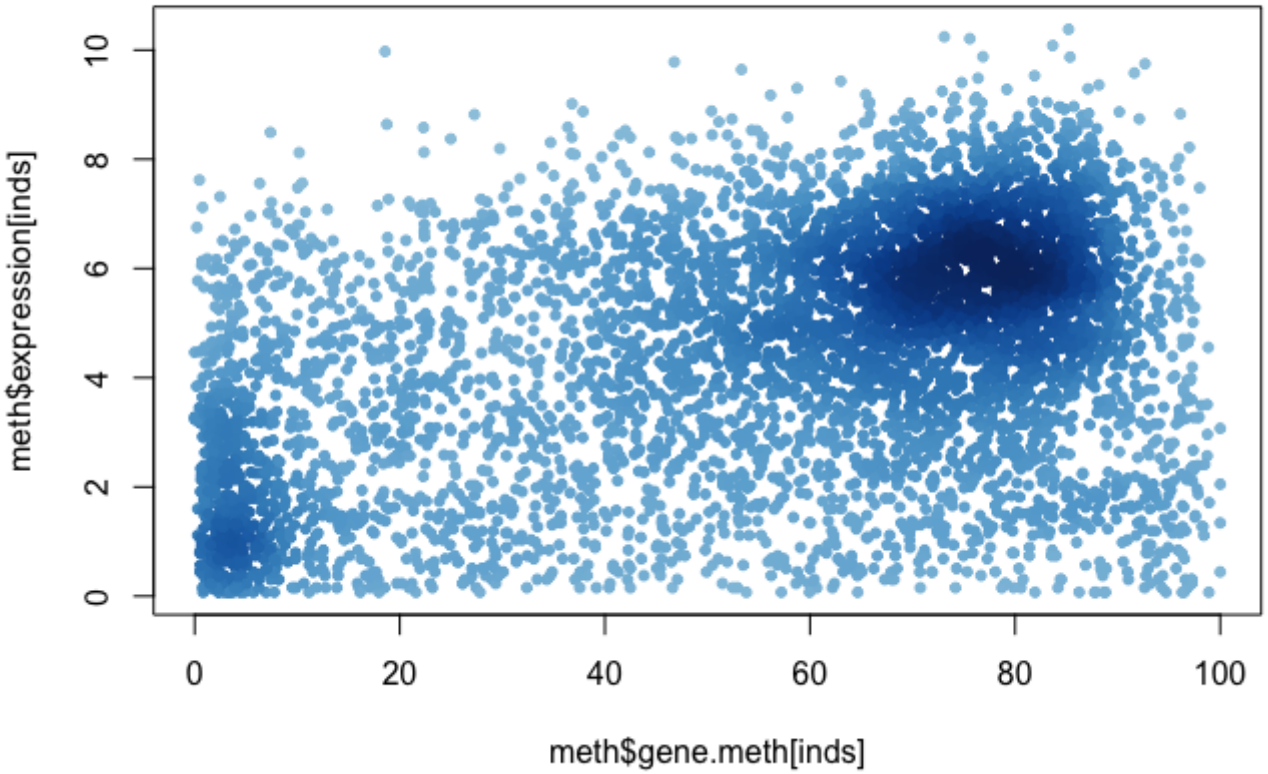
Hide

```
# Find the indices of genes with above 0 expresion
inds <- meth$expression > 0

# Plot just these genes
plot(meth$gene.meth[inds], meth$expression[inds])
```



Hide

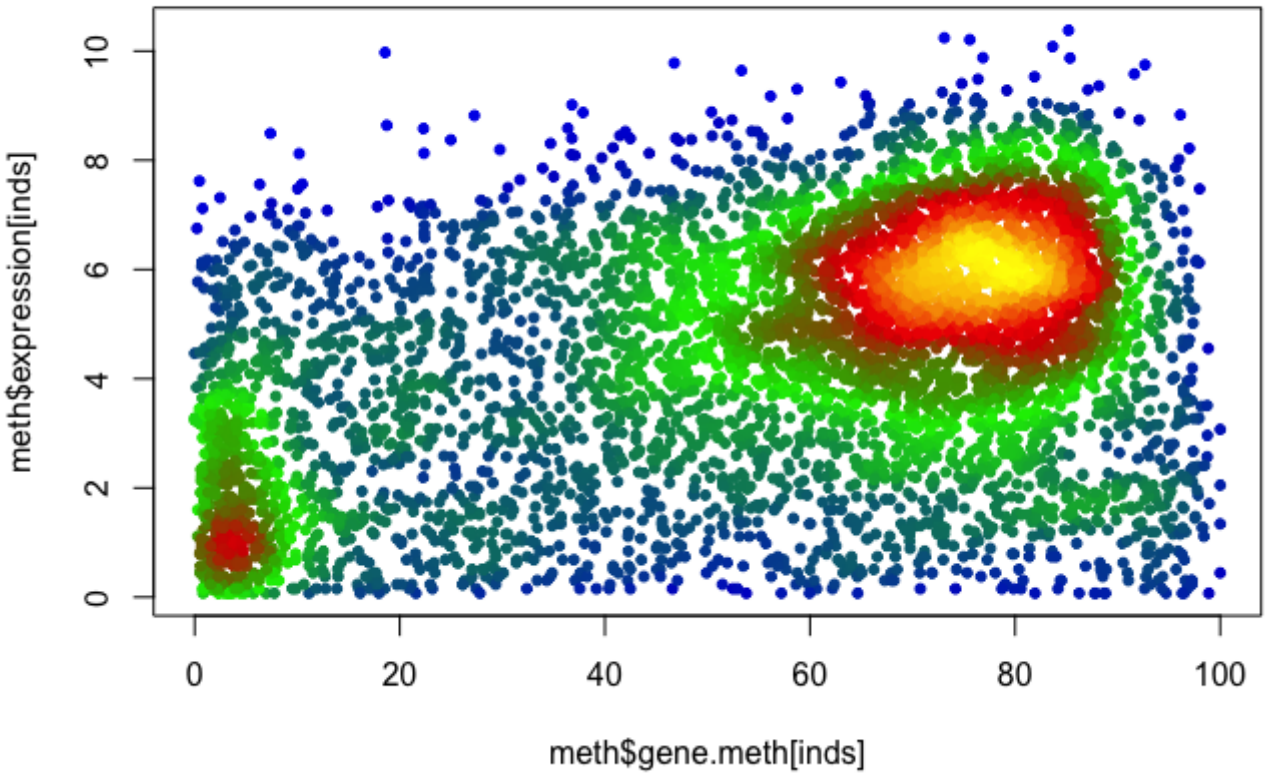
```
## Make a desnisty color vector for these genes and plot
dcols <- densCols(meth$gene.meth[inds], meth$expression[inds])

plot(meth$gene.meth[inds], meth$expression[inds], col = dcols, pch = 20)
```



- Change the `colramp` used by the `densCols()` function to go between blue, green, red and yellow with the `colorRampPalette()` function.

Hide

```
dcols.custom <- densCols(meth$gene.meth[inds], meth$expression[inds],
                         colramp = colorRampPalette(c("blue2",
                                                      "green2",
                                                      "red2",
                                                      "yellow")) )

plot(meth$gene.meth[inds], meth$expression[inds],
     col = dcols.custom, pch = 20)
```



Section 4: Optional extension

4A. Mapping colors

Whilst color in general is not a great way to represent quantitative data it can be sometimes useful to use the color of points in a plot as an extra quantitative variable. This will then require a way to map colors from a palette to values in a quantitative set. There is no built in base function within R to do this so we have borrowed one from a fictional lab mate - see the file `color_to_value_map.r` that contains a function `map.colors()` that may help us do this.

Understanding someone else's code

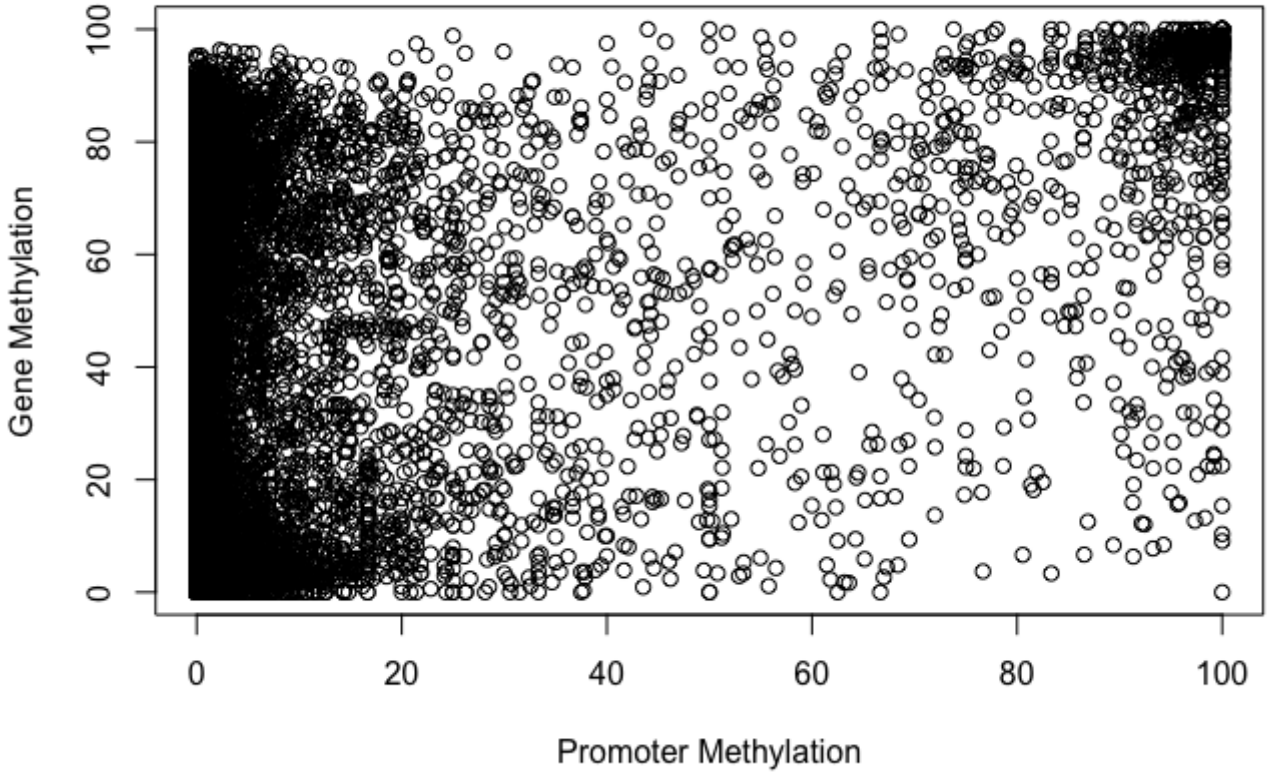
Note: We will come back to this section next day to help explain **functions** and what makes a good and (in this case) BAD function.

> This is an example of a poorly written function typical of something you might get from a lab mate that knows some R. We will discuss why it is poorly written next day. For now can you use it as described below? Points for discussion next day include: Can you figure out what this function is supposed to do? What format should the inputs be in order to work? How could we improve this function?

- Draw a scatterplot (plot) of the methylation dataset from the last section but this time plot the `promoter.meth` column against the `gene.meth` column.

Hide

```
#meth <- read.delim("bimm143_05_rstats/expression_methylation.txt")

plot(meth$promoter.meth, meth$gene.meth, ylab="Gene Methylation", xlab="Promoter Methylation")
```



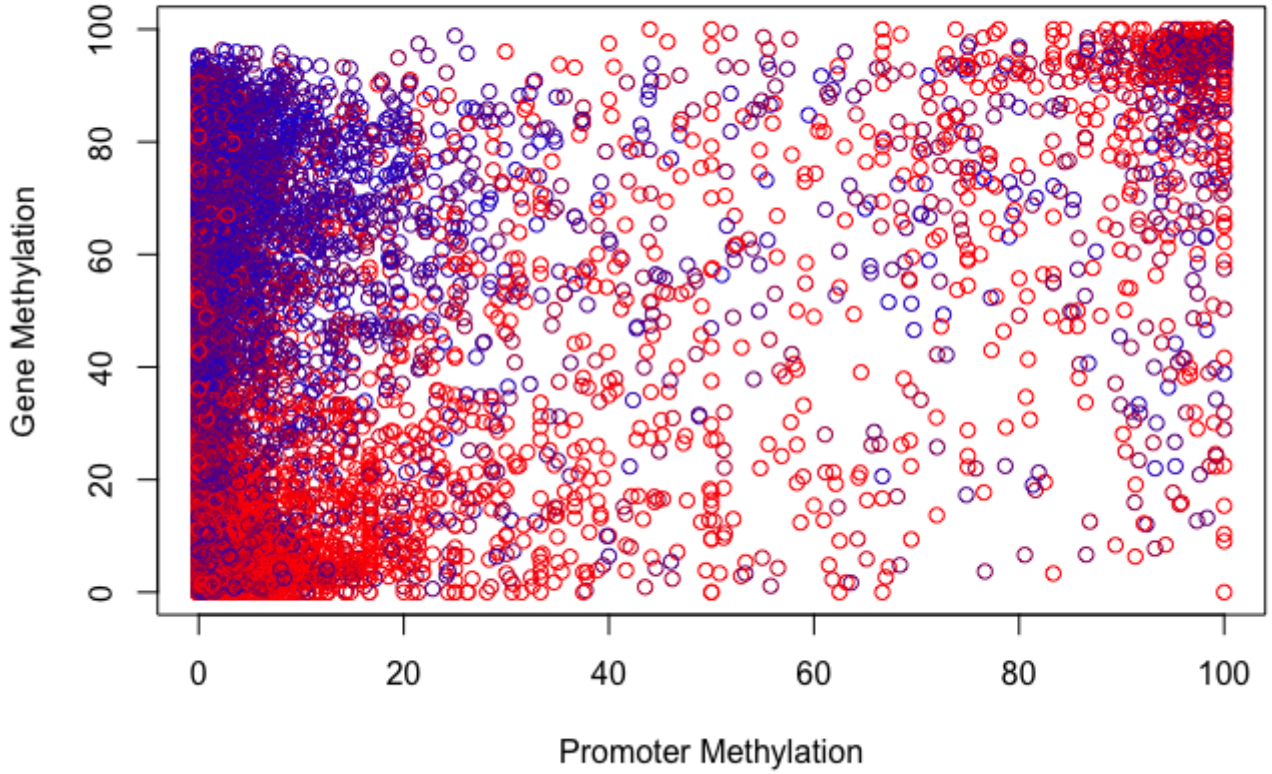
- Now you can work on coloring this plot by the expression column. For this you are going to need to run the `map.colors()` function we provided. This requires two bits of data – one is simply the values in the expression column, but the other is a vector of colors which define your color scale.

- You will need to start by constructing a color palette function from grey to red.
- Run `colorRampPalette(c("grey","red"))` to show that you can generate a function which will make colors running from grey to red.
- Call the function to generate a set of 100 colors and save these into a suitably named variable i.e: `colorRampPalette(c("grey","red"))(100)`.
- Now generate your actual color vector you are going to use. Call the `map.colors()` function you have been given with the set of expression column values, and the vector of colors you just generated. Save the per point colors which are returned into a new variable.

- Finally redraw the plot passing the per point colors to the `col` parameter.

Hide

```
# source the provided function so we can use it
source("bimm143_05_rstats/color_to_value_map.r")

mycols=map.colors(meth$expression,
                  c(max(meth$expression), min(meth$expression)),
                  colorRampPalette(c("blue","red"))(100))

plot(meth$promoter.meth, meth$gene.meth,
     ylab="Gene Methylation",
     xlab="Promoter Methylation",
     col=mycols)
```



Section 5: Detailed guide

A detailed guide to plotting with base R is provided as a PDF suplement (https://bioboot.github.io/bimm143_F18/class-material/lecture5-BIMM143_lab.pdf) that you can use as a reference whenever you need to plot data with base R. From **Page 7** onward it goes into more details and covers more plot types than we could in today's hands-on session.

Muddy Point Assessment

Link to today's muddy point assesment (https://goo.gl/forms/qIW4O4PUoixTzy7J2).

Session Info

Hide

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.5.1     backports_1.1.2    magrittr_1.5
##  [4] rprojroot_1.3-2    tools_3.5.1        htmltools_0.3.6
##  [7] yaml_2.2.0         Rcpp_0.12.19       KernSmooth_2.23-15
## [10] stringi_1.2.4      rmarkdown_1.10     knitr_1.20
## [13] stringr_1.3.1      digest_0.6.18      evaluate_0.12
```