BirminghamMDIBL

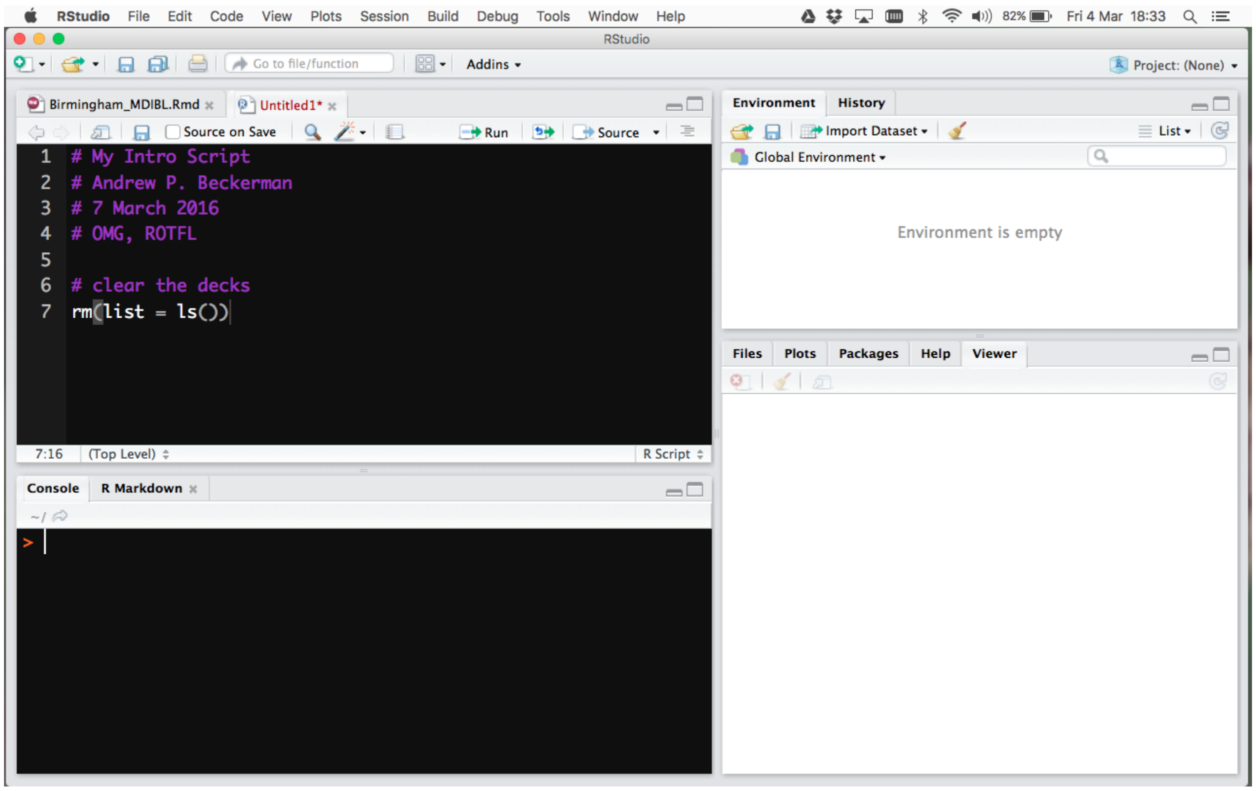
APB

28 February 2016

Using R with Rstudio is AWESOME.

Building an annotated, archived, repeatable, share-able cross platform record of what you do.

- 4 panes: the console (engine), the script (your code), Plots/Packages and Data Stuff



```
# My Intro Script
# Andrew P. Beckerman
# 7 March 2016
# OMG, ROTFL

# clear the decks
rm(list = ls())
```

Using R as a calculator

R is a big giant calculator. Lets practice in the CONSOLE. As you do these, WATCH what RStudio does with ()'s

```
# ------------------------------------------------------------------------
# BASIC PRACTICAL
# ------------------------------------------------------------------------
# Maths and Functions
1+1
2*7/8-9
log(exp(1))
log10(1000)
log(1000)
```

```r
sin(2*pi)
2^10
sqrt(81)
```

What you should know from this.

- R does maths correctly
- the default logarithm is ln, not base 10
- R has function and objects

Lets look at what more R can do. Here we see the basics of how R can make sequences. We see it can make integers using ':' and sequences of certain lengths or 'by' certain steps.

Moving to the script.

Lets start using the script now. Add some basic annotation at the top and a good practice activity - clearing R's brain.

```r
# NAME
# DATE
# Intro to R script

# clear R's brain
rm(list = ls())
```

Now, lets have some more fun… building sequences of numbers to see how R works.

```r
# Sequences, Vectorisation
1:10
seq(from = 1, to = 10, by = 1) # note the three arguments, from, to and by
seq(from = 1, to = 10, length = 12) # note the three arguments, from, to and length
```

What we learn here:

- R can make sequences of integers easily
- seq() is a function with at least three arguments.
- from, to, by is a sequence in "steps"
- from, to, length is a sequnce of a fixed number of numbers.

You can make objects in R. We call this ASSIGNMENT, and we typically use `<-` (not `=`, though it is possible)

```r
# assignments
x <- 1:10

# look at it
x

# use it to make another variable
y <- x^2

# work with both
x+y
x*y
```

What we learn here:

- You can create objects to use.
- The objects can be vectors, matrices, data frames etc.
- R can perform operations with objects.
- operations are by default *element - by - element*
- there is facility for matrix multiplication and linear algebra

Seeing what you've done, inside R.

Lets move focus to the console for a moment.... wait. Have you saved the script? Use CTRL - 2. Ohhh - check out the View Menu... Once there, type:

```
ls()
```

```
## character(0)
```

`ls()` in R is like `ls` in unix/linux. It is listing the objects....

Making some graphics.... BASE graphics

Here we introduce the basics of making a plot. There are several ways to make pictures. The most popular now is ggplot2, but there is also the lattice library. More on packages and libraries below.

Start by making some data. Then, roll this data in to a data frame, which is like a spreadsheet.

We use the base function `plot()` and a *formula* that specifies the y and the x axes of the plot.

```
# Your first plot
x <- 1:10
y <- x^2

myDat <- data.frame(Beer = x, Goggles = y)
myDat
```

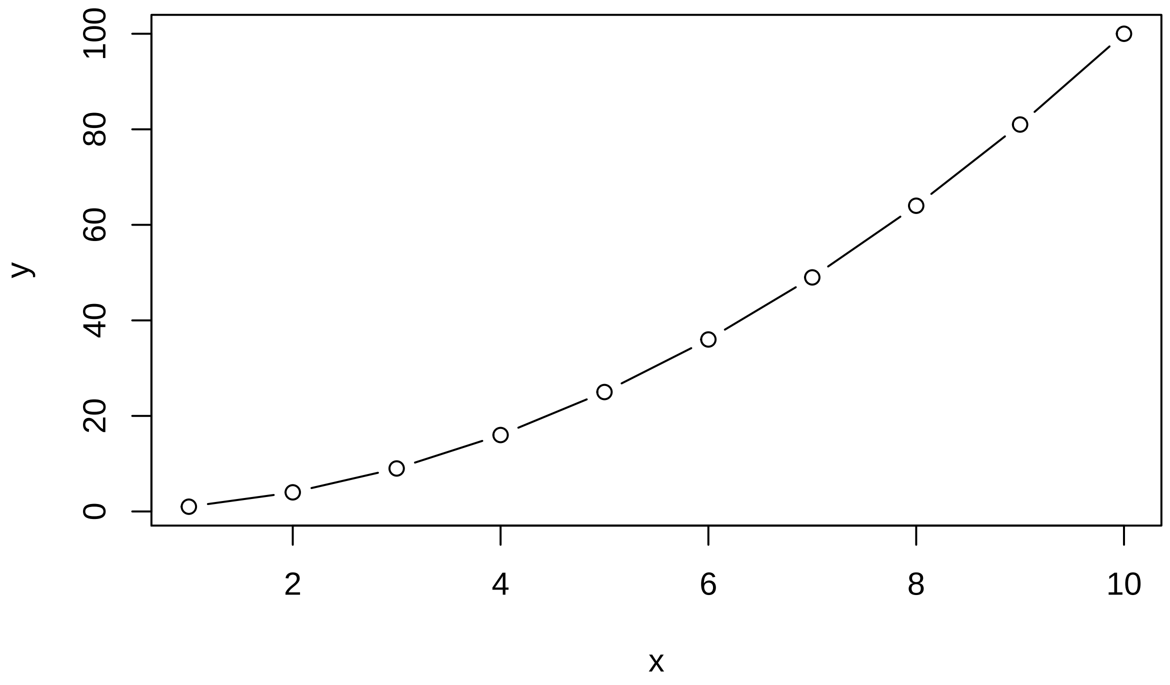
```
##    Beer Goggles
## 1     1       1
## 2     2       4
## 3     3       9
## 4     4      16
## 5     5      25
## 6     6      36
## 7     7      49
## 8     8      64
## 9     9      81
## 10   10     100
```

What we learn here:

- data.frame makes a spreadsheet like object with columns
- we name the data frame using `<-`
- we name the columns inside the data.frame using `=`

Right…lets make the picture!

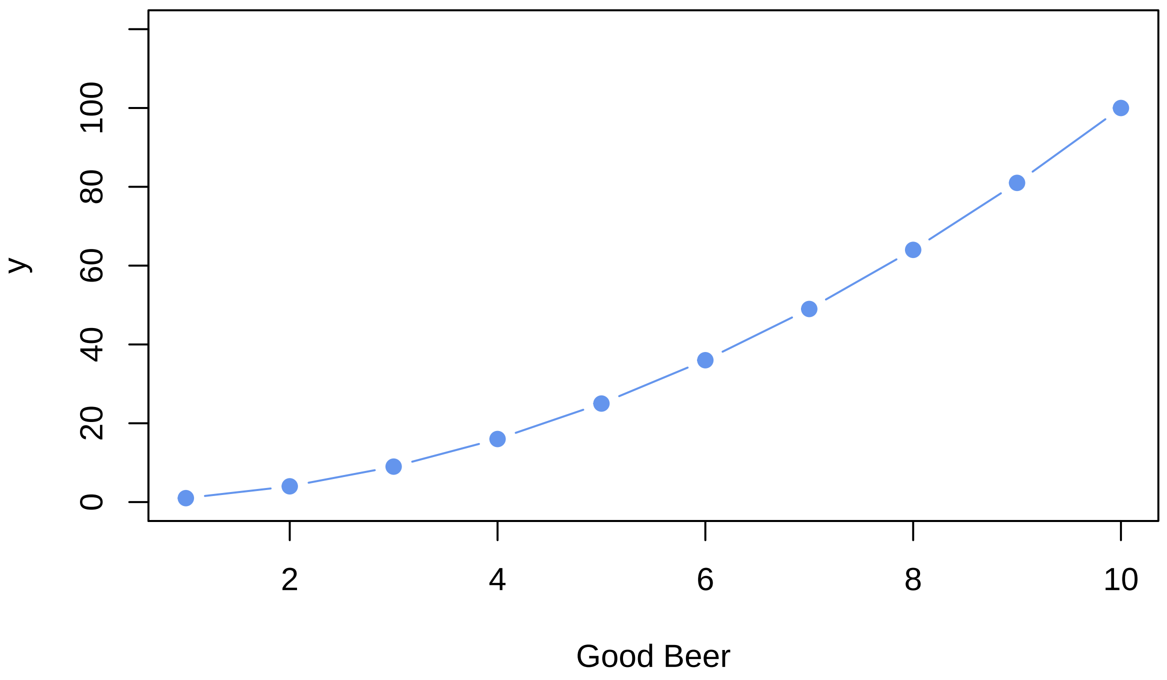
```r
# FORMULA INTERFACE
plot(y ~ x, data = myDat, type = "b")
```



What we learn here: * plot is nice * when the data frame has column names that are nice, they look nice on the graph * the `type` argument *b* is for both. *p* is for points only. *l* is for lines only.

Plots can be customised…. here we alter the limits of the y-axis, the label on the x-axis, and make the points filled with cornflowerblue-ness….

```r
# FORMULA INTERFACE
plot(y ~ x, data = myDat, type = "b",
     ylim = c(0,120),
     xlab = "Good Beer",
     pch = 19, col = "cornflowerblue")
```



Back to some Mechanics... Managing Packages

In your 'omics life, you will need to access packages from TWO locations. CRAN, which is the *C*omprehensive *R* *A*rchive *N*etwork, and Bioconductor.

CRAN has ~7500 packages available and they are curated at the CRAN Task Views website:

Packages: LINK `http://star-www.st-andrews.ac.uk/cran/web/packages/`

TaskViews: LINK `http://star-www.st-andrews.ac.uk/cran/web/views/`

Bioconductor is more focused on 'omics, and has a different curation mechanism

Here are the basics for installing stuff.

```
# ----------------------------------------------------------------------
# Installing packages
# ----------------------------------------------------------------------

# R-CRAN
# get dplyr and ggplot2 - see http://ggplot2.org and https://cran.rstudio.com/web/packages/dplyr/vignet
# and just use google
# install.packages gets packages and installs them
# SEE RSTUDIO TOO!

install.packages(c('ggplot2', 'dplyr'))

# R - bioconductor
# the first line grabs a special piece of code to make it easy to get packages from
# BIOCONDUCTOR.
# The second line gets a specific set of packages.... you'll be using these!

source("https://bioconductor.org/biocLite.R")
biocLite(c("edgeR", "KEGGREST", "pathview","org.Dr.eg.db"))
```

Some stuff will happen on your screen when you do this…

You have to think about packages like this. You INSTALL PACKAGES from an app store. To use them, however, we need to invoke a second function (like pressing on the icon). This is called `library()`

Typically, we'd do the isntallations - and you only need to do this ONCE - and then put the library() commands at the top of your script….

```
# I need these libraries
library(KEGGREST)
library(pathview)
library(org.Dr.eg.db)
```

Last *formal* thing - reading data in from the web, or your own computer.

```
# ----------------------------------------------------------------------
# read some data using read.csv()
# ----------------------------------------------------------------------

# this is a path to data on the web
UrlAddress<-'https://raw.githubusercontent.com/andbeck/RDatas/master/myDF.csv'

# # you will often use a path on your computer
# compAddress<-'C:/Documents/RStuff/myDF.csv' # windows
# compAddress<-'~/Documents/RStuff/myDF.csv' # linux/unix/osx

# get the data
myDF <- read.csv(UrlAddress)
str(myDF)
```

```
## 'data.frame':    100 obs. of  4 variables:
##  $ X         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ expression: num  12.52 1.31 9.02 9.5 10.06 ...
##  $ size      : num  -0.506 1.8712 -0.1033 0.1141 0.0369 ...
##  $ category  : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
```

```
# # with comp
# myDF <- read.csv(compAddress)

# # also directly using a path
# myDF <- read.csv('~/Documents/Rstuff')
```

```
# ----------------------------
# make a list (also common)
myList <- list(x = 1:10,
   y = LETTERS[1:5],
   w = matrix(rnorm(100,0,1), nrow = 10, ncol = 10, byrow = TRUE))
```

What lies below is a bit about how to find your way around data

'omics data structures are highly variable. Things are stored in data frames and in lists. Below you have - an introduction to `[row, column]`, `$` for grabbing bits of data frames - an introducto to `subset()` for more efficient grabbing - an introduction to `[[ ]]` for grabbing things from lists

You'll want to work through these before tomorrow's activities.

-NOTE - lots of people are using the package dplyr now with verbs select(), slice(), filter(); VERY good tutorials online

```
# ----------------------------------------------------------------------
# Explore the data with important functions in R
# [ ] and $ and ==
# REMEMBER: ROWS THEN COLUMNS
# ----------------------------------------------------------------------
# NOTE - lots of people are using the package dplyr now with
# verbs select(), slice(), filter()
# VERY good tutorials online
# ----------------------------------------------------------------------

# BASE BASICS
myDF[5,2] # 5th row, 2nd column
```

```
## [1] 10.05893
```

```
myDF$expression # expression column
```

```
##   [1] 12.517538  1.314097  9.016838  9.503552 10.058929  7.665571 10.888666
##   [8]  7.978765  7.660154  8.647248  6.014360 13.729142 11.808561  6.199458
##  [15]  5.672131 12.018454 15.260583  3.868344 14.726813 11.665405 12.494673
##  [22] 15.193945  6.033581  6.425320  9.787794 15.281029 16.443706  3.192887
##  [29] 13.885776  4.950545  3.195074 26.605668 12.156804  3.535982 12.737946
##  [36] 21.454717  6.797553 15.073445 11.310967 11.433194  7.964187  6.546595
##  [43] 14.422700  5.523245 14.150787  5.677373 22.440774 10.443891 20.035131
##  [50]  9.031042 28.517471 29.475969 34.862050 30.531403 31.592105 32.048833
##  [57] 31.461238 24.528072 38.931406 33.208995 30.994866 29.040634 27.535157
##  [64] 27.531762 21.975596 26.610769 28.663481 26.534582 32.622669 34.101971
##  [71] 28.728651 31.806852 30.507381 23.458605 33.575201 25.412200 35.492783
##  [78] 33.556421 23.281725 30.938442 24.628242 23.968088 30.886866 33.312532
##  [85] 23.507332 31.591445 25.820811 33.364705 21.973055 31.934126 31.137705
##  [92] 27.894069 28.283657 29.103208 26.568325 35.225372 28.497635 37.298256
##  [99] 35.073917 32.311718
```

```
myDF[,1] # column 1
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
myList[[3]] # 3rd piece of the list => w
```

```
##            [,1]        [,2]        [,3]       [,4]       [,5]        [,6]
##  [1,] 0.3784818 -0.98555255  1.18293363 -0.6084560 -0.5824356  2.13520712
##  [2,] 1.8674174  1.17228526  0.84954069  0.5529234 -2.4139080 -0.55079032
##  [3,] -1.0154936 -0.60957192 -1.08564719 -0.2128565  0.3064752  1.05447891
##  [4,] -0.9268055 -0.00524542 -0.06006215  0.1598590  1.3887750 -0.64768999
##  [5,]  1.2495550 -0.89509442 -0.26988564  1.1017365  0.3178161  1.11880722
##  [6,]  0.7248319  1.54375279  0.14274157  0.2354464  2.3865192 -0.47403741
##  [7,]  0.3603023 -0.67308661  0.53841839 -1.4854961  0.9503612 -0.38307292
##  [8,] -1.8973337 -0.17592055 -0.55620750 -0.7863620 -1.2814259 -0.48631656
##  [9,]  1.0881571  0.24594515 -0.31494258  1.3231478  0.4338763 -0.81672010
## [10,]  0.5377668 -0.75556567 -1.03460194  0.1506846 -0.5358933 -0.08369076
##              [,7]        [,8]         [,9]       [,10]
##  [1,] -0.67268882 -1.03542177 -0.738048628  1.10609900
##  [2,] -0.37380164  0.70598613 -0.873376347 -0.65011861
##  [3,] -0.05009911 -0.64426968 -0.004147781  0.88183915
##  [4,]  0.48470362 -1.90162969  0.014172725  0.98978551
##  [5,]  0.49737925 -0.30503972 -1.277125001 -0.93551181
##  [6,] -0.32906275 -0.07367749 -0.746511878  0.64064053
##  [7,]  0.37019037  0.79920000 -1.110998782  0.95249322
##  [8,] -1.21430710 -1.39657717  1.349441308  0.38570434
##  [9,] -1.15109427 -0.70773230  0.598888286 -0.09052874
## [10,] -0.61884382 -0.94069806  0.930523661  0.32398842
```

```
# MORE harder
myList[["w"]][2,] # get the ninth row from the matrix w that is the third element in myList
```

```
##  [1]  1.8674174  1.1722853  0.8495407  0.5529234 -2.4139080 -0.5507903
##  [7] -0.3738016  0.7059861 -0.8733763 -0.6501186
```

```
myList$w[2,] # get the ninth row from the matrix w that is the third element in myList
```

```
##  [1]  1.8674174  1.1722853  0.8495407  0.5529234 -2.4139080 -0.5507903
##  [7] -0.3738016  0.7059861 -0.8733763 -0.6501186
```

```
# More Harder
myDF[51:100,] # rows 51:100 are labeled "B"
```

```
##       X expression        size category
## 51   51   28.51747 -0.45405598        B
## 52   52   29.47597 -1.97200873        B
## 53   53   34.86205 -0.50698562        B
## 54   54   30.53140 -0.61006478        B
## 55   55   31.59210 -0.57905778        B
## 56   56   32.04883  0.73580411        B
## 57   57   31.46124  1.62414287        B
## 58   58   24.52807  0.36799274        B
## 59   59   38.93141 -0.23602739        B
## 60   60   33.20899  1.36383951        B
## 61   61   30.99487  0.86499675        B
## 62   62   29.04063  0.30832280        B
## 63   63   27.53516 -0.83933275        B
## 64   64   27.53176  1.58860207        B
## 65   65   21.97560  1.16240904        B
## 66   66   26.61077  1.16098381        B
## 67   67   28.66348 -0.20333346        B
## 68   68   26.53458  0.92408774        B
## 69   69   32.62267 -1.32905326        B
## 70   70   34.10197 -0.06406315        B
## 71   71   28.72865  0.46624152        B
## 72   72   31.80685  0.84064708        B
## 73   73   30.50738  0.83803932        B
## 74   74   23.45861  0.76108249        B
## 75   75   33.57520  0.25438475        B
## 76   76   25.41220  0.84145960        B
## 77   77   35.49278  1.51905983        B
## 78   78   33.55642 -0.78395274        B
## 79   79   23.28172  0.06206436        B
## 80   80   30.93844  1.99305817        B
## 81   81   24.62824  0.89936350        B
## 82   82   23.96809  1.96480011        B
## 83   83   30.88687  0.89873445        B
## 84   84   33.31253 -0.01559538        B
## 85   85   23.50733 -0.18048792        B
## 86   86   31.59144  1.18288538        B
## 87   87   25.82081 -0.20844010        B
## 88   88   33.36471 -0.11618843        B
## 89   89   21.97306  1.92841533        B
## 90   90   31.93413 -2.01769093        B
## 91   91   31.13771 -0.29022348        B
## 92   92   27.89407 -0.14470734        B
## 93   93   28.28366 -0.10148508        B
## 94   94   29.10321  1.13478704        B
## 95   95   26.56832  1.68365002        B
## 96   96   35.22537 -0.42649623        B
## 97   97   28.49763 -1.86591499        B
## 98   98   37.29826  0.11377617        B
## 99   99   35.07392  0.53367742        B
## 100 100   32.31172 -1.33071604        B
```

```
myDF[myDF$category=="B",] # rows 51:100, but access functionally by definition in category column.
```

```
##        X expression        size category
## 51   51   28.51747 -0.45405598        B
## 52   52   29.47597 -1.97200873        B
## 53   53   34.86205 -0.50698562        B
## 54   54   30.53140 -0.61006478        B
## 55   55   31.59210 -0.57905778        B
## 56   56   32.04883  0.73580411        B
## 57   57   31.46124  1.62414287        B
## 58   58   24.52807  0.36799274        B
## 59   59   38.93141 -0.23602739        B
## 60   60   33.20899  1.36383951        B
## 61   61   30.99487  0.86499675        B
## 62   62   29.04063  0.30832280        B
## 63   63   27.53516 -0.83933275        B
## 64   64   27.53176  1.58860207        B
## 65   65   21.97560  1.16240904        B
## 66   66   26.61077  1.16098381        B
## 67   67   28.66348 -0.20333346        B
## 68   68   26.53458  0.92408774        B
## 69   69   32.62267 -1.32905326        B
## 70   70   34.10197 -0.06406315        B
## 71   71   28.72865  0.46624152        B
## 72   72   31.80685  0.84064708        B
## 73   73   30.50738  0.83803932        B
## 74   74   23.45861  0.76108249        B
## 75   75   33.57520  0.25438475        B
## 76   76   25.41220  0.84145960        B
## 77   77   35.49278  1.51905983        B
## 78   78   33.55642 -0.78395274        B
## 79   79   23.28172  0.06206436        B
## 80   80   30.93844  1.99305817        B
## 81   81   24.62824  0.89936350        B
## 82   82   23.96809  1.96480011        B
## 83   83   30.88687  0.89873445        B
## 84   84   33.31253 -0.01559538        B
## 85   85   23.50733 -0.18048792        B
## 86   86   31.59144  1.18288538        B
## 87   87   25.82081 -0.20844010        B
## 88   88   33.36471 -0.11618843        B
## 89   89   21.97306  1.92841533        B
## 90   90   31.93413 -2.01769093        B
## 91   91   31.13771 -0.29022348        B
## 92   92   27.89407 -0.14470734        B
## 93   93   28.28366 -0.10148508        B
## 94   94   29.10321  1.13478704        B
## 95   95   26.56832  1.68365002        B
## 96   96   35.22537 -0.42649623        B
## 97   97   28.49763 -1.86591499        B
## 98   98   37.29826  0.11377617        B
## 99   99   35.07392  0.53367742        B
## 100 100   32.31172 -1.33071604        B
```

```
# ----------------------------------------------------------------------
# learn how to use subset
# ----------------------------------------------------------------------
subset(myDF, category=="B") # rows
```

```
##       X expression        size category
## 51   51   28.51747 -0.45405598        B
## 52   52   29.47597 -1.97200873        B
## 53   53   34.86205 -0.50698562        B
## 54   54   30.53140 -0.61006478        B
## 55   55   31.59210 -0.57905778        B
## 56   56   32.04883  0.73580411        B
## 57   57   31.46124  1.62414287        B
## 58   58   24.52807  0.36799274        B
## 59   59   38.93141 -0.23602739        B
## 60   60   33.20899  1.36383951        B
## 61   61   30.99487  0.86499675        B
## 62   62   29.04063  0.30832280        B
## 63   63   27.53516 -0.83933275        B
## 64   64   27.53176  1.58860207        B
## 65   65   21.97560  1.16240904         B
## 66   66   26.61077  1.16098381         B
## 67   67   28.66348 -0.20333346         B
## 68   68   26.53458  0.92408774         B
## 69   69   32.62267 -1.32905326         B
## 70   70   34.10197 -0.06406315         B
## 71   71   28.72865  0.46624152         B
## 72   72   31.80685  0.84064708         B
## 73   73   30.50738  0.83803932         B
## 74   74   23.45861  0.76108249         B
## 75   75   33.57520  0.25438475         B
## 76   76   25.41220  0.84145960         B
## 77   77   35.49278  1.51905983         B
## 78   78   33.55642 -0.78395274         B
## 79   79   23.28172  0.06206436         B
## 80   80   30.93844  1.99305817         B
## 81   81   24.62824  0.89936350         B
## 82   82   23.96809  1.96480011         B
## 83   83   30.88687  0.89873445         B
## 84   84   33.31253 -0.01559538         B
## 85   85   23.50733 -0.18048792         B
## 86   86   31.59144  1.18288538         B
## 87   87   25.82081 -0.20844010         B
## 88   88   33.36471 -0.11618843         B
## 89   89   21.97306  1.92841533         B
## 90   90   31.93413 -2.01769093         B
## 91   91   31.13771 -0.29022348         B
## 92   92   27.89407 -0.14470734         B
## 93   93   28.28366 -0.10148508         B
## 94   94   29.10321  1.13478704         B
## 95   95   26.56832  1.68365002         B
## 96   96   35.22537 -0.42649623         B
## 97   97   28.49763 -1.86591499         B
## 98   98   37.29826  0.11377617         B
## 99   99   35.07392  0.53367742         B
## 100 100   32.31172 -1.33071604         B
```

```r
subset(myDF, select = "expression") # columns
```

```
##     expression
## 1    12.517538
## 2     1.314097
## 3     9.016838
## 4     9.503552
## 5    10.058929
## 6     7.665571
## 7    10.888666
## 8     7.978765
## 9     7.660154
## 10    8.647248
## 11    6.014360
## 12   13.729142
## 13   11.808561
## 14    6.199458
## 15    5.672131
## 16   12.018454
## 17   15.260583
## 18    3.868344
## 19   14.726813
## 20   11.665405
## 21   12.494673
## 22   15.193945
## 23    6.033581
## 24    6.425320
## 25    9.787794
## 26   15.281029
## 27   16.443706
## 28    3.192887
## 29   13.885776
## 30    4.950545
## 31    3.195074
## 32   26.605668
## 33   12.156804
## 34    3.535982
## 35   12.737946
## 36   21.454717
## 37    6.797553
## 38   15.073445
## 39   11.310967
## 40   11.433194
## 41    7.964187
## 42    6.546595
## 43   14.422700
## 44    5.523245
## 45   14.150787
## 46    5.677373
## 47   22.440774
## 48   10.443891
## 49   20.035131
## 50    9.031042
## 51   28.517471
## 52   29.475969
## 53   34.862050
## 54   30.531403
## 55   31.592105
## 56   32.048833
## 57   31.461238
## 58   24.528072
## 59   38.931406
## 60   33.208995
## 61   30.994866
## 62   29.040634
## 63   27.535157
## 64   27.531762
## 65   21.975596
## 66   26.610769
## 67   28.663481
## 68   26.534582
## 69   32.622669
## 70   34.101971
## 71   28.728651
## 72   31.806852
## 73   30.507381
## 74   23.458605
## 75   33.575201
## 76   25.412200
## 77   35.492783
## 78   33.556421
## 79   23.281725
## 80   30.938442
## 81   24.628242
## 82   23.968088
## 83   30.886866
## 84   33.312532
## 85   23.507332
## 86   31.591445
## 87   25.820811
## 88   33.364705
## 89   21.973055
## 90   31.934126
## 91   31.137705
## 92   27.894069
## 93   28.283657
## 94   29.103208
## 95   26.568325
## 96   35.225372
## 97   28.497635
## 98   37.298256
## 99   35.073917
## 100  32.311718
```

```r
subset(myDF, category=="B", select = "expression") # both
```

```
##     expression
## 51    28.51747
## 52    29.47597
## 53    34.86205
## 54    30.53140
## 55    31.59210
## 56    32.04883
## 57    31.46124
## 58    24.52807
## 59    38.93141
## 60    33.20899
## 61    30.99487
## 62    29.04063
## 63    27.53516
## 64    27.53176
## 65    21.97560
## 66    26.61077
## 67    28.66348
## 68    26.53458
## 69    32.62267
## 70    34.10197
## 71    28.72865
## 72    31.80685
## 73    30.50738
## 74    23.45861
## 75    33.57520
## 76    25.41220
## 77    35.49278
## 78    33.55642
## 79    23.28172
## 80    30.93844
## 81    24.62824
## 82    23.96809
## 83    30.88687
## 84    33.31253
## 85    23.50733
## 86    31.59144
## 87    25.82081
## 88    33.36471
## 89    21.97306
## 90    31.93413
## 91    31.13771
## 92    27.89407
## 93    28.28366
## 94    29.10321
## 95    26.56832
## 96    35.22537
## 97    28.49763
## 98    37.29826
## 99    35.07392
## 100   32.31172
```

summarising data

Here we introduce the use of BASE functions aggregate and tapply, which are used to generate means, standard deviations etc of data by grouping variables (e.g. parts of experimental designs)

-NOTE - lots of people are using the package dplyr now with verbs `select()`, `slice()`, `filter()` along with `summarise()` and `group_by()`; VERY good tutorials online

```
# ----------------------------------------------------------------------
# Summarize the data using tapply and aggregate
# ----------------------------------------------------------------------
with(myDF,
    aggregate(x = expression, by = list(category), FUN = mean))
```

```
##   Group.1        x
## 1       A 10.60882
## 2       B 29.79817
```

```
# OR
aggregate(expression ~ category, data = myDF, FUN = mean)
```

```
##   category expression
## 1        A   10.60882
## 2        B   29.79817
```

```
#
with(myDF,
    tapply(X = expression, INDEX = list(category), FUN = mean))
```

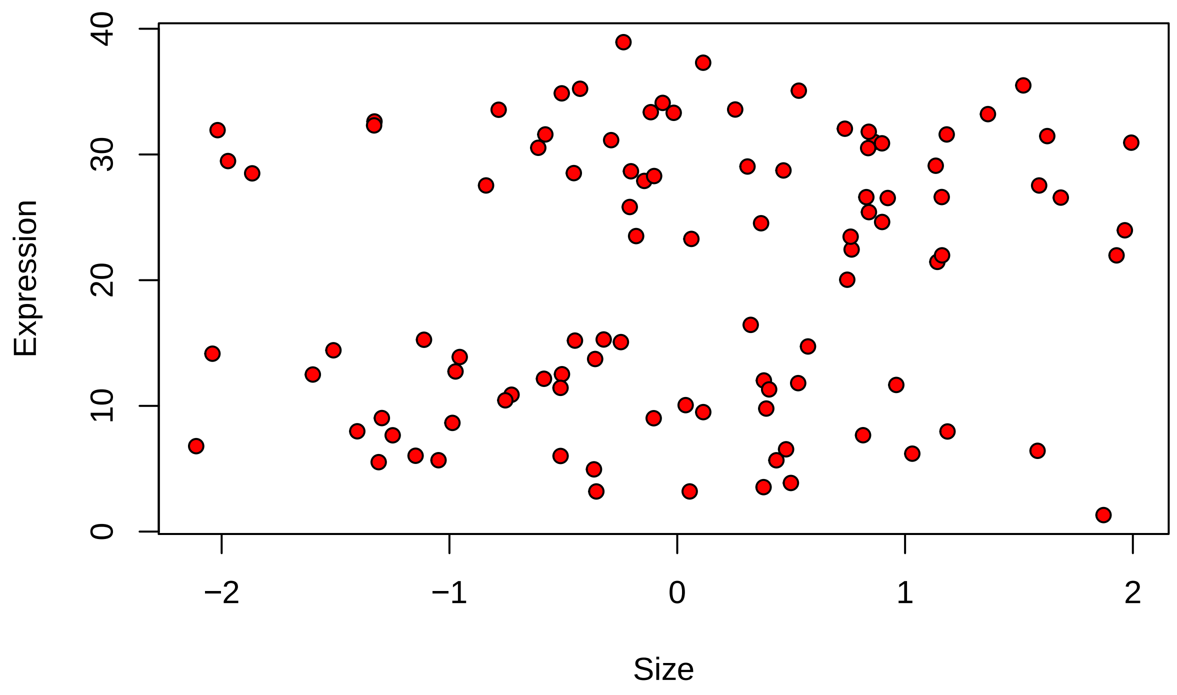
```
##        A        B
## 10.60882 29.79817
```

```
# YOU CAN assign the values returned by subset, aggregate and tapply to objects
mean.nums <- with(myDF,
    tapply(X = expression, INDEX = list(category), FUN = mean))
```

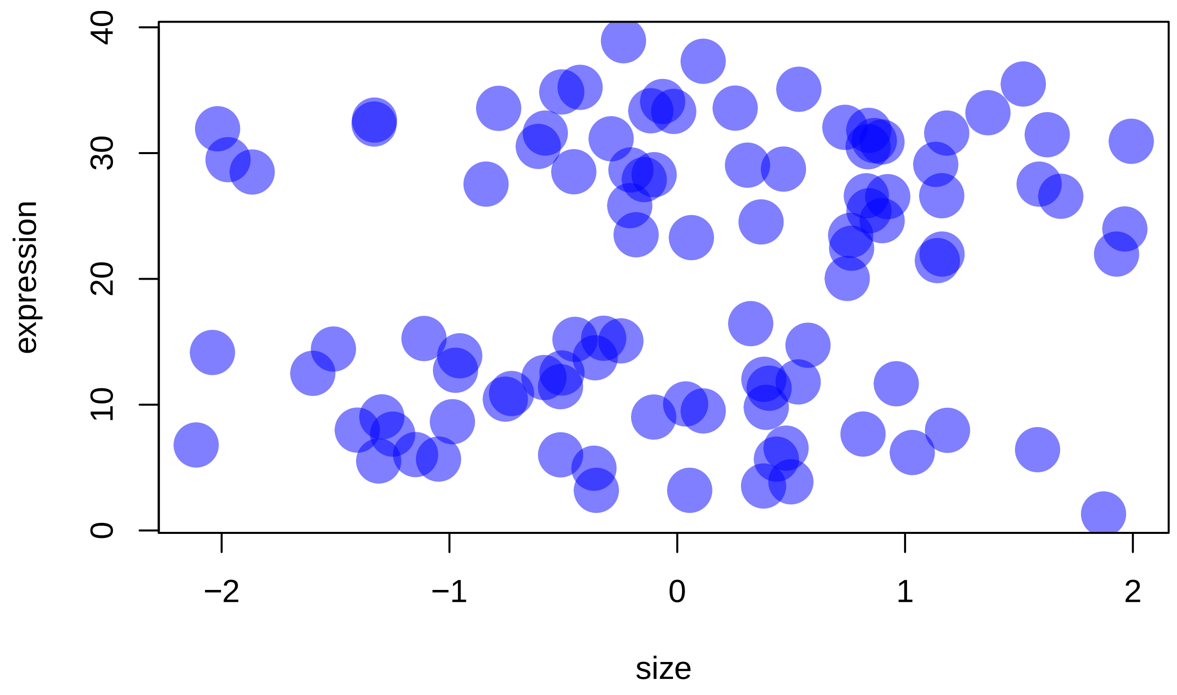
a more in depth overview of various plottings you can do

MANY methods for making plots with `omics data. - advanced use of`plot()`- introduction to`ggplot2`- introduction to`lattice‘

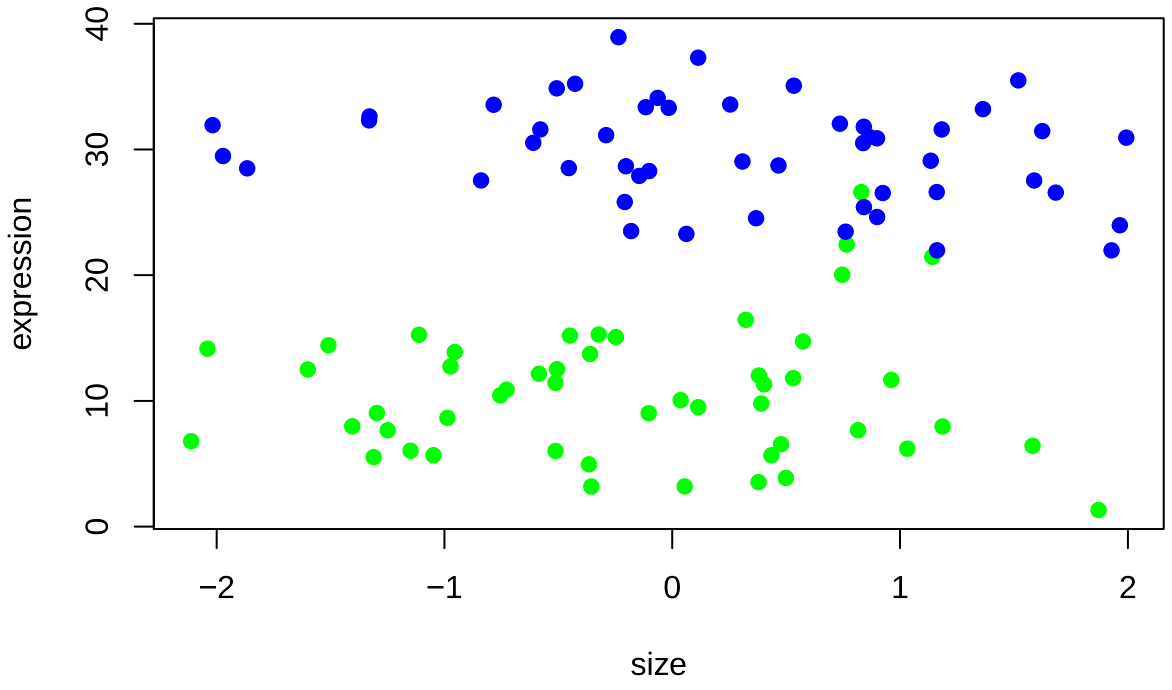
```
# ----------------------------------------------------------------------
# make some different types of plots
# plot, barplot, histograms, heatmaps
# reinforce function and arguments
# ----------------------------------------------------------------------

# plot, using the formula method - the best
plot(expression ~ size, data = myDF, pch = 21, bg = "red",
   xlab="Size", ylab = "Expression")
```



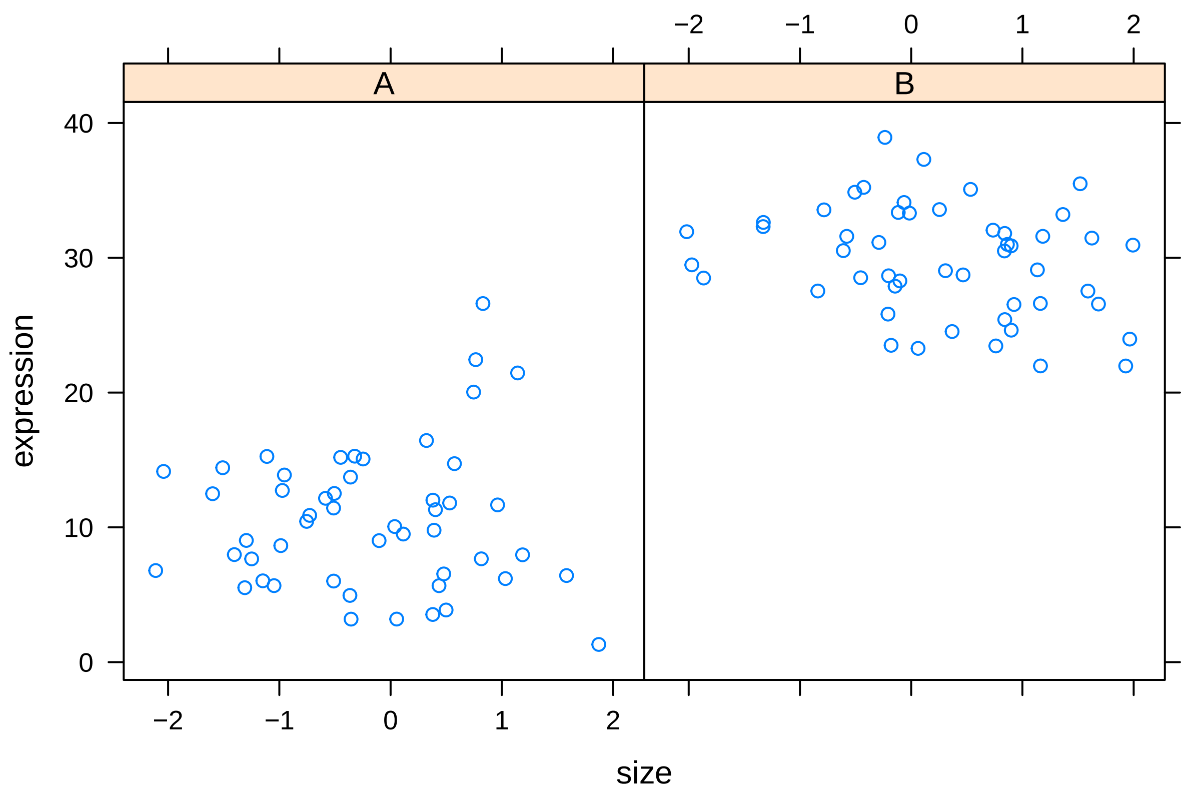
```r
# alter the color, and use alpha transparency
plot(expression ~ size, data = myDF, pch = 19, cex = 3, col = rgb(0,0,1,alpha = 0.5))
```



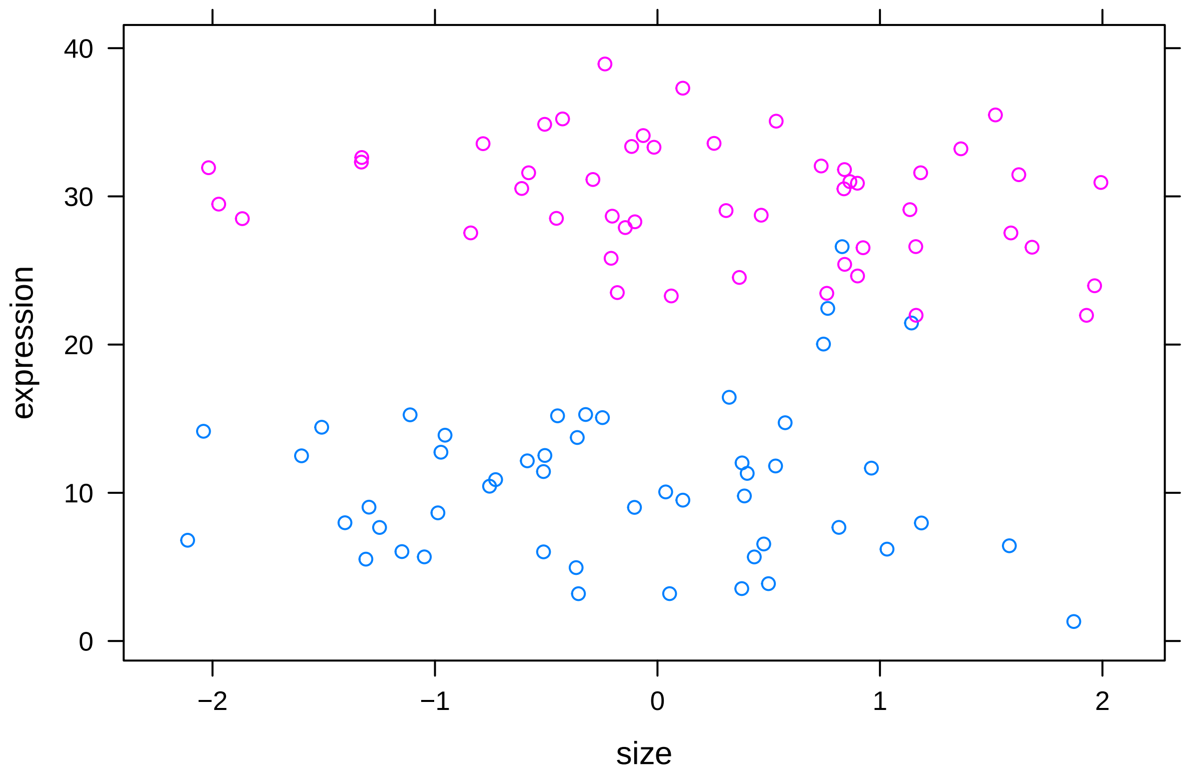
```r
# specifying categories of colours on the points based on a categorical column
# note the special use of [ ]'s
plot(expression ~ size, data = myDF,
    pch = 19, col = c("green","blue")[category])
```



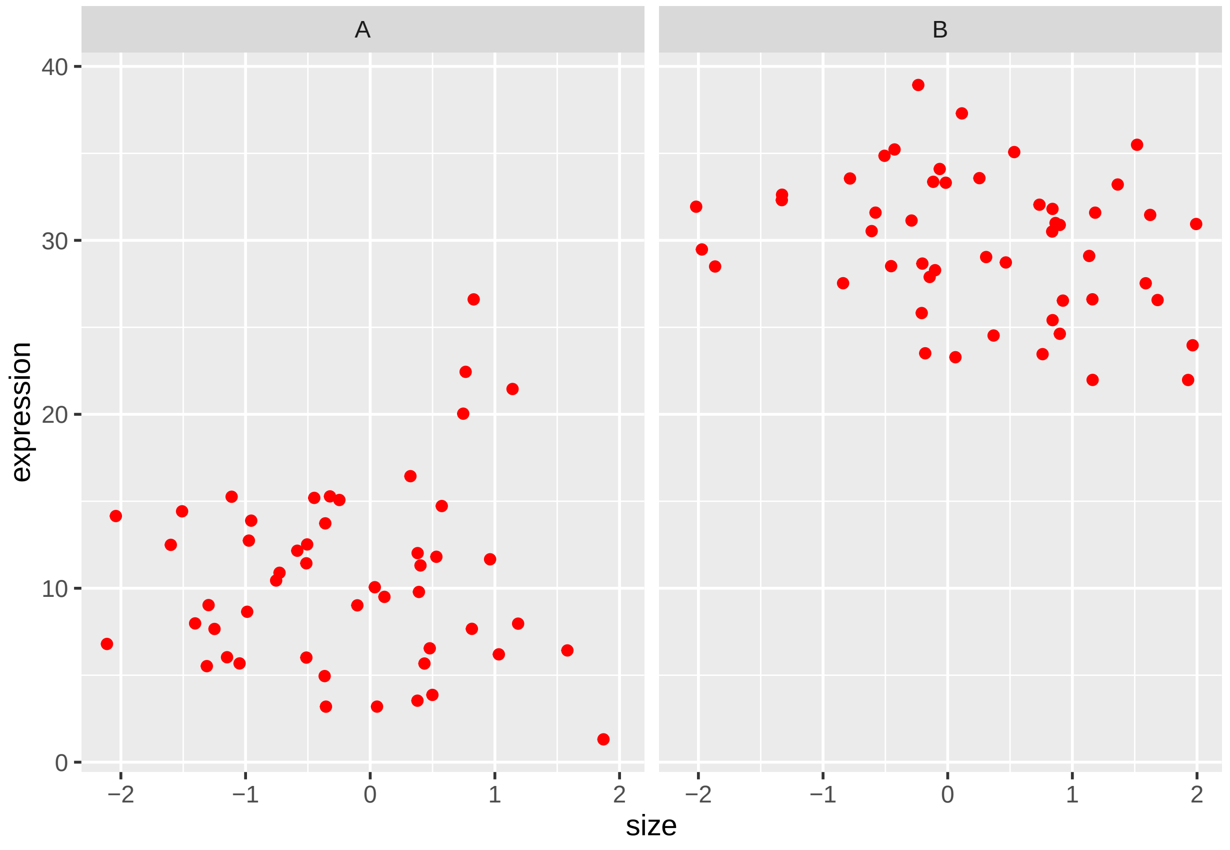
```
# lattice graphics - super fun exploration
library(lattice) # loads a special graphics library into R's brain

xyplot(expression ~ size | category, data = myDF) # panels
```



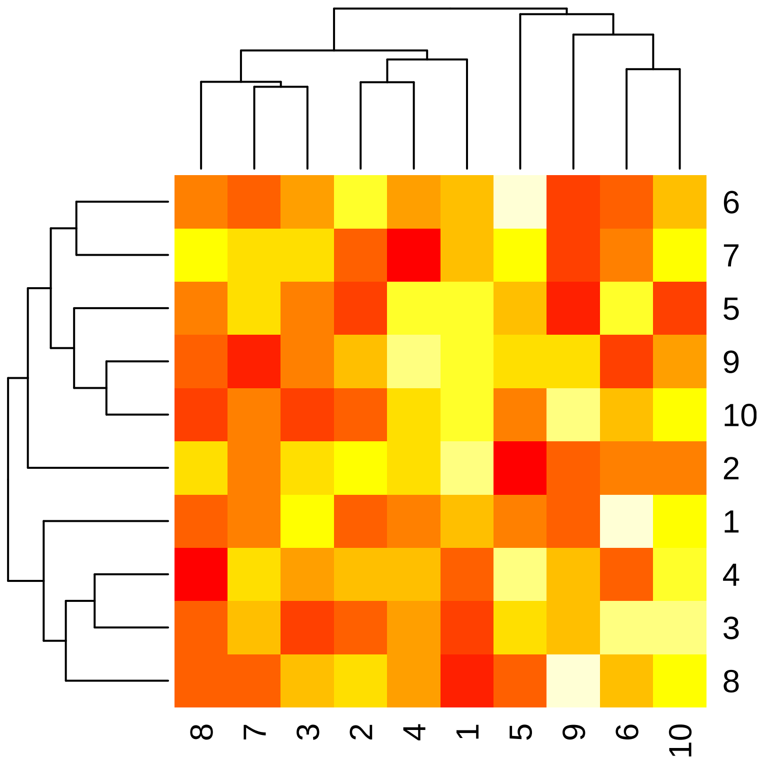
```
xyplot(expression ~ size, groups = category, data = myDF) # groups
```



```
# ggplot graphics - very popular now too
library(ggplot2)
ggplot(myDF, aes(x = size, y = expression))+
    geom_point(col = "red")+
    facet_wrap(~category)
```



```r
# heatmaps - very common in molecular data
?heatmap
heatmap(myList[[3]])
heatmap(myList$w)
```


6
7
5
9
10
2
1
4
3
8
8
7
3
2
4
1
5
9
6
10