An R Tutorial

Jo Hardin, Pomona College

Tuesday, September 02, 2014

1. Starting Out

R is an interactive environment for statistical computing and graphics. This tutorial will assume usage of R 2.15 on a PC. However, except in rare situations, these commands will work in R on UNIX and Macintosh machines as well as in S-Plus on any platform. R can be freely downloaded at http://www.r-project.org/.

You also need to download RStudio. After installing R (see above) on your computer, you will install RStudio, http://rstudio.org/.

Your assignmnets will be created using R Markdown. After writing the R code and relevant text, click the **Knit** button. A document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

2. Basics

Preliminaries

```
# A `\#' sign is considered a comment and will not be processed by R.
# You can press the up arrow key to rewrite the previous prompt.
# For most functions, R has useful help information.  For example, try help(cos)
# to see help on the cosine function.  From this help, you can read that the
# angle is in radians and that the function for arccos is called acos() in R.
```

Arithmetic

```
5                       # you type in a 5 at the prompt and press Enter
```

```
## [1] 5
```

```
4^ 3-2*7 + 9/2          # computes 4^3-(2*7)+(9/ 2). R follows the rules for the order of
```

```
## [1] 54.5
```

```
# opperations and ignores spaces \textit{between} numbers (or objects)
```

Objects

```
pi                # gives pi (rounded, of course)
```

```
## [1] 3.142
```

```
temp = 11/2       # Create an object called temp which will be stored in your workspace
temp              # now temp is an object you can use like pi (note that caps are important)
```

```
## [1] 5.5
```

```
ls()              # To list the objects you have created in your workspace
```

```
## [1] "metadata" "temp"
```

```
remove(temp)      # To remove an object from your directory
```

Vectors

```
x = c(3,4,7)         # c is concatenate.
x
```

```
## [1] 3 4 7
```

```
3*x-7                # use arithmetic on each number in the vector x
```

```
## [1]  2  5 14
```

```
x[2]                 # To view the second element of x.  In general, square brackets [ ]
```

```
## [1] 4
```

```
# are used to get an element from a vector or matrix
x[c(1,3)]            # To view the first and third elements of x.
```

```
## [1] 3 7
```

```
# x[1,3]             # Note that x[1,3] fails (so I can't type it in my markdown file!)
                     # In general, round brackets ( ) are used in functions like c()
x[-1]                # To view all but the 1st element of x
```

```
## [1] 4 7
```

Matrices

The last two arguments of the matrix function are the number of rows and columns,brespectively. If you would like the numbers to be entered by row instead of the default by column, add an additional argument, byrow=T. The matrix function is: matrix(vector of data, numrows, numcols, byrow=T)

```r
# An example of how to create a 2x3 matrix:
y = matrix(c(1,2,3,4,3,4),2,3)
y                                # To view the matrix, y
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    3
## [2,]    2    4    4
```

```r
y[1,]                            # To view the first row of y
```

```
## [1] 1 3 3
```

```r
y[,c(2,3)]                       # To view the second and third columns of y
```

```
##      [,1] [,2]
## [1,]    3    3
## [2,]    4    4
```

Modes

Most of the objects in your directory will be of one of three forms: vector, matrix (also calledban array), data.frame, or list. A vector means that it's a one dimensional matrix. A list means that itbcontains all sorts of thing (table, vector, array,… we'll see more later).

The following functions will help you figure out what time of object you are working with: *length, dim, mode, names*

If you have a list or a data.frame, *names* will tell you the names of the part of the list. *names* will also tell you the names of the columns in a data.frame, if the columns have been assigned names. If you have a numeric object, using *length* and *dim* will tell you whether you have a vector or a matrix.

```r
length(x)
```

```
## [1] 3
```

```r
dim(x)            # because x doesn't have at least 2 dimensions
```

```
## NULL
```

```r
length(y)         # the total length of y
```

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

```r
dim(y)            # two rows by three columns
```

```
## [1] 2 3
```

3. Plots

R has some very nice graphics capabilities. We'll only cover a minimal amount of information here.

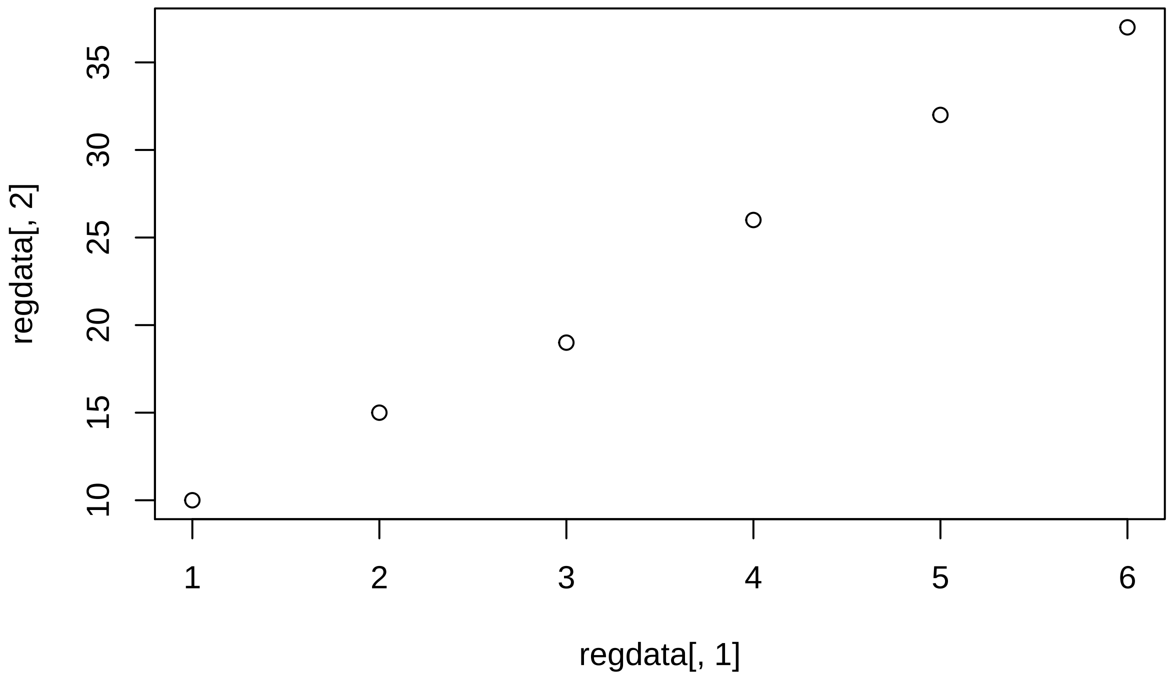
Scatterplots

```r
# Let's create a mock simple linear regression data matrix:
regdata = matrix(c(1:6,c(10,15,19,26,32,37)),6,2)
regdata
```

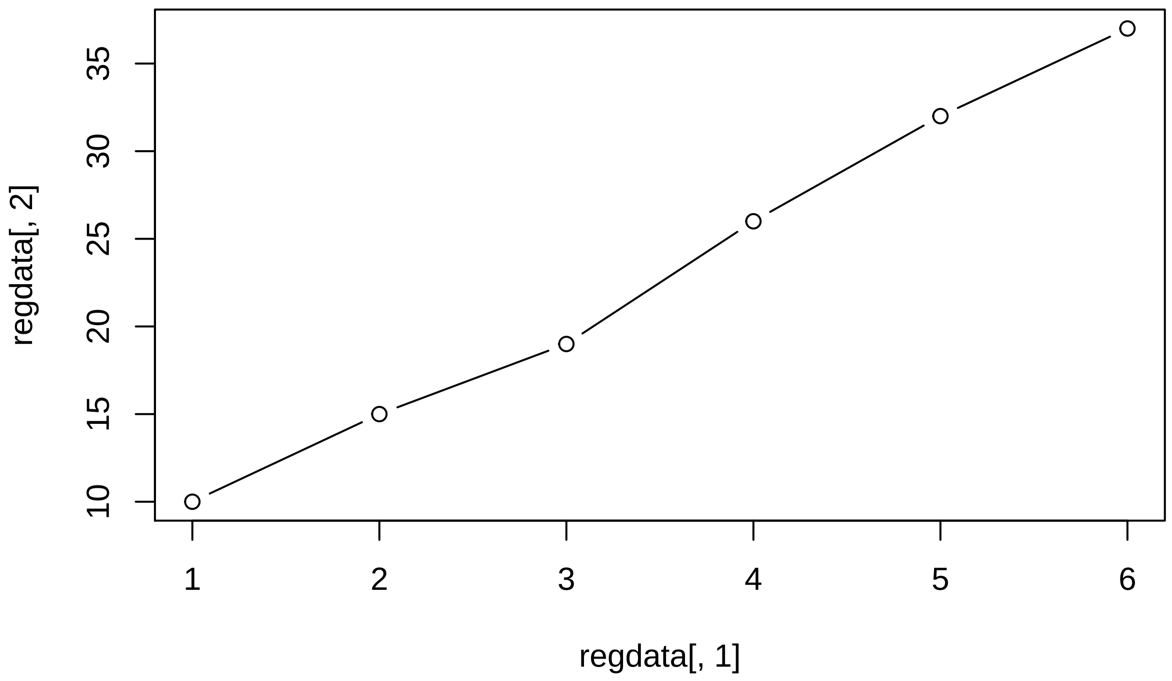
```
##      [,1] [,2]
## [1,]    1   10
## [2,]    2   15
## [3,]    3   19
## [4,]    4   26
## [5,]    5   32
## [6,]    6   37
```

Here, we will assume that the first column is the explanatory variable and the second column is the response variable.

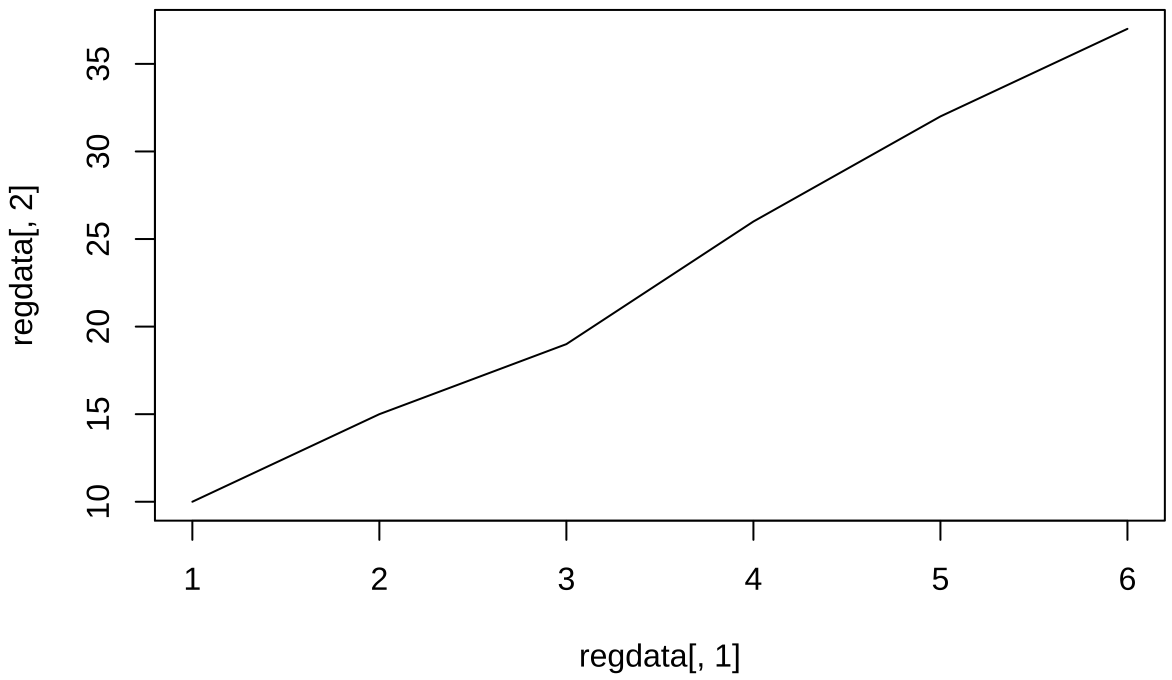
```r
# To plot the data points:
plot(regdata[,1], regdata[,2])     # the x-axis variable is the first argument
```



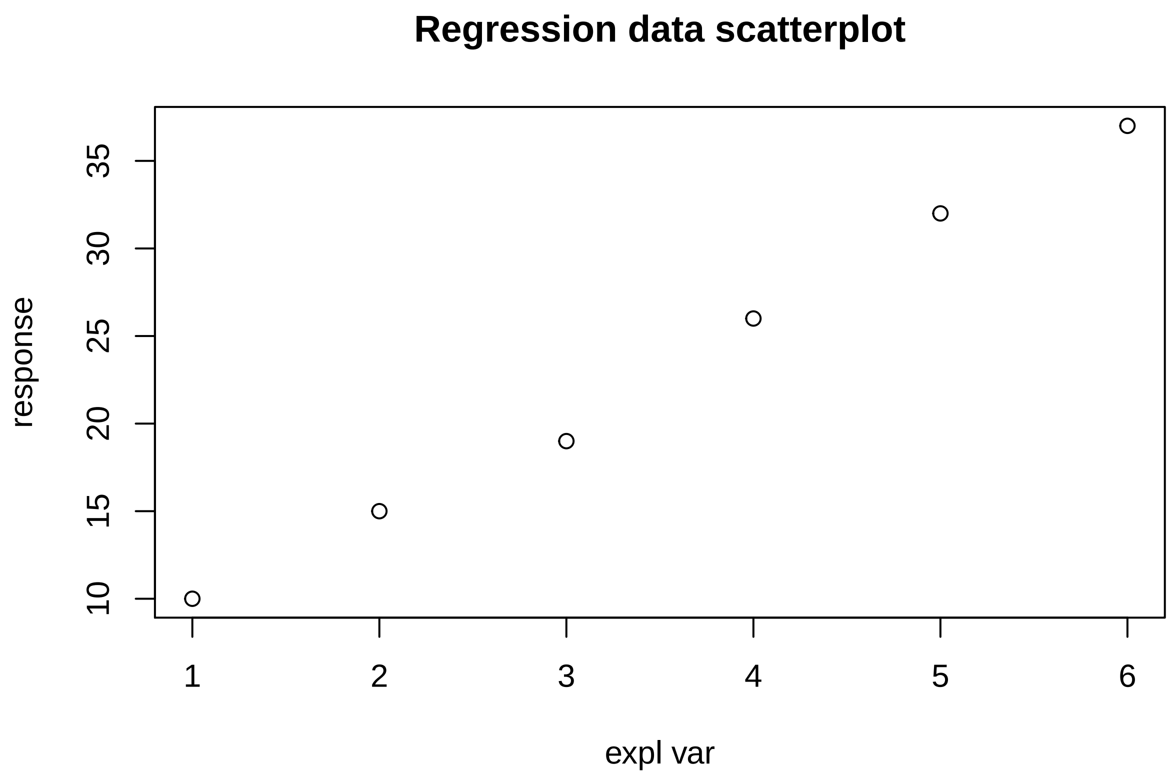
```r
# To plot the data points and have the points connected by lines:
plot(regdata[,1], regdata[,2], type='b')
```



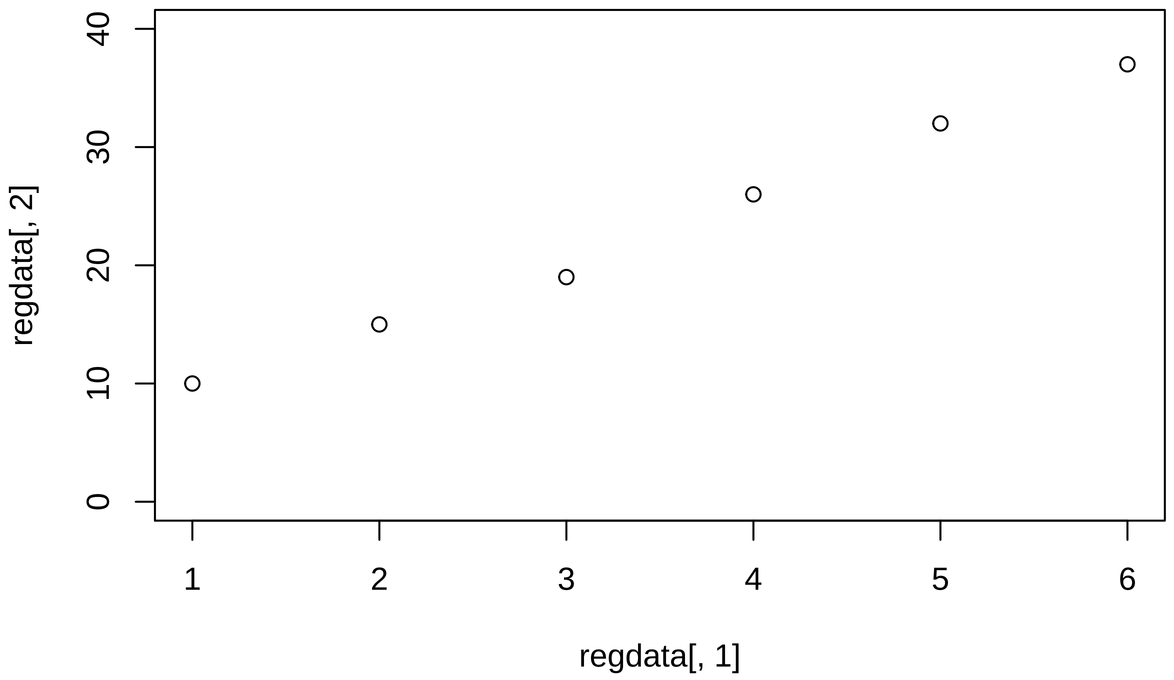
```r
# To plot only a line (the argument is the letter *l* not a number *1*)
plot(regdata[,1], regdata[,2], type='l')
```



```
# To add meaningful axis labels and a title to the scatterplot:
plot(regdata[,1], regdata[,2], xlab='expl var', ylab='response', main='Regression data scatterplot')
```

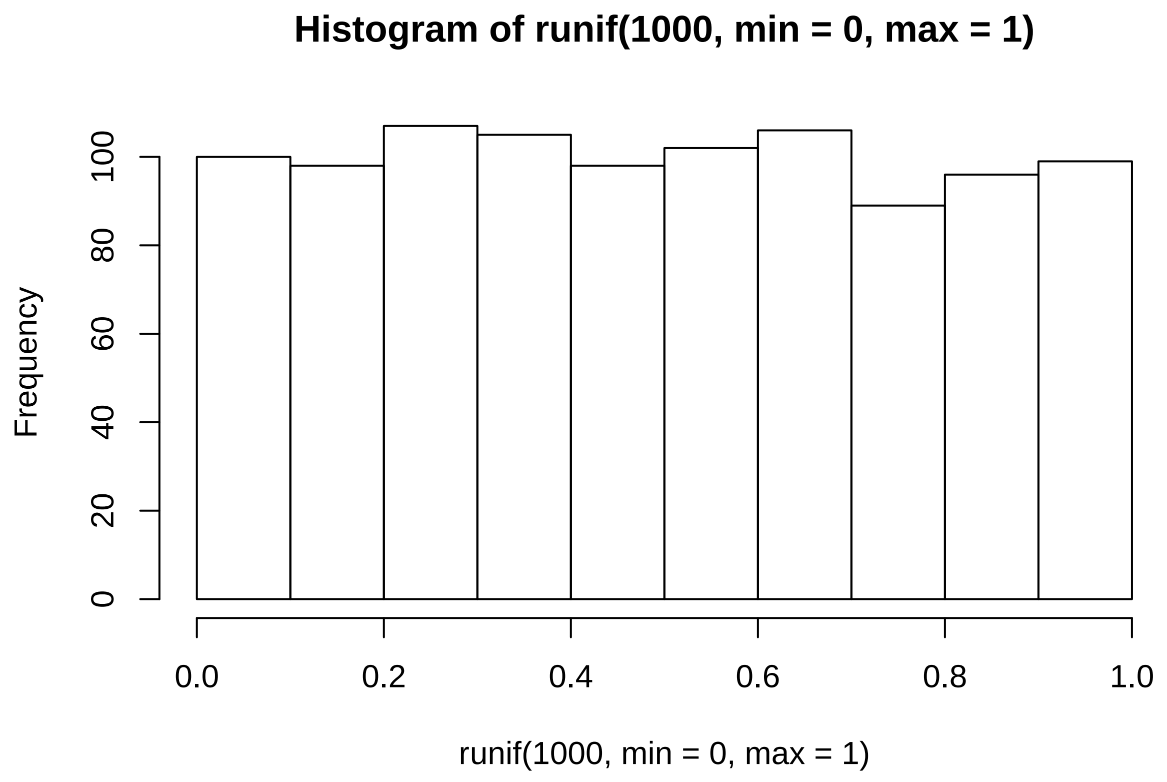


```r
# To set your own limits for the y-axis:
plot(regdata[,1], regdata[,2], ylim=c(0,40))
```

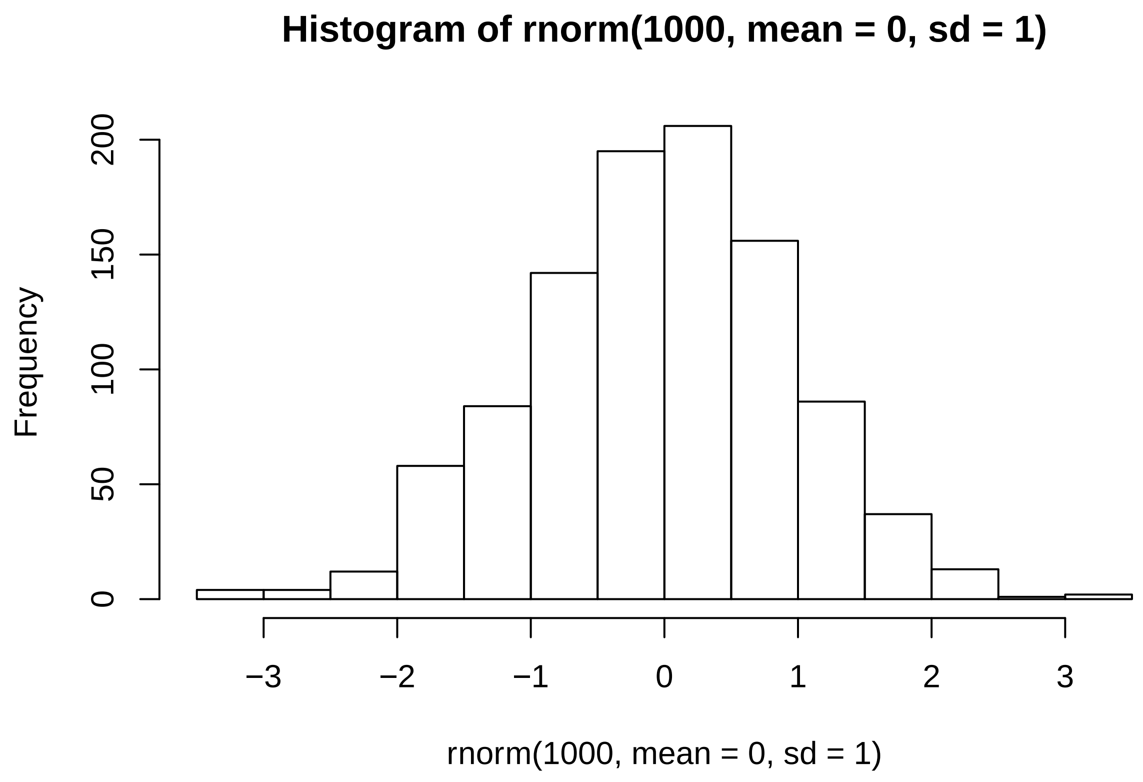


Histograms

```
# To plot a histogram of uniform(0,1) random data:
hist(runif(1000, min=0, max=1))
```



```r
# Or standard normal data:
hist(rnorm(1000, mean=0, sd=1))
```



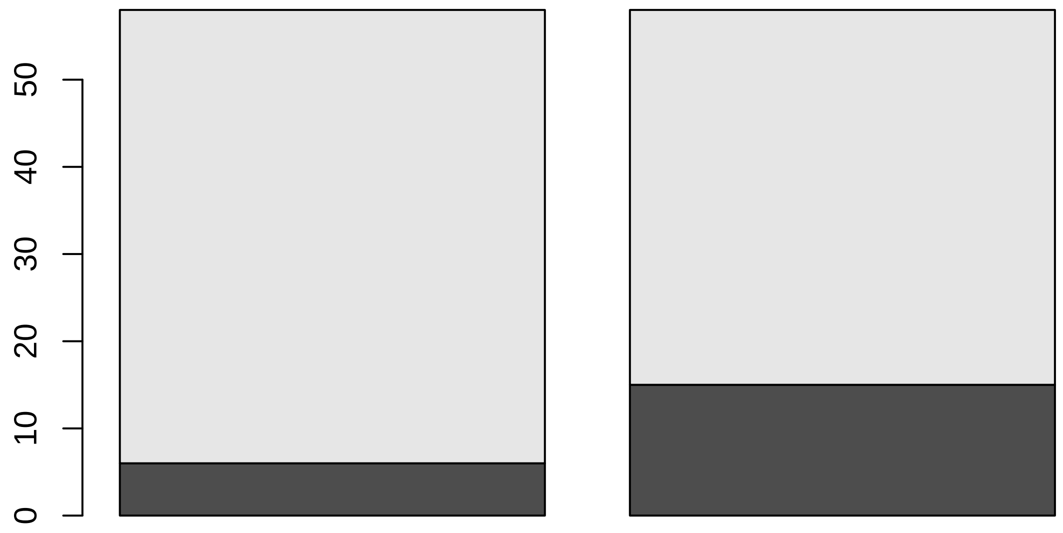
```
# The function *hist* allows for several useful options.  Type *help(hist)* for details.
```

Barplots

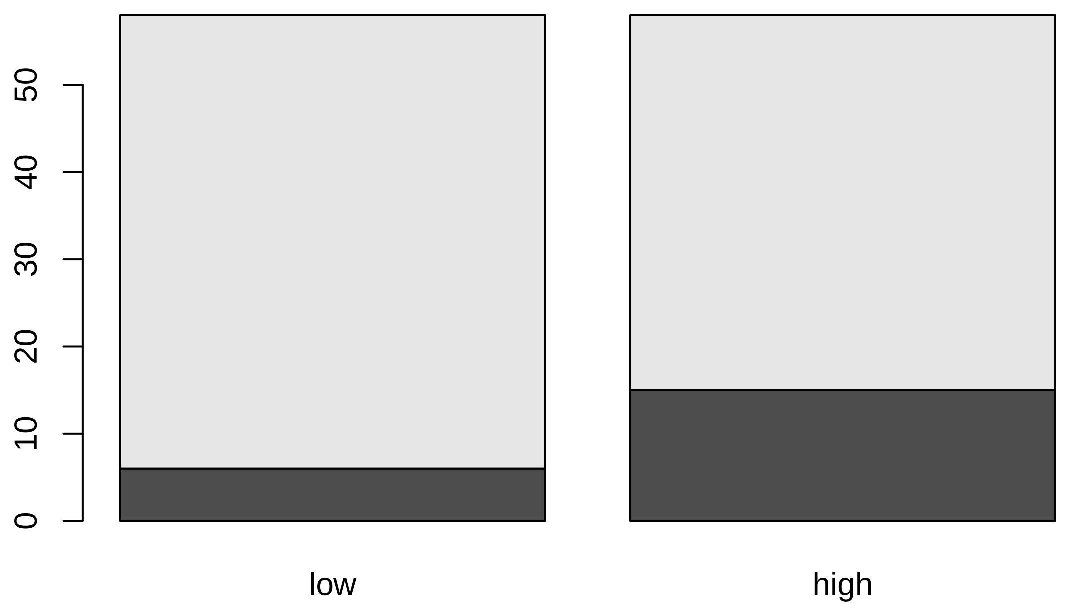
```
popcorn = matrix(c(6,52,15,43),ncol=2)
help(barplot)
```

```
## starting httpd help server ... done
```

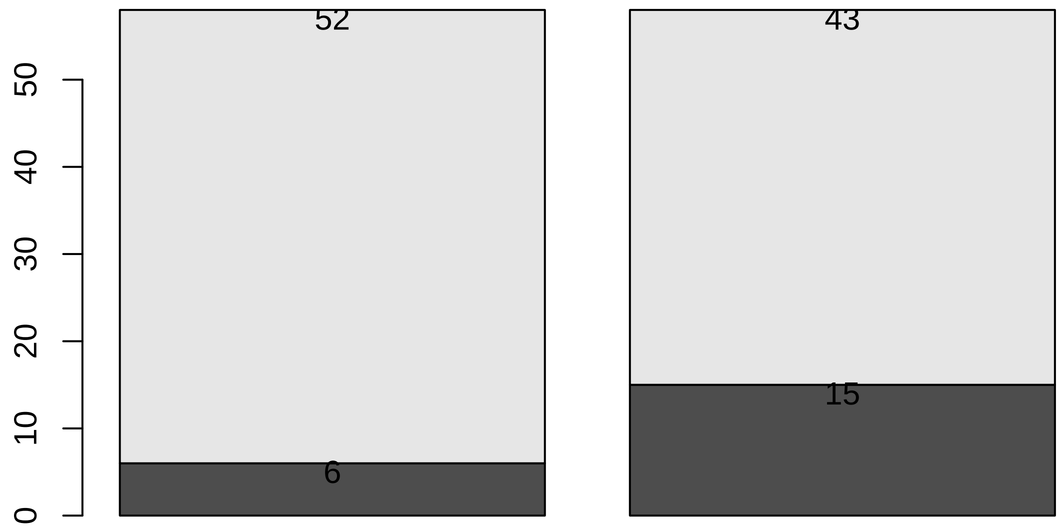
```
barplot(popcorn)                              # gives a simple barplot
```



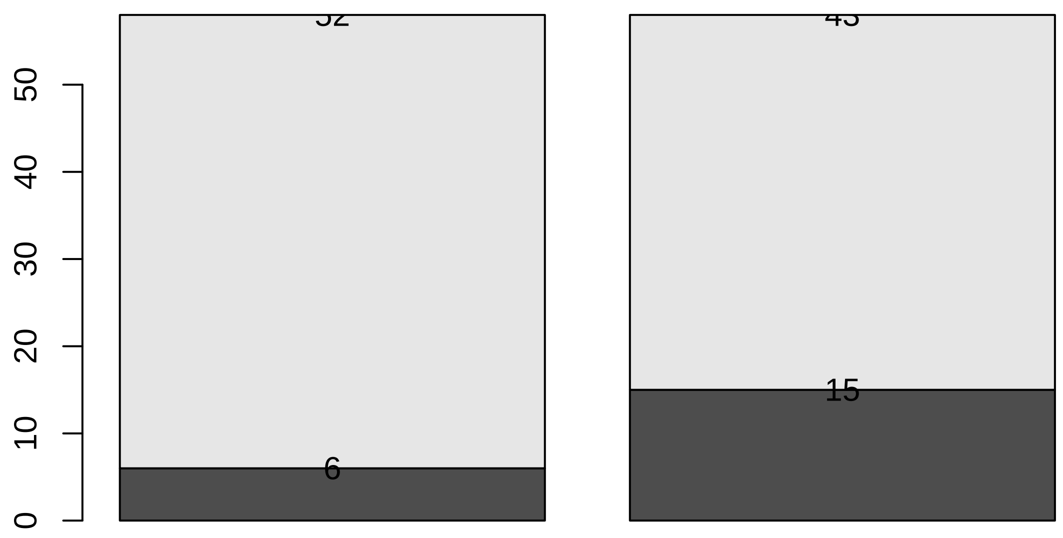
```r
barplot(popcorn,names.arg=c("low","high"))    # adds labels to the x-axis
```



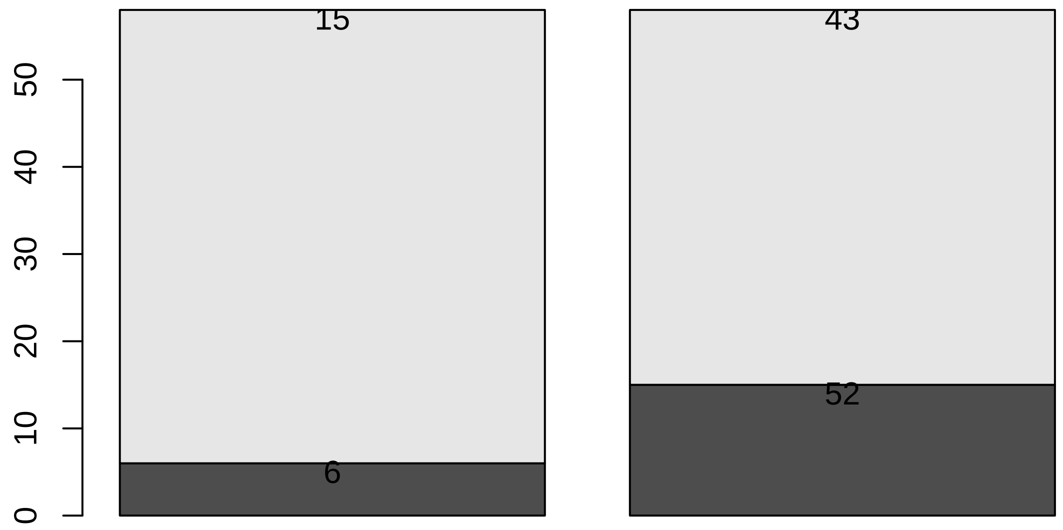
```
pop.bp<-barplot(popcorn)
# To add text to the plot:  the first argument is the barplot, the second is the y-axis
# values, and the third argument is the text you want to place on the barplot
text(pop.bp,c(popcorn[1,]-1,popcorn[2,]+popcorn[1,]-1) ,t(popcorn))
```



```
# Try removing the "-1" to see what you get:
barplot(popcorn)
text(pop.bp,c(popcorn[1,],popcorn[2,]+popcorn[1,]) ,t(popcorn))
```



```
# Or try not transposing the last argument, the "text" you want to put on the plot:
barplot(popcorn)
text(pop.bp,c(popcorn[1,]-1,popcorn[2,]+popcorn[1,]-1),popcorn)
```



4. Some Useful Feastures and Functions

```
 x = c(3,4,7)     # create our original object vector, x:
x >= 5            # indicates which elements in x are greater than or equal to 5\\
```

```
## [1] FALSE FALSE  TRUE
```

```
x[x >=5]          # produces all numbers in x that are greater than or equal to 5\\
```

```
## [1] 7
```

```
round(x/7,2)      # view only the first two decimal places of x/7\\
```

```
## [1] 0.43 0.57 1.00
```

Other Functions to Create Simple Vectors

```
# For most of these functions, use help(function) for additional information
seq(3,1,by=-.5)      # Creates a sequence from 3 to 1 counting by -.5
```

```
## [1] 3.0 2.5 2.0 1.5 1.0
```

```r
# look at help(seq) to see how else you can make a sequence
1:3                 # A shortcut for creating a sequence counting by 1
```

```
## [1] 1 2 3
```

```r
# note that 3:1 will count 3,2,1
rep(1,3)            # repeat 1 three times
```

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

```r
rep(1:3,2)          # repeat a sequence from 1 to 3 two times
```

```
## [1] 1 2 3 1 2 3
```

```r
# functions follow normal order of opperations i.e. stuff in ()s happens first
sort(rep(seq(1,2,length=3),2))  # the length parameter in seq() sets the length of the seq
```

```
## [1] 1.0 1.0 1.5 1.5 2.0 2.0
```

```r
# sort() arranges a vector small to large (by default)
rev(c(1,4,3))       # reverses the order of the elements in the vector
```

```
## [1] 3 4 1
```

```r
c(sort(c(seq(1,2,by=.2),1.7))[-c(2,3)],.7)  # make sure you can understand!
```

```
## [1] 1.0 1.6 1.7 1.8 2.0 0.7
```

```r
# remember that [-x] removes the x^th element
```

An Example (on cars)

```r
library(MASS)            # MASS is a library that contains datasets and functions
data(Cars93)             # We're going to use the Cars93 dataset
dim(Cars93)              # How big is the dataset?
```

```
## [1] 93 27
```

```r
length(Cars93)           # What is the difference between dim and length?
```

```
## [1] 27
```

```
names(Cars93)            # What are the variables in the Cars93 dataset?
```

```
##  [1] "Manufacturer"       "Model"              "Type"
##  [4] "Min.Price"          "Price"              "Max.Price"
##  [7] "MPG.city"           "MPG.highway"        "AirBags"
## [10] "DriveTrain"         "Cylinders"          "EngineSize"
## [13] "Horsepower"         "RPM"                "Rev.per.mile"
## [16] "Man.trans.avail"    "Fuel.tank.capacity" "Passengers"
## [19] "Length"             "Wheelbase"          "Width"
## [22] "Turn.circle"        "Rear.seat.room"     "Luggage.room"
## [25] "Weight"             "Origin"             "Make"
```

```
table(Cars93$Type)       # One way to see what types of cars are in the dataset
```

```
##
## Compact   Large Midsize   Small  Sporty     Van
##      16      11      22      21      14       9
```

```
table(Cars93[,3])        # "Type" happens to be the 3rd column of the dataset
```

```
##
## Compact   Large Midsize   Small  Sporty     Van
##      16      11      22      21      14       9
```

```
table(Cars93$AirBags)    # But what if we want to know which types of cars have airbags?
```

```
##
## Driver & Passenger        Driver only               None
##                 16                 43                 34
```

```
Cars93[1:10,c(3,9)]      # If our interest is in type vs. airbag
```

```
##       Type            AirBags
## 1    Small               None
## 2  Midsize Driver & Passenger
## 3  Compact        Driver only
## 4  Midsize Driver & Passenger
## 5  Midsize        Driver only
## 6  Midsize        Driver only
## 7    Large        Driver only
## 8    Large        Driver only
## 9  Midsize        Driver only
## 10   Large        Driver only
```

```
table(Cars93[,c(3,9)])   # Which type of car seems most likely to have an airbag (in 1993)?
```

```
##          AirBags
## Type      Driver & Passenger Driver only None
##   Compact                  2           9    5
##   Large                    4           7    0
##   Midsize                  7          11    4
##   Small                    0           5   16
##   Sporty                   3           8    3
##   Van                      0           3    6
```

Functions to Analyze Vectors

```
z = c(1,4,3)
length(z)                         # Determine the length of the vector z:
```

```
## [1] 3
```

```
c(max(z), sort(z)[length(z)])     # Find the maximum value of z in two ways:
```

```
## [1] 4 4
```

```
# note that the answer is a vector because I concatenated the solutions of two functions
rank(z)                           # Identify the rank of the elements of z:
```

```
## [1] 1 3 2
```

```
# i.e., the 1st element is the min and the 2nd element is the max
z[rank(z)==2]                     # Which is the second smallest element in z
```

```
## [1] 3
```

```
# or you can also say \verb!sort(z)[2]!
sample(z,10,replace=T)            # To randomly sample from an existing vector:
```

```
##  [1] 4 3 1 3 1 1 1 3 3 3
```

```
# This is a random sample with replacement of z
sample(1:500,4,replace=F)         # Randomly sample from a sequence from 1 to 500:
```

```
## [1] 254 420 243  59
```

```
# replace=F means the same number won't be sampled twice
x = c(3,4,7)
rbind(x,z)                        # rbind binds by rows to make a matrix
```

```
##   [,1] [,2] [,3]
## x    3    4    7
## z    1    4    3
```

```
cbind(x,z)                         # cbind binds by columns (treat x and z as columns)
```

```
##      x z
## [1,] 3 1
## [2,] 4 4
## [3,] 7 3
```

Dealing with Data and Data Files

```
# To read in an external dataset called data.txt:
# tempfile <-scan(`data.txt')=
#
# Notice that scan brings the data into R in a vector format regardless of the original format.
# To read in an external dataset in a more flexible format use *read.table( )*.
# Check *help(read.table)*.  Also, note that both *read.table()* and *scan()*
# have several options which are useful for datasets with interesting features, such as the
# column names on the first row, spacing by special symbols, etc.
#
# Writing data to a file, tempout1, by column:
#write(u, file=`tempout1')
# Writing data to a file, tempout2, by row:
#write(t(u), file=`tempout2')
# Writing another vector to the same file by row:
# write(t(x), file=`tempout2', append=T)
```

More Useful Functions

For additional information on any of these functions use help(function) i.e. *help(log)* (note the default of the *log* function is base *e*!).

sin, cos, tan, asin, acos, atan log, log10, exp min, median, max, quantile sum, prod var, sd, cov, cor union, intersect t (for transposing a matrix) solve (for the inverse of a matrix) diag (for the diagonal of a matix)

Getting Help

Most importantly, please email me if you are stuck!! Additionally, my website has links to 3 R manuals that you may find helpful.

The R site has some useful information, but it isn't always easy to navigate. Try:\ {`www.r-project.org`}\ {`http://cran.r-project.org/doc/contrib/refcard.pdf`}

- `Minimal R for Intro Stats http://cran.r-project.org/web/packages/mosaic/vignettes/V1MinimalR.pdf`
- `Commands for Intro Stats http://cran.r-project.org/web/packages/mosaic/vignettes/V3Commands.pdf`
- `Venables, Smith, & R Team (2009).` *`{An Introduction to R (2ed)}.  Network Theory Limited.`*
- *`Everitt, B., Hothorn T. (2006).`* `{A Handbook of Statistical Analyses Using R.} Champman & Hall.`
- `Dalgaard P. (2002).` *`{Introductory Statistics with R.} New York:  Springer-Verlag.`*

- *Ripley B.D., Venables W.N. (2002).* {Modern Applied Statistics with S (4ed).} New York: Springer-Verlag.

Using Help

As mentioned above, help can be found on any function by using, *help(function)*. The help files are all formatted in the same way: + Description: is a few words describing what the function does + Useage: gives the actual function, and what the inputs (i.e., arguments) to the function are + Arguments: specifies the format of the arguments. This section also gives the default values (if applicable) for the arguments + Details: gives any additional relevant information about the function + Value: specifies the output of the function. Sometimes the output is a plot, sometimes a matrix, sometimes a list of information, sometimes either a plot or a matrix of information (e.g., *hist*) + References: where to get more information about the function + See Also: other R functions that are related + Examples: examples to try. If you don't understand what the function is doing / what you should input / what the format of the output is, you should walk through the examples. All the examples should use data and functions that are internal to R; that is, simply copy and paste the example into the R session.