Introduction to R

Madelaine Gogol

Computational Biology

October 2011

[Introduction 1](#_Toc308171322)

[1. Getting Started 2](#_Toc308171324)

[2. R as a calculator 4](#_Toc308171335)

[3. Variables 5](#_Toc308171337)

[4. Vectors 5](#_Toc308171338)

[5. Data from Files 8](#_Toc308171341)

[6. Basic Plotting 10](#_Toc308171344)

[7. For loops 16](#_Toc308171351)

[8. Multifigure Plotting 16](#_Toc308171352)

[9. Getting plots out of R 17](#_Toc308171353)

[10. Boxplots 18](#_Toc308171354)

[11. Linear Regression 19](#_Toc308171356)

[12. Writing data out 20](#_Toc308171358)

[13. T-test 21](#_Toc308171359)

[14. Packages 21](#_Toc308171360)

[15. Colors 22](#_Toc308171361)

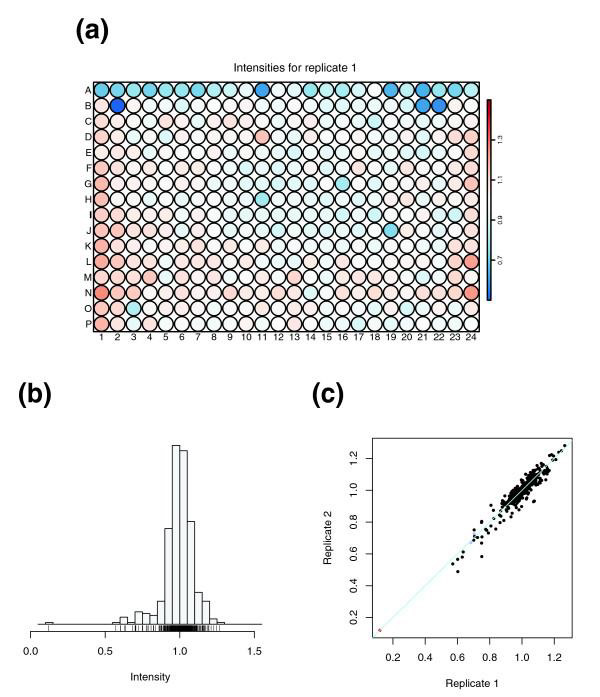
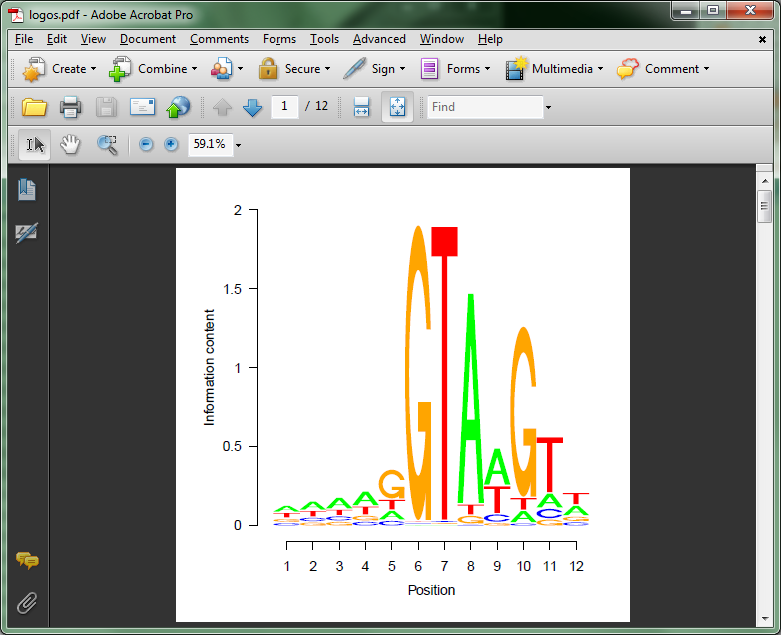
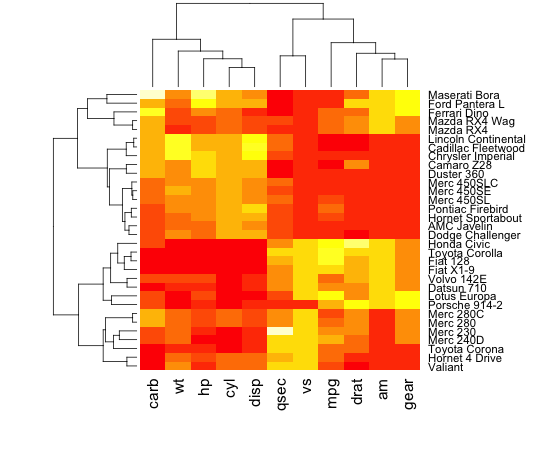
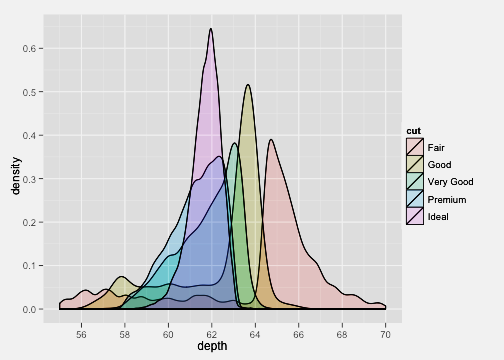
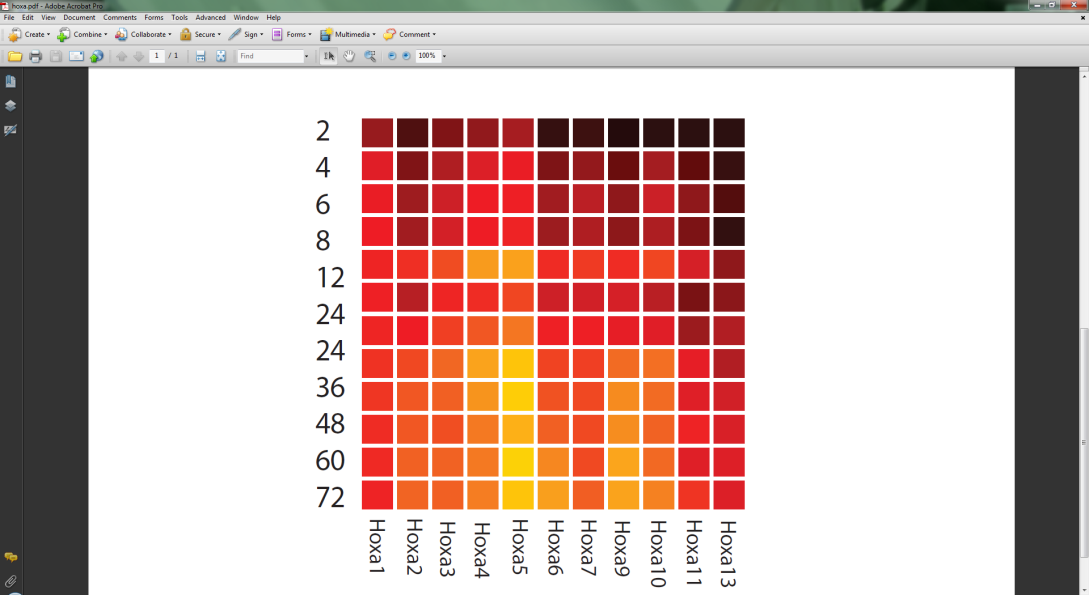
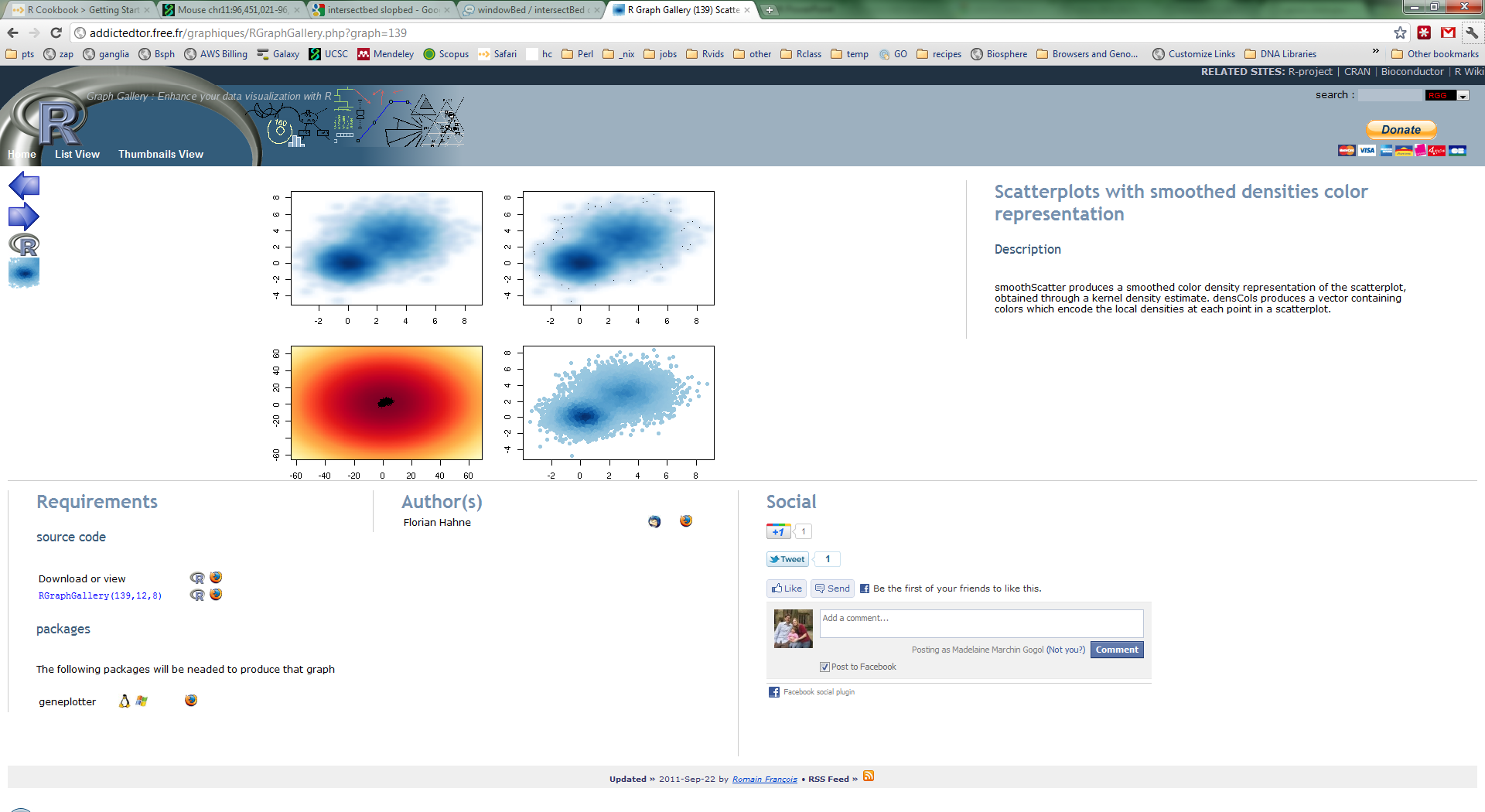
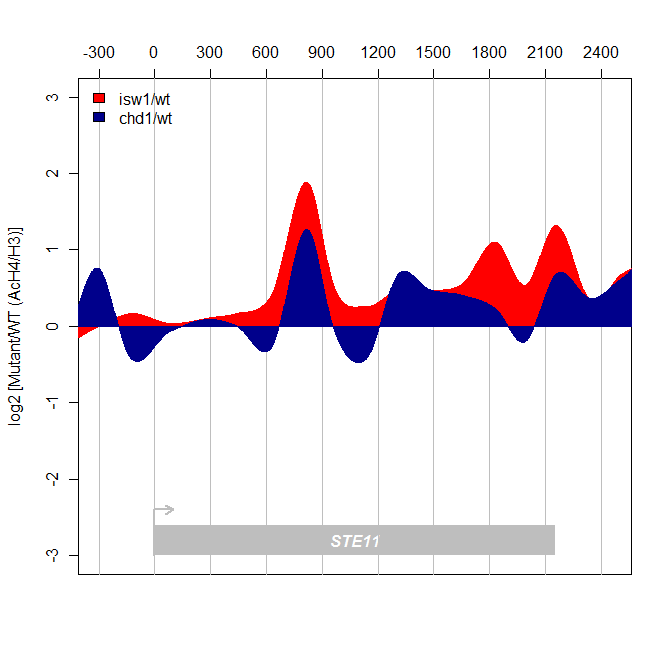
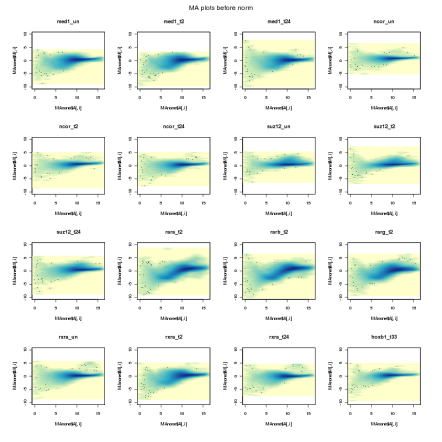
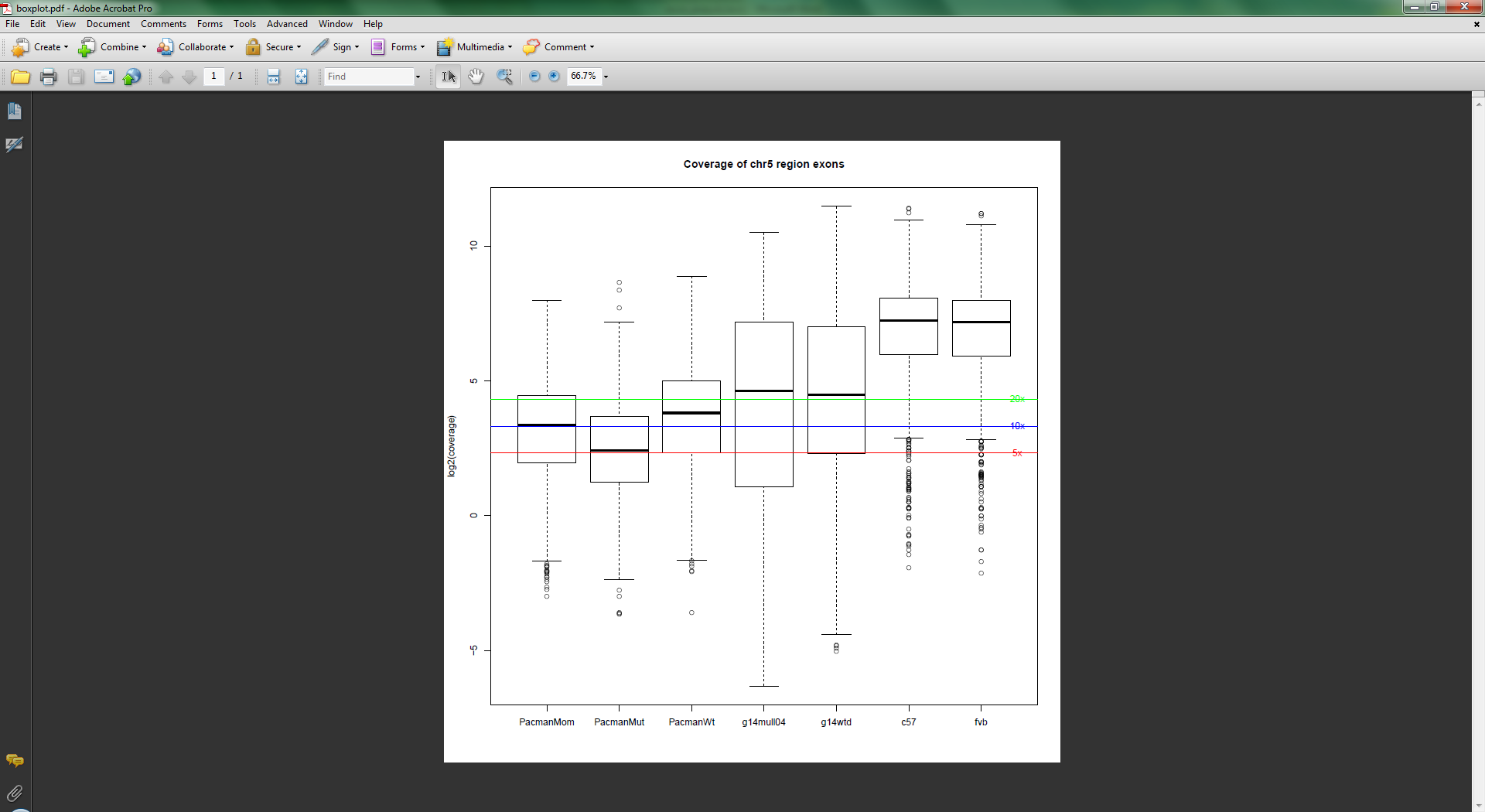
[16. Heat maps 24](#_Toc308171362)

# Introduction

R is a language for statistical data analysis and graphics. It is free and runs on Windows, Mac, or Linux. Two reasons to learn R:

(1) Powerful plotting and visualization capabilities.

(2) Packages written for data analysis (especially for biological data)



# 1. Getting Started

## Download and install R

<http://www.r-project.org>

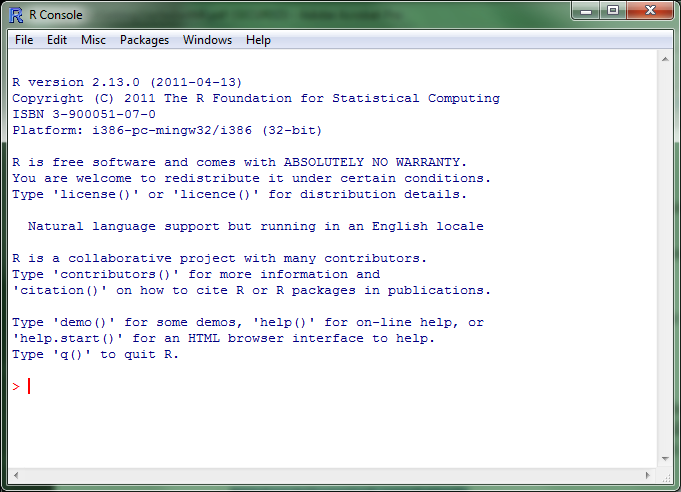
Select a mirror site near you, select Windows / Mac, download and Install R (base)

Optional: Download, install, and run Rstudio

<http://rstudio.org/download/desktop>

## R console

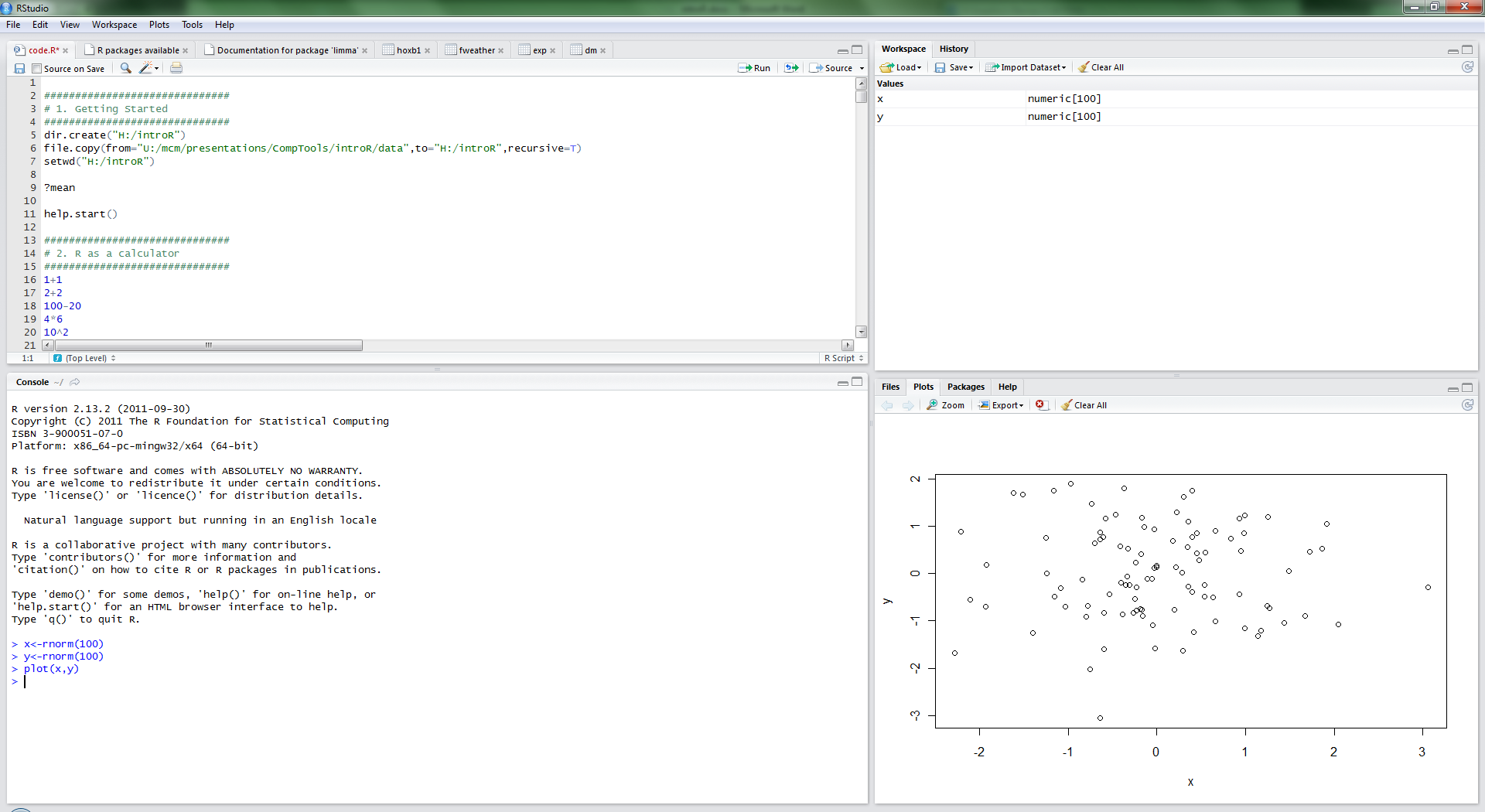
This is the R console if you run it by itself.



Type or paste code here.

## Rstudio

This is the Rstudio integrated development environment.



Plots or help shows up here.

This is the console, where stuff is executed. Up arrow to view past entries.

Data shows up here as you create it.

This is your text file with code. Ctrl-Enter to send a line or selection to the console.

## Working with R

Typically when working with R, you will have some code you are working on in a text file in one window and the R console will be open in a separate window. You will usually write your code in the text editor window and then copy / paste / send it into the R console to run. I will be showing you how to do this using Rstudio, but other options include Tinn-R or Wordpad (or your text editor of choice).

## Set up for this tutorial

In this document, anything you see in blue should be what you enter into the R console, and in black will be the R output.

For the purposes of this tutorial (this only applies to the Stowers computing environment), the following commands will create a folder on your H drive and copy some data to it that we'll be using in the tutorial. Paste these into your R console.

dir.create("H:/introR")

file.copy(from="U:/mcm/presentations/CompTools/introR/data",to="H:/introR",recursive=T)

file.copy(from="U:/mcm/presentations/CompTools/introR/code/code.R",to="H:/introR")

file.copy(from="U:/mcm/presentations/CompTools/introR/introR.docx",to="H:/introR")

If you have a mac, you'll need to mount your HOME drive (Finder - GO - Connect to Server) and the projects drive, then paste the following into R:

dir.create("/Volumes/HOME/introR")

file.copy(from="/Volumes/projects/mcm/presentations/CompTools/introR/data",to="/Volumes/HOME/introR",recursive=T)

file.copy(from="/Volumes/projects/mcm/presentations/CompTools/introR/code/code.R",to="/Volumes/HOME/introR")

file.copy(from="/Volumes/projects/mcm/presentations/CompTools/introR/introR.docx",to="/Volumes/HOME/introR")

## Set your working directory

Your working directory will be where any files you generate end up. It can also be convenient to be the location of any data you might be reading in along with the location of any R script you are currently working on.

setwd("H:/introR")

#or mac

setwd("/Volumes/HOME/introR")

If you don’t explicitly set your working directory on windows, it will be set to H:\My Documents (at Stowers), C:\Users\mcm\Documents (Windows Vista, Windows 7), or /Users/mcm (Mac).

## Get help on a function

*Try asking for help on setwd*

?mean

## Browse Documentation

help.start()

Learning a Programming Language

If this is the first time you've learned a programming language, prepare to be tripped up for awhile by the precision required when typing. Leaving off a ), }, ], " or making small errors will likely trip you up at first and take some getting used to. Ask for help if you are getting an error you don't understand. Programming is fun and creative, but also a continual exercise in humility.

You should not expect to know R by the end of this class (2 sessions of 1.5 hours each). My goal is to give you the basic framework and tools that will allow you to do a few things and continue learning. We'll learn R basics and create a few different types of plots - scatter plots, barplots, and heatmaps.

Definitions

A variable in R can be thought of as a named bucket that could contain any number of things - a single number, a string, a vector, or a data frame (table of data).

A string is a sequence of characters. Strings are surrounded by quotes and can include letters, numbers, spaces, or symbols.

A vector in R is a linear data structure. You can think of it as a series of numbered buckets, each of which contains a number, or a string, or a logical value (TRUE or FALSE)

A function in R is like a command - you give it input, it gives you a result. Functions are structured like name(input), where the name of the function is followed immediately by parentheses containing its inputs.

An argument is something you pass as input to a function. so when you say log2(8), 8 is the argument and log2 is the function. Or when you say setwd("H:/introR"), "H:/introR" is the argument.

# 2. R as a calculator

You can use R like a simple calculator:

1+1

[1] 2

Try some of these:

100-20

4\*6

10^2

100/4

sqrt(2)

log2(2)

Remember, to get help on a function, type question mark followed by function name

?log2

*What is 2 to the 12th?*

*What is the log (base 10) of 10000?*

# 3. Variables

A variable can be thought of as a named container for data. The simplest type of data is a single element, which could be a number, a text string, or a TRUE/FALSE value. To assign a value to a variable, use the <- assignment operator.

m <- 5

Now type m and you will see the value of m (what m contains).

m

[1] 5

n <- 7.45

mytext <- "hello"

In these examples, m, n, and mytext are all variable names and the values being assigned to them are 5, 7.45, and the text "hello".

# 4. Vectors

You can use the *c()* function to **combine** single elements into a vector. The things you want to combine are arguments to the function, and should be separated by commas.

c(1,3,6,13,8)

[1] 1 3 6 13 8

use the assignment operator "*<-*" to assign the vector to the variable x. I could have called x many other things here, like "heights" or "numbers" or "myvector".

x <- c(1,3,6,13,8)

When you assign the vector to x, it no longer shows up on your screen. To see what x contains, type x, then enter. Note: when you put a # in front of a line, it becomes a comment and is ignored by R.

x

[1] 1 3 6 13 8

#create another vector, y.

y <- c(2,5,4,12,7)

To see what y contains, type y, then enter.

y

[1] 2 5 4 12 7

To access a specific element of a vector, use the square brackets - to access the fifth element of x, do:

x[5]

[1] 8

Create some more vectors:

vector1 <- c("hi","how","are","you")

vector2 <- c("Hi, how are you?", "I am fine.")

#note -- TRUE and FALSE are special words in R, don't use quotes.

vector3 <- c(TRUE,TRUE,FALSE,TRUE,TRUE)

There is a shortcut that will come in handy later to create a vector with integers from one number to another.

1:10

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

To store that in a variable, you would do

m <- 1:10

*try creating a vector of words of your choosing*

*create a vector of the numbers from 5 to 20, save it to a variable with a name of your choosing.*

## Calculate some basic statistics on vectors

mean(x)

[1] 6.2

median(x)

[1] 6

min(y)

[1] 2

max(y)

[1] 12

#which.min returns the POSITION of the minimum element in the vector. The minimum element in vector y can be found at position 1. The minimum element in vector y can be found at position 5.

which.min(y)

[1] 1

which.max(y)

[1] 5

#sample standard deviation

sd(x)

[1] 4.658326

summary(x)

Min. 1st Qu.Median Mean 3rd Qu. Max.

1.0 3.0 6.0 6.2 8.0 13.0

#correlation

cor(x,y)

[1] 0.9442803

length(y)

[1] 5

Two equals signs is a special operator meaning "is equal to". So x==6 will return a logical (TRUE/FALSE) vector the same size as x, with a TRUE everywhere that x is equal to 6.

x==6

[1] FALSE FALSE TRUE FALSE FALSE

To get the position of all TRUE elements, use *which().*

which(x==6)

#you can also use greater than or less than, <=, >=.

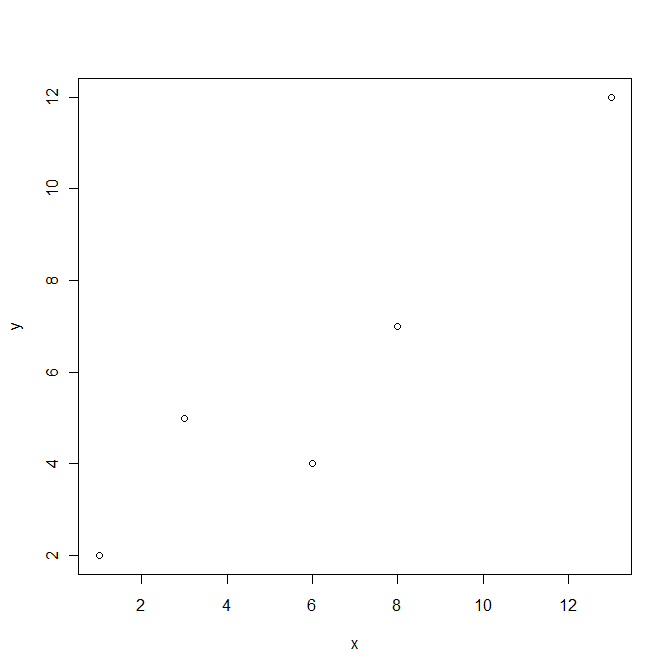
which(x<3)

*what is the median of numbers 4 to 7?*

*which elements of y are greater than 5?*

## Plot x vs y

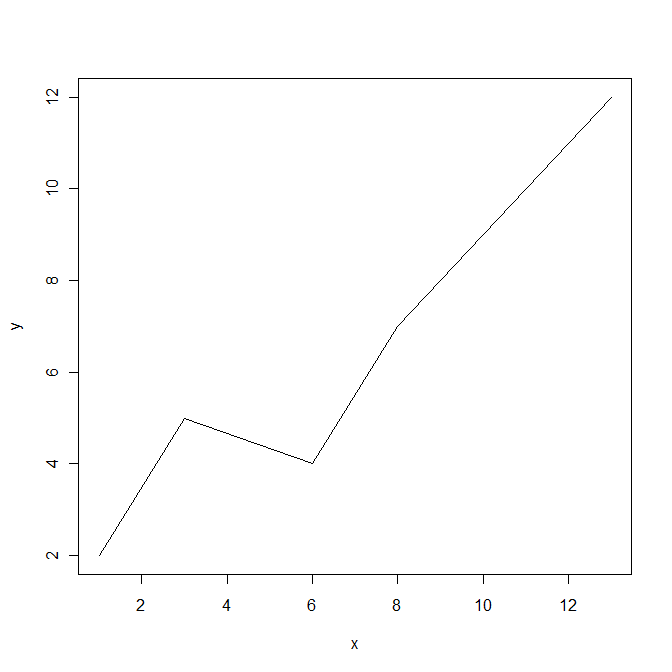
plot(x,y)



To make a line plot instead, add an argument to the plot function of type="l".

#note, the type="l" below is a lowercase L for line, not the number 1.

plot(x,y,type="l")



*plot y vs. x instead of x vs. y.*

*plot y vs. x as a line plot.*

# 5. Data from Files

## Read in some data

yeast <- read.table("H:/introR/data/yeast/gene\_relationships.txt",header=T,sep="\t")

#or mac

yeast <- read.table("/Volumes/HOME/introR/data/yeast/gene\_relationships.txt",header=T,sep="\t")

In this case, data.txt must be a text file with the following properties:

* In each row, fields are separated by a tab, comma, or other single-character
* Each row contains the same number of fields

If your data doesn't look like that, options include

* Edit it manually in Excel / Text editor if it's a text format and needs a minor change
* Edit it with a perl / python / ruby script or a linux command line tool like sed or awk
* Google refine? <http://code.google.com/p/google-refine/>
* Use a specialized R package if it's a special file type (Rsamtools for BAM, affy for CEL, XML for XML)

At some point as you're learning R, you will run into problems reading in a data file. Try adding the following arguments to your *read.table()* function.

* sep="\t" *(separation character is a tab)*
* quote="" *(ignore quotes)*
* comment.char='"" *(ignore comment character "#")*
* strip.white=T *(strip extra white space)*
* try reading in the first few lines with nlines=5 or skipping the first line with skip=1

Other options outside of R

* open in Excel or a text editor, look for weird characters or spacing problems, extra tabs, etc.
* Google the error (in quotes) along with the words cran or R
* If it gives you an error on a specific line, look at that line in the file using excel or a text editor that has line numbering

## Data frames

A Data frame is a rectangular collection of columns. Each column can be a different type of data, numbers, strings, logical (TRUE/FALSE). If you use *read.table()*, the result is already a data frame.

To access parts of a data frame, use the square brackets. [ ] Square brackets are always used to access part of a data structure.

You can access an individual row of a data frame (the first row):

yeast[1,]

an individual column, like column 4:

yeast[,4]

or a specific element, like the element in row 2, col 7:

yeast[2,7]

as well as sections:

#first 10 lines

yeast[1:10,]

#head() is a special function to view the first 6 lines of a data frame.

head(yeast)

Examine the column names or row names, or set them using *colnames()* and *rownames()* functions

colnames(yeast)

#setting the row names to the gene names

rownames(yeast) <- yeast$gene

You can also access individual columns by name, two different, equivalent ways:

yeast$gene

yeast[,"gene"]

Or get a few columns

yeast[,c("gene","chrom")]

If you wanted to access a specific gene's row by gene name, you could do the following:

iv <- yeast[,4]=="YIL162W"

yeast[iv,]

iv is something we call an index vector, which has a logical value of TRUE/FALSE for every element in the column telling whether or not it is equal to "YIL162W". Then we are applying that index vector to every row in the data frame, only pulling out rows for which it is TRUE.

You don't have to explicitly define the index vector, you can directly put it in the brackets like this:

yeast[yeast[,4] == "YIL162W", ]

You could do something similar to find all overlapping genes:

yeast[yeast$left\_gene\_relationship == "overlapping",]

You can sort a data frame using *order()*

yeast.sort <- yeast[order(yeast$chrom),]

You can use some neat functions to summarize over rows or columns of a data frame.

#Find the means of two columns

colMeans(yeast[,c(8,11)])

#apply any function (here median) to columns (MARGIN=2) or rows (MARGIN=1) of a data frame

apply(yeast[,c(8,11)],MARGIN=2,FUN=median)

*Save the 4th column from the data frame into a new variable called "genes".*

*Plot left\_gene\_dist vs right\_gene\_dist*

*Find the genes with a left\_gene\_dist less than 0.*

# 6. Basic Plotting

## More on scatter plots

Read in a data set listing temperature measurements across the US from 1851 until 2009. *read.csv()* is another function to read in a file specifically for comma-separated-values, you could also use *read.table()* here with sep=",".

weather <- read.csv("H:/introR/data/weather/us\_weather.csv",as.is=T,strip.white=T)

#or mac

weather <- read.csv("/Volumes/HOME/introR/data/weather/us\_weather.csv",as.is=T,strip.white=T)

Make a version with all the temperatures in Fahrenheit and store it in a variable called fweather

First, copy the data to the new variable.

fweather <- weather

Then, modify columns 7 through 18 (the temperature data) by converting each number to Fahrenheit from Celsius, (multiply by 9/5 and add 32 to each value).

fweather[,7:18] <- weather[,7:18] \* 9/5 + 32

Select only the weather for Boston, Massachusetts and put it into a variable called boston.

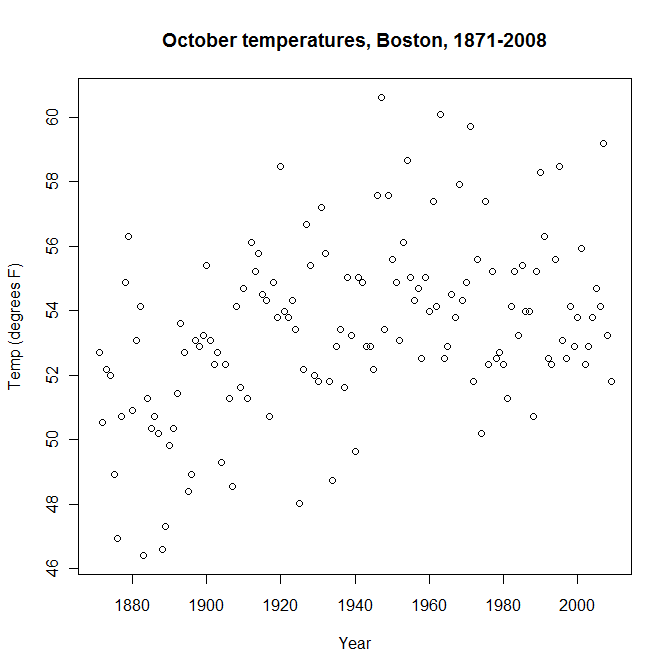
boston.iv <- fweather[,1]=="BOSTON"

boston <- fweather[boston.iv,]

plot(boston[,"Period"],boston[,"Oct"])

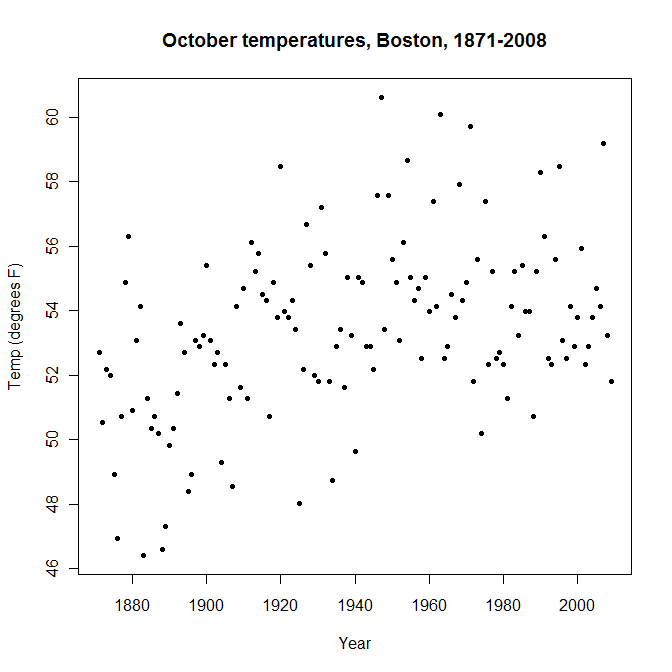
Add a title to the plot with the function argument *main*, and label the axes with *xlab* and *ylab*.

plot(boston[,"Period"],boston[,"Oct"],main="October temperatures, Boston, 1871-2008",xlab="Year",ylab="Temp (degrees F)")

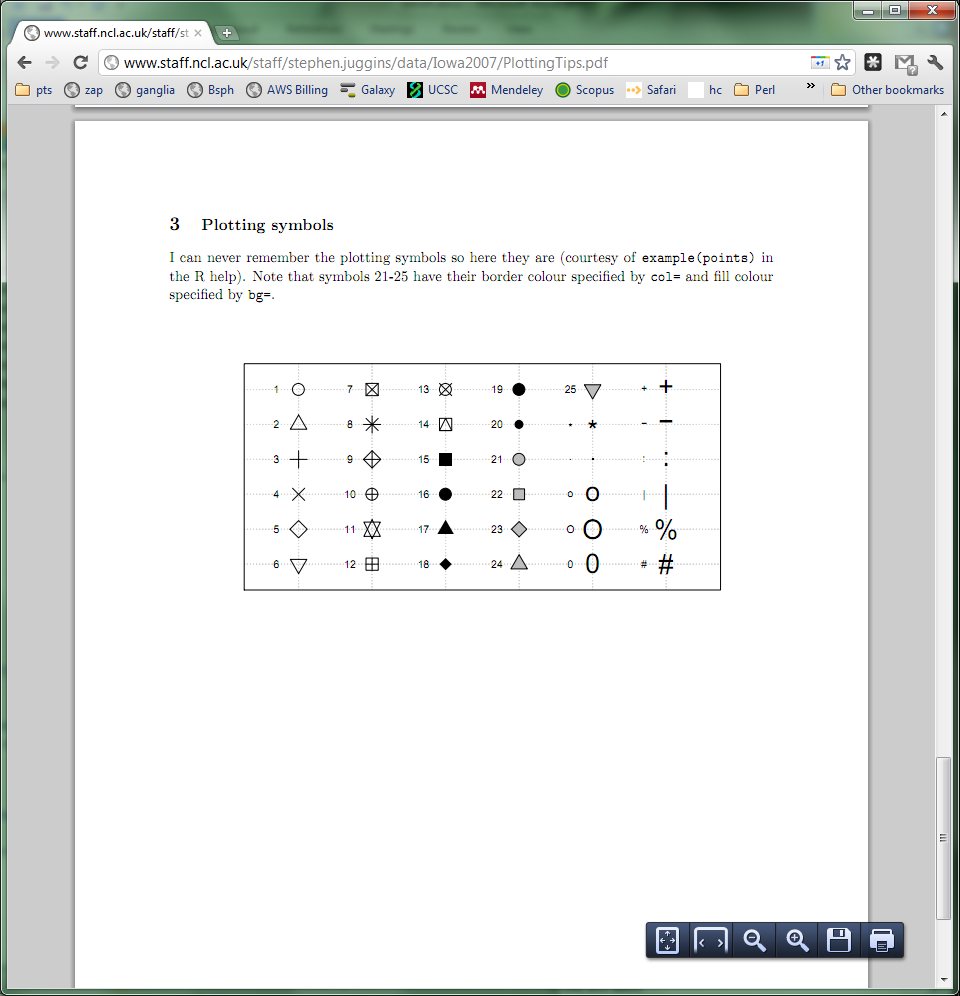


Change the plotting character (the points on the plot) with the *pch* argument

plot(boston[,"Period"],boston[,"Oct"],main="October temperatures, Boston, 1871-2008",xlab="Year",ylab="Temp (degrees F)",pch=20)



Here are some other options for plotting character, or you can use letters or symbols like "." or "A".

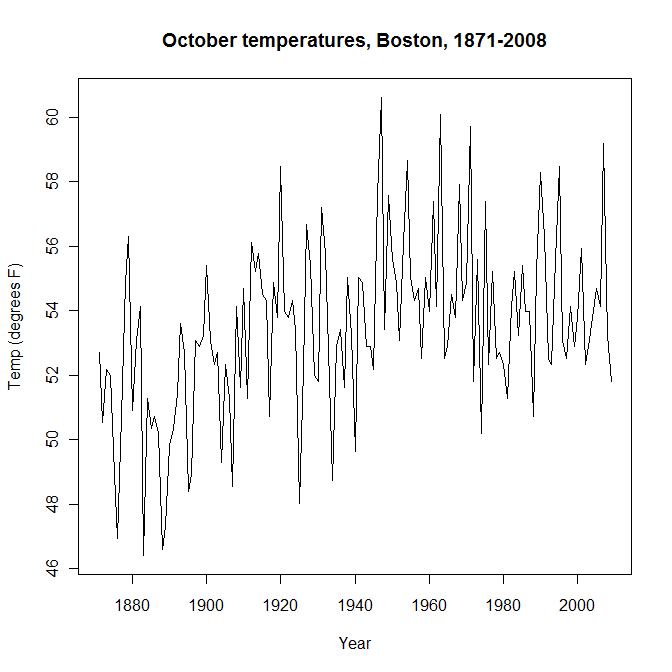


*Plot October temperatures vs. November temperatures from the boston data set.*

## Line plot

A line plot is just a variation of a scatter plot with lines instead of points. To get a line plot, you just include argument *type="l"* to the *plot()* command. L as in line.

plot(boston[,"Period"],boston[,"Oct"],main="October temperatures, Boston, 1871-2008",xlab="Year",ylab="Temp (degrees F)",type="l")



Let's smooth out the line a bit and add a few more lines in different colors. To add lines to an already open plot, use *lines(). lowess()* is used to smooth the lines*.*

plot(lowess(boston[,"Period"],boston[,"Oct"],f=.1),main="Temperatures, Boston, 1871-2008",xlab="Year",ylab="Temp (degrees F)",type="l",ylim=c(10,70))

lines(lowess(boston[,"Period"],boston[,"Nov"],f=.1),col="blue")

lines(lowess(boston[,"Period"],boston[,"Dec"],f=.1),col="red")

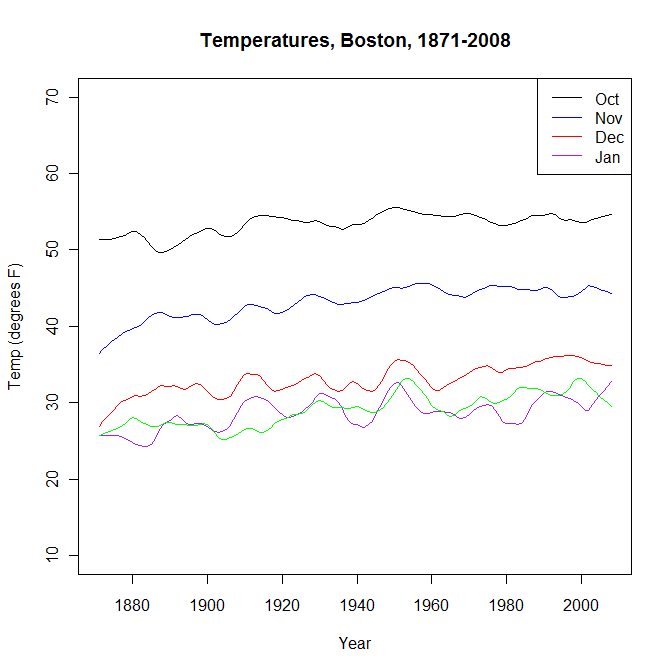
lines(lowess(boston[,"Period"],boston[,"Jan"],f=.1),col="purple")

lines(lowess(boston[,"Period"],boston[,"Feb"],f=.1),col="green")

Legend

You can add a legend to a plot with *legend()*

legend("topright",legend=c("Oct","Nov","Dec","Jan"),col=c("black","blue","red","purple","green"),lty=1)



Histogram

*Remove the smoothing from the above line plots and change all lines into points by removing type="l" from the plot function and using points() instead of lines().*

*Create a legend with points instead of lines (get rid of lty argument and add pch=1).*

Make a histogram of the October mean temperatures in Boston.

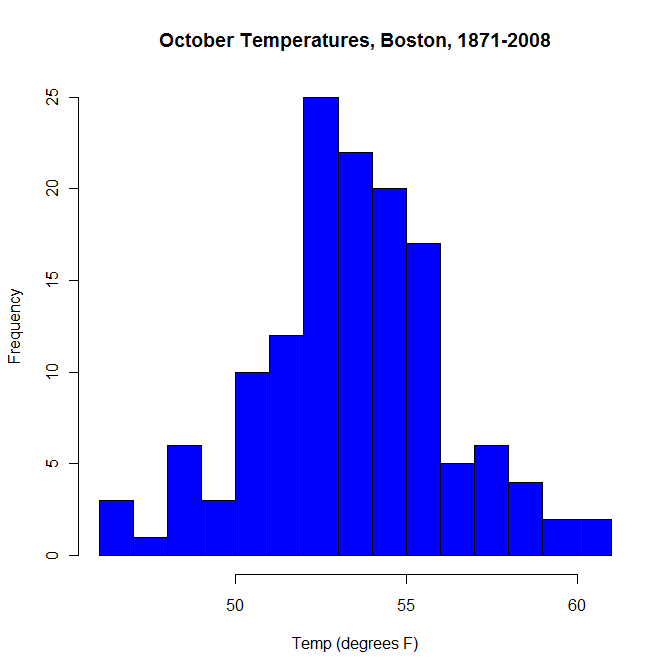
hist(boston[,"Oct"])

Divide the data into more bins in the histogram by adding the argument *breaks*.

hist(boston[,"Oct"],breaks=20)

Make the histogram a different color and give it a better title and labels

hist(boston[,"Oct"],breaks=20,col="blue",main="October Temperatures, Boston, 1871-2008",xlab="Temp (degrees F)",ylab="Frequency")



*Make a histogram of December temperatures.*

## Create a barplot

Read in some hox gene qPCR data (RA induction in mouse ES cells).

hox <- read.table("H:/introR/data/hox/hox\_qpcr.txt",sep="\t",header=T)

#or mac

hox <- read.table("/Volumes/HOME/introR/data/hox/hox\_qpcr.txt",sep="\t",header=T)

Change the column names by substituting X with t in the column names using the *gsub()* function, with arguments for the initial *pattern*, the desired *replacement*, and the text in which to make the substitution.

colnames(hox) <- gsub(pattern="X",replacement="t", x=colnames(hox))

And changing column names to lowercase

colnames(hox) <- tolower(colnames(hox))

Try making a barplot of one gene. First, let's just pull out the data for one gene into a separate data frame called hoxb1. (2:42 is to pull out the numeric data only, because column 1 is the gene name).

hoxb1 <- hox[hox$gene == "Hoxb1",2:42]

If we try to make a barplot with hoxb1, we get an error.

barplot(hoxb1)

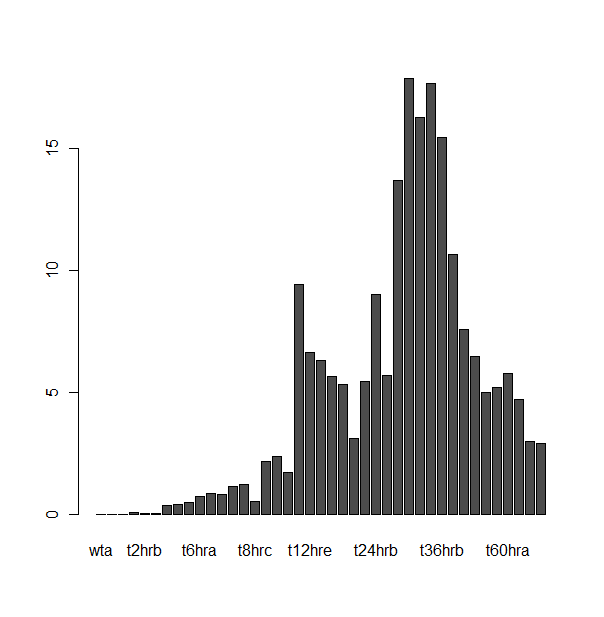
Error in barplot.default(hoxb1) : 'height' must be a vector or a matrix

This is because hoxb1 is a data frame, and *barplot()* requires a vector or a matrix, take a look at ?barplot. A matrix is like a data frame, but with all the columns of the same type - numbers, strings, or logical.

class(hoxb1)

[1] "data.frame"

barplot(as.matrix(hoxb1))



## Improve the basic barplot

We made a basic barplot, but let's fix a few things. *rainbow()* is a quick function to get a list of colors of a certain length. *rep()* is a way to repeat a certain sequence, in this case using the *each* argument to repeat each color 3 times (because we have 3 replicates for each sample and want to make them the same color).

cols <- rainbow(13)

cols <- rep(rainbow(13),each=3)

#if we look at cols, we see the hex values of colors each repeated 3 times.

cols

[1] "#FF0000FF" "#FF0000FF" "#FF0000FF" "#FF7600FF" "#FF7600FF" "#FF7600FF" "#FFEB00FF" "#FFEB00FF" "#FFEB00FF"

[10] "#9DFF00FF" "#9DFF00FF" "#9DFF00FF" "#27FF00FF" "#27FF00FF" "#27FF00FF" "#00FF4EFF" "#00FF4EFF" "#00FF4EFF"

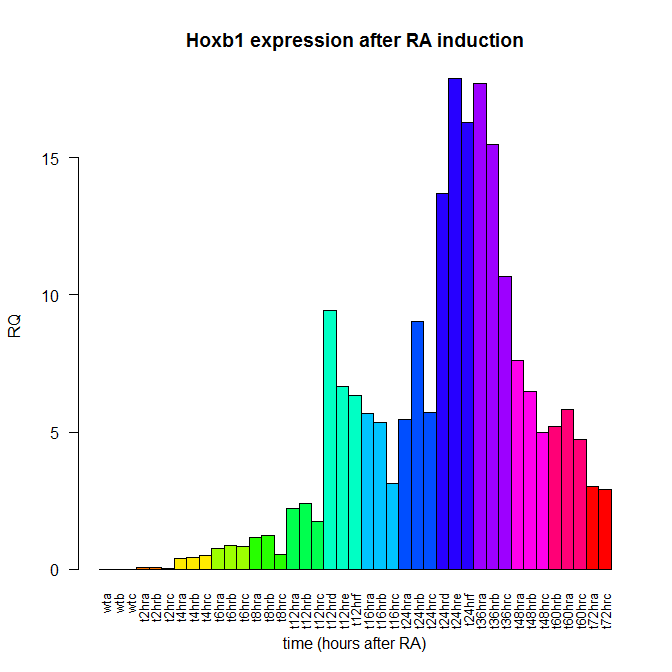
[19] "#00FFC4FF" "#00FFC4FF" "#00FFC4FF" "#00C4FFFF" "#00C4FFFF" "#00C4FFFF" "#004EFFFF" "#004EFFFF" "#004EFFFF"

[28] "#2700FFFF" "#2700FFFF" "#2700FFFF" "#9D00FFFF" "#9D00FFFF" "#9D00FFFF" "#FF00EBFF" "#FF00EBFF" "#FF00EBFF"

[37] "#FF0076FF" "#FF0076FF" "#FF0076FF"

Here is our barplot with colors, making the axis labels perpendicular to the axis with *las*, adding axis labels with *xlab* and *ylab* and a title with *main*, and shrinking the label size with *cex.names*=.8. cex stands for character expansion, and .8 will shrink the characters to 80% of their original size. For more information on plotting parameters, look at ?par.

barplot(t(hoxb1),col=cols,las=2,names.arg=colnames(hox)[2:42],beside=T,ylab="RQ",xlab="time (hours after RA)",main="Hoxb1 expression after RA induction",cex.names=.8)



*Copy the code for hoxb1 to create a similar barplot for hoxb2. First pull out the data, then make the plot.*

# 7. For loops

A for loop is a way of doing the same thing over and over a given number of times.

Remember our data frame of hox genes?

head(hox)

If we wanted to plot the first 10 genes, we could do the following. This will go through the numbers from 1 to 10, and make a barplot each time (for each row from row 1 to row 10).

for(i in 1:10)

{

x11() #to open a new plot each time, otherwise they'll get overwritten

barplot(t(hox[i, 2:42]), beside=T, las=2, names.arg=colnames(hox)[2:42], col=cols, main=hox[i,"gene"], cex.names=.8)

}

One non-plotting example of a for loop: suppose you wanted to square the numbers 3, 9, 4, and 7. *cat()* is a function to output something to the console.

for(i in c(3,9,4,7

{

cat("The square of ",i," is ", i^2,".\n",sep="")

}

Loops can be slow in R and can often be replaced with one of the following functions: apply(), tapply(), lapply(), mapply(). [More info here](http://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/).

cat(unlist(lapply(c(3,9,4,7),function(x) {paste("the square of ",x," is ",x^2,".\n",sep="")})))

*use cat() to print the numbers 1:100 from inside a for loop.*

# 8. Multifigure Plotting

It's easy to put multiple plots on one figure. use *par()* with *mfrow* to set plotting parameters before plotting.

#mfrow stands for multi-figure row, and is expecting number of rows, then number of columns

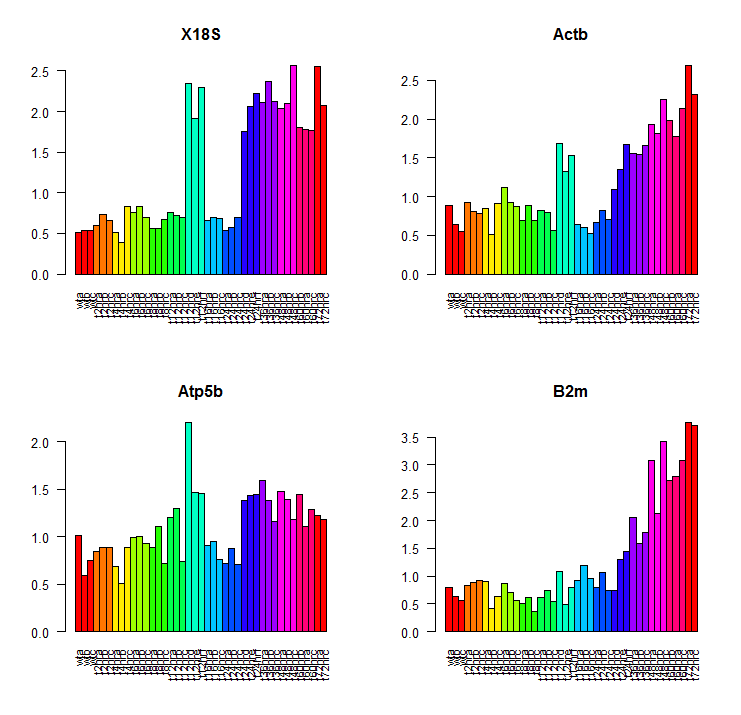
par(mfrow=c(2,2))

for(i in 1:4)

{

barplot(t(hox[i,2:42]), beside=T, names.arg=colnames(hox)[2:42], col=cols, main=hox[i,"gene"],las=2,cex.names=.8)

}



*Change the code above so it produces one row with 4 columns of plots.*

# 9. Getting plots out of R

To save a plot to a PDF from Rstudio, use Export (above the plot window). When you are able, export plots in a vector format, preferably PDF, PS, or Metafile. This will give you the highest resolution later. If you choose bitmap, you will see some blurriness in the image.

|  |  |
| --- | --- |
| Bitmap | Metafile |
|  |  |

Rather than saving your plots one by one after they are generated, you can generate them within your code on the fly with *pdf() and dev.off()*. *pdf()* will open the pdf and start writing plots to it, and any plot you generate will be a new page in the pdf until you call *dev.off().*

pdf("myplots.pdf")

#any plot you make here gets added to the pdf until the dev.off() function is called.

plot(x,y)

for(i in 1:length(hox[,1]))

{

barplot(t(hox[i,2:42]), beside=T, names.arg=colnames(hox)[2:42], col=cols, main=hox[i,"gene"],las=2,cex.names=.8)

}

*Save one of your plots to a pdf.*

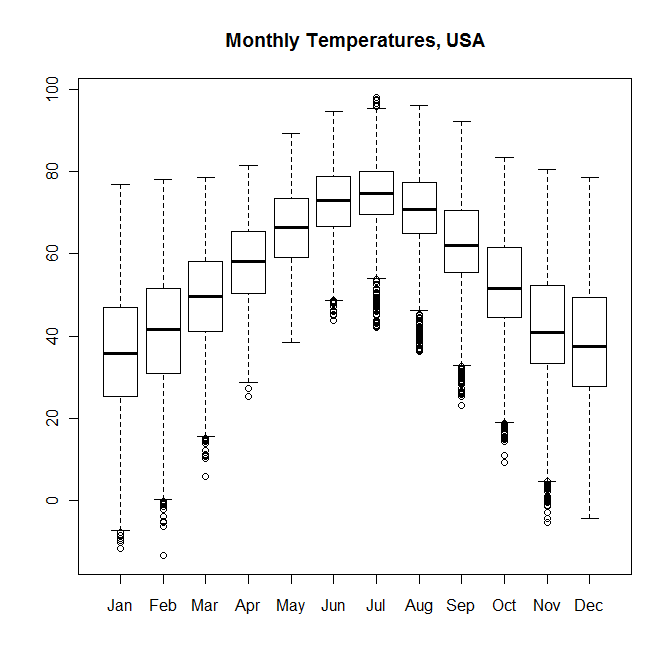
dev.off()

# 10. Boxplots

A boxplot is another way (besides a histogram) to look at a distribution of numbers. Let's go back to our weather data set.

head(fweather)

boxplot(fweather[,7:18], main="Monthly Temperatures, USA")



Here we are seeing the distribution of temperatures for each month throughout the whole data set.

Parts of a boxplot



*Make a boxplot of Boston (boston) temperatures by month.*

median

lower quartile - 25% of the data less than this value

upper quartile - 25% of the data higher than this value

max non-outlier value

outlier (more than 3/2 times the upper quartile)

outlier (less than 3/2 times the lower quartile)

min non-outlier value

# 11. Linear Regression

Linear regression is a way to model the relationship between two (or more) variables using a linear function. Slope and intercept of the function are estimated from the data. (Note - you have to set up the formula like y~x)

What is the relationship between Year and the October temperature in Boston?

lm(boston$Oct~boston$Period)

Call:

lm(formula = boston$Oct ~ boston$Period)

Coefficients:

(Intercept) boston$Period

-1.78033 0.02847

You could also store the result in a variable if you want

model <- lm(boston$Oct~boston$Period)

What is in the variable "model"? What does the function *lm()* return? *class()* will tell you what something is.

class(model)

[1] "lm"

What is in it, or how can you access it? *str()* is a way to examine a mysterious variable in R.

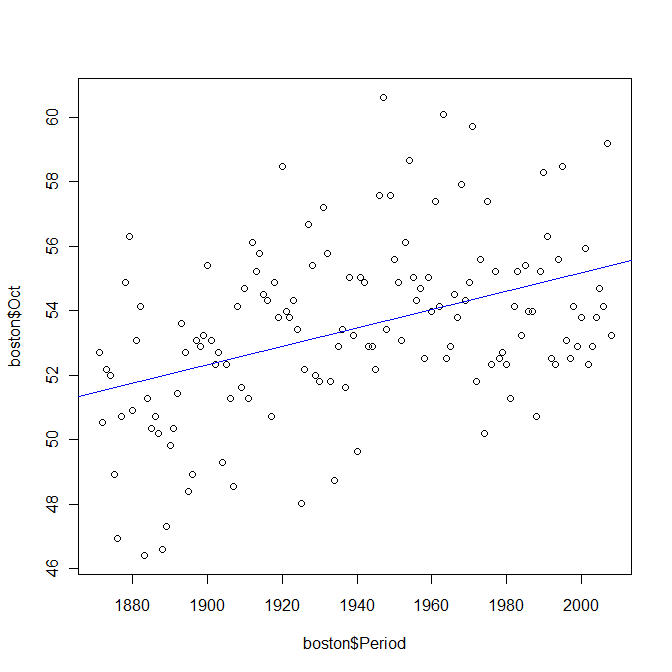
str(model)

Plot the regression line on the scatter plot.

plot(boston$Period,boston$Oct)

abline(lm(boston$Oct~boston$Period),col="blue")

There is an increasing trend in October temperatures in Boston.



*What is the slope of the regression line?*

## Highlighting points on a scatter plot

Highlight some outliers on the plot (based on residual, which is their distance from the regression line).

model <- lm(boston$Oct~boston$Period)

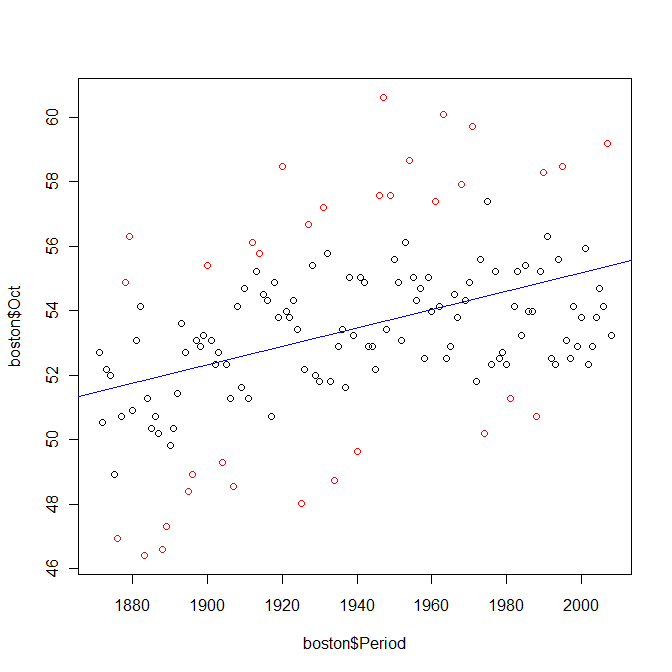
iv <- abs(model$resid) > 3

iv is an index vector, a way of selecting some part of your data in R. If you type *iv*, you see a lot of TRUE and FALSE values. iv is the same length as weather, with a TRUE or FALSE for each element depending on whether or not its residual > 3. Now we will highlight on the plot the points which have a residual > 3.

plot(boston$Period,boston$Oct)

abline(lm(boston$Oct~boston$Period),col="blue")

points(boston$Period[iv],boston$Oct[iv], col="red")



Now that we have our index vector, we can use it to examine our data frame as well. This will show us the data for years with the hottest and coldest Octobers (relative to their time point).

*Read the following file into a variable called "microarray":*

*H:/introR/data/array/microarray1.txt*

*Is there a header on the file? Don't forget header=T*

*Once you have read in the file, try to make a scatter plot of Cy3 vs Cy5.*

*Create an index vector selecting points that have log2(Cy5/Cy3) > 1.*

*Highlight those points on the plot in blue.*

boston[iv,]

# 12. Writing data out

Let's say we want to save some data out to a file. We can use *write.table()*. We give it the data we want in the file, the file name, the separating character with *sep* (\t means tab), and tell it we don't want everything in quotes, and we don't want it to output the row names. This file will appear in your working directory. To check what your working directory is, do *getwd()*

write.table(boston[iv,],file="outliers.txt",sep="\t",quote=F,row.names=F)

# 13. T-test

Many statistical tests are available in R. A simple statistical test is a t-test, which you can use to test whether the means of two samples are equal. Is there a significant difference between October and November in our Boston weather data?

t.test(boston$Oct,boston$Nov)

Just the p-value:

tt <- t.test(boston$Oct,boston$Nov)

tt$p.value

[1] 1.098882e-82

There is a statistically significant difference between October and November temperatures with a p-value of

*Any significant difference between January and February?*

1.099e-82.

# 14. Packages

There is a lot of functionality built into R by itself (base), but there are also many interesting packages that have been created for R that you can use. In order to use these package, you will have to download and install them, and then call library(package) to load them into your current session.

There are two places to get packages - CRAN and bioconductor. CRAN's package repository (comprehensive R archive network) has packages for all kinds of things, while bioconductor is focused on biological data. To browse around and see what packages are available at each, look at <http://cran.r-project.org/> or <http://www.bioconductor.org/>

To install a package once you know the name (case sensitive), do the following: For a cran package:

install.packages("RColorBrewer")

In Rstudio, you can use the menu - go to Tools, install packages.

For Bioconductor packages,

source("http://bioconductor.org/biocLite.R")

biocLite("limma")

Once a package is installed (need to do this ONCE only), load it into your current session with

library("limma")

You have to load it into your session every time you use R. You can see which packages you currently have available with

library()

And see what's loaded into your current session with

sessionInfo()

You can get more info about a package with

help(package="limma")

*What is the ChIPpeakAnno package for?*

# 15. Colors

You can do a lot in R with colors. [ColorChart.pdf](http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf) is a handy resource for colors in R. (Thanks, Earl Glynn).

Colors can be referred to three different ways: text name "yellow", hexadecimal #FFFF00, or RGB with 3 numbers from 0 to 255.

Let's use a palette in a plot. First, back to the hoxb1 data -- let's average the replicates using the aggregate() function.

hoxb1\_avg <- aggregate(t(hoxb1),by=list(rep(1:14,each=3)[1:41]),mean)[,2]

And add some sensible names.

names(hoxb1\_avg)<-c("wt","t2","t4","t6","t8","t12\_1","t12\_2","t16","t24\_1","t24\_2","t36","t48","t60","t72")

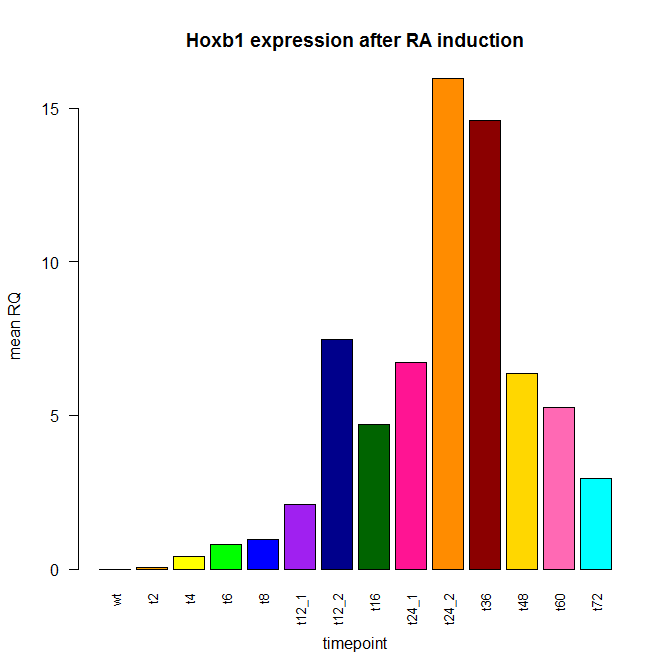
How many colors do we need for our plot?

length(hoxb1\_avg)

We could make our own palette from color names.

mypal<-c("red","orange","yellow","green","blue","purple","darkblue","darkgreen","deeppink","darkorange","darkred","gold","hotpink","cyan")

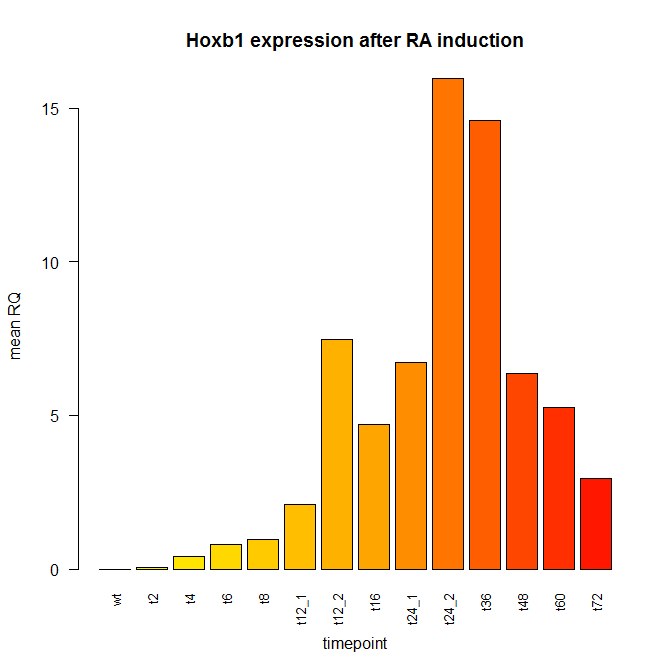
barplot(hoxb1\_avg,col=mypal,las=2,names.arg=names(hoxb1\_avg),ylab="mean RQ",xlab="timepoint",main="Hoxb1 expression after RA induction",cex.names=.8)



To get a palette that ranges from one color to another, we can use *colorRampPalette().* This function is a little unusual in that it returns a function. So the syntax has to be *colorRampPalette()(numcolors)*.

cols <- colorRampPalette(c("yellow","orange","red"))(14)

barplot(hoxb1\_avg,col=cols,las=2,names.arg=names(hoxb1\_avg),ylab="mean RQ",xlab="timepoint",main="Hoxb1 expression after RA induction",cex.names=.8)



A useful package for getting sets of colors (palettes) for plotting is RColorBrewer. It should already be installed (on the training room computers). If it wasn't, to install it, you would do:

install.packages("RColorBrewer")

Load the package into your current session:

library(RColorBrewer)

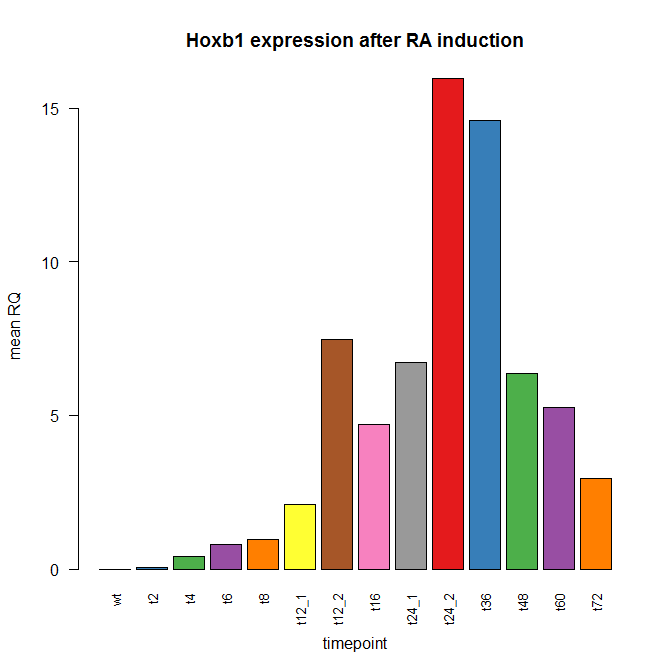
Show all the available palettes in this package.

display.brewer.all()

Let's use a categorical color palette from RColorBrewer:

cols <- brewer.pal(9,name="Set1")

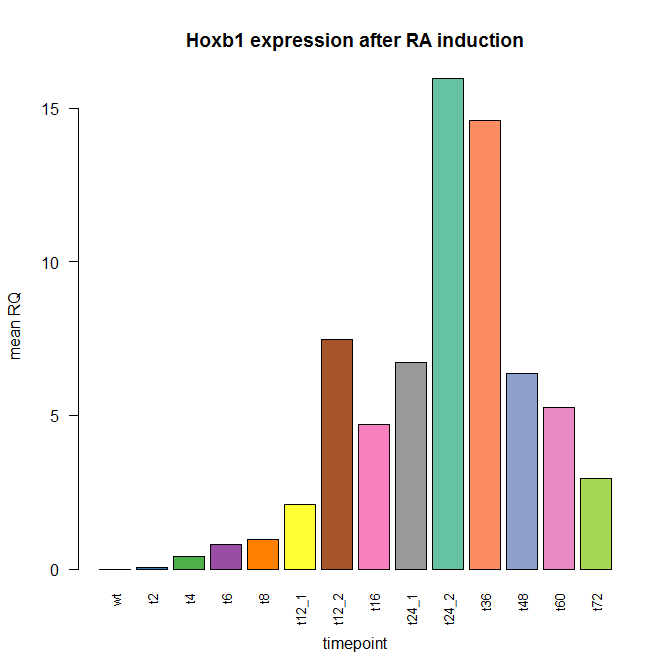
barplot(hoxb1\_avg,col=cols, las=2, names.arg=names(hoxb1\_avg), ylab="mean RQ", xlab="timepoint", main="Hoxb1 expression after RA induction", cex.names=.8)



Notice the colors are recycled if there are more variables than colors (green, purple, orange are shown twice). To fix this, we could combine two RColorBrewer palettes:

cols <- c(brewer.pal(9,"Set1"),brewer.pal(8,"Set2"))

barplot(hoxb1\_avg, col=cols,las=2, names.arg=names(hoxb1\_avg), ylab="mean RQ", xlab="timepoint", main="Hoxb1 expression after RA induction", cex.names=.8)



*Create your own set of colors using c() and color names -- type colors() if you need ideas for colors. Recreate the barplot with your colors.*

# 16. Heat maps

Sometimes it is useful to visualize complex data using heat maps (false color image) to pick out patterns. There are many possibilities to create clustered (or unclustered) heat maps in R using *heatmap()* and *image().* Let's read in some mouse gene expression atlas data (affy arrays on a number of mouse tissues).

exp <- read.csv("H:/introR/data/mouse\_ge/geneatlas\_MOE430.csv",as.is=T)

names <- read.table("H:/introR/data/mouse\_ge/moe4302.txt",as.is=T,header=T,sep="\t")

#or mac

exp <- read.csv("/Volumes/HOME/introR/data/mouse\_ge/geneatlas\_MOE430.csv",as.is=T)

names <- read.table("/Volumes/HOME/introR/data/mouse\_ge/moe4302.txt",as.is=T,header=T,sep="\t")

Let's select some specific columns from the data frame for plotting.

selected <- exp[,c("MEF", "adipose\_brown","adipose\_white", "amygdala", "bladder", "bone",

"bone\_marrow", "cerebellum", "cerebral\_cortex", "cornea", "heart", "iris", "kidney", "lens", "liver", "lung", "olfactory\_bulb", "ovary", "pancreas", "pituitary", "placenta", "prostate", "retina", "salivary\_gland", "skeletal\_muscle", "spinal\_cord", "spleen", "stomach", "testis", "umbilical\_cord", "uterus")]

#setting the rownames to be probeset\_id (gene\_name)

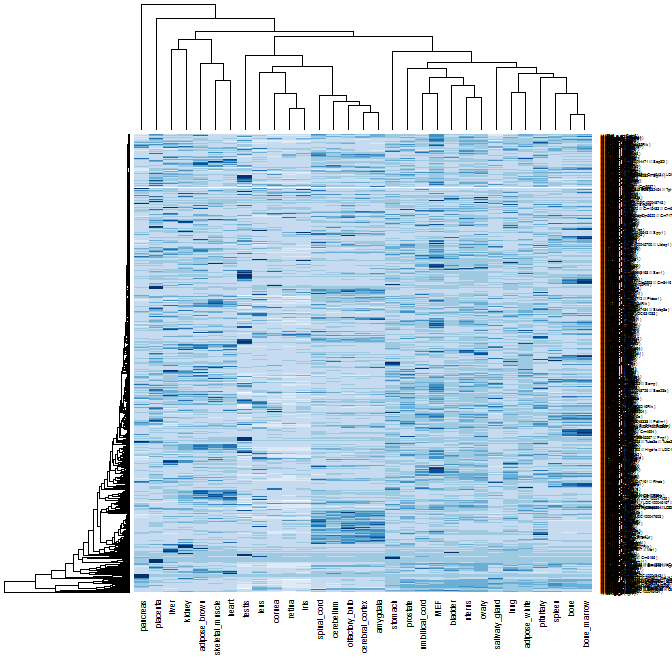
rownames(selected)<-paste(exp[,1],"(",names[match(names[,1],exp[,1]),2],")")

#set the colors

cols <- brewer.pal(9,"Blues")

#create the heatmap

heatmap(as.matrix(selected[1:1000,]),col=cols,cexCol=.6,cexRow=.3)

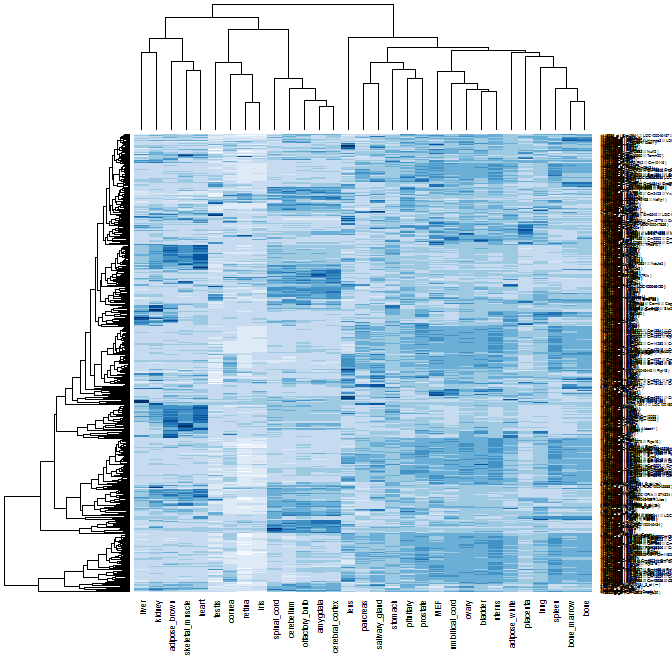


The default *heatmap()* command in R does a hierarchical clustering of the data and rearranges both rows and columns based on their distance from eachother. For more information, see ?heatmap, ?hclust and ?dist.

The size of the heatmap you can create is limited by your computer. Let's just look at the top 500 most highly expressed genes.

highest500.iv <- order(-rowMeans(selected))[1:500]

heatmap(as.matrix(selected[highest500.iv,]),col=cols,cexCol=.6,cexRow=.3)



More flexible heat maps are possible using the *image()* function. If you want to change the heat map a lot, you may have to use some combination of *heatmap()*, saving the resulting orders of rows and columns, and then use *image()*. To add a color scale to the heatmap, you will have to use *image()* and *layout()*.

*Try making a heatmap with the Boston weather data (boston[,7:18]). Don't forget as.matrix().*

# For your consideration

Stuff we didn't cover in class, but you might find useful.

## ****Missing data****

If data is missing in R, it will be assigned a special value called NA. Some functions have arguments like *na.rm=T* to remove NA. You may find yourself needing to use *is.na(data)* to find which values are missing.

boston[is.na(boston)]

unlist(boston)[is.na(boston)]

## Lists

A list is another type of data structure in R. Every element of a list can be of a different type, so it is a high-level data structure. To create a list, use *list()*, and to access an element of a list, use [[1]]. To get a list into a vector, use *unlist()*. To get a simple list to a data frame, use *as.data.frame(list)*.

mylist <- list(c(1,2,3), "A thing", matrix(4,nrow=4,ncol=4), data.frame=yeast)

mylist[[1]]

[1] 1 2 3

mylist[[2]]

[1] "A thing"

mylist[[3]]

[,1] [,2] [,3] [,4]

[1,] 4 4 4 4

[2,] 4 4 4 4

[3,] 4 4 4 4

[4,] 4 4 4 4

mylist[[4]][1:10,]

## Factors

Factors are a feature of R used for categorical data, but might trip you up at first. When reading in a file with *read.table()*, string data will automatically be converted to a factor unless you specify *as.is=T*.

yeast <- read.table("H:/introR/data/yeast/gene\_relationships.txt",header=T,sep="\t",as.is=T)

If you have a factor that you would like to treat as character strings instead, use *as.character()*.

# Resources

R search engine - <http://rseek.org>

Stack Overflow - <http://stackoverflow.com>

R graph gallery - <http://addictedtor.free.fr/graphiques/>

R reference card - <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

R Color Chart - <http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf>

# Homework?

Take the data from H:/introR/data/genomes/genomesizes.txt. Read in the data into a data frame.

Which organism has the minimum genome size in this list? maximum? median?

Make a barplot of the genome sizes. Make the labels perpendicular to the axis.

Bonus points: Order the bars by size. Adjust margins so you can see the axis. Add colors of your choosing.

# Acknowledgements and Data

Thanks to Gaye Hattem, Chris Seidel, and Jim Vallandingham for helping with the class and/or providing some examples. Some other examples were adapted from "25 recipes for getting started with R", by Paul Teetor.

Data sets include:

Yeast gene data derived from UCSC table browser for sacCer2

Mouse ES cell Hox qPCR data from Mark Parrish and Bony De Kumar, Krumlauf lab

US Weather station data from the guardian <http://www.guardian.co.uk/environment/datablog/2009/dec/08/uk-us-temperature-change-global-met>

Mouse Gene Atlas data from <http://biogps.org/downloads/>

Genome sizes from UCSC genome browser