Introduction to R

Madelaine Gogol

Computational Biology

October 2011

[Introduction 2](#_Toc308008687)

[1. Getting Started 3](#_Toc308008689)

[2. R as a calculator 5](#_Toc308008700)

[3. Vectors 6](#_Toc308008702)

[4. Data from Files 8](#_Toc308008705)

[5. Basic Plotting 11](#_Toc308008710)

[6. For loops 17](#_Toc308008717)

[7. Multifigure Plotting 17](#_Toc308008718)

[8. Getting plots out of R 18](#_Toc308008719)

[9. Boxplots 19](#_Toc308008720)

[10. Linear Regression 22](#_Toc308008722)

[11. Writing data out 24](#_Toc308008724)

[12. Missing data 24](#_Toc308008725)

[13. Packages 24](#_Toc308008726)

[14. Colors 26](#_Toc308008727)

[15. Heat maps 28](#_Toc308008728)

[Homework 30](#_Toc308008729)

[Acknowledgements and Data 30](#_Toc308008730)

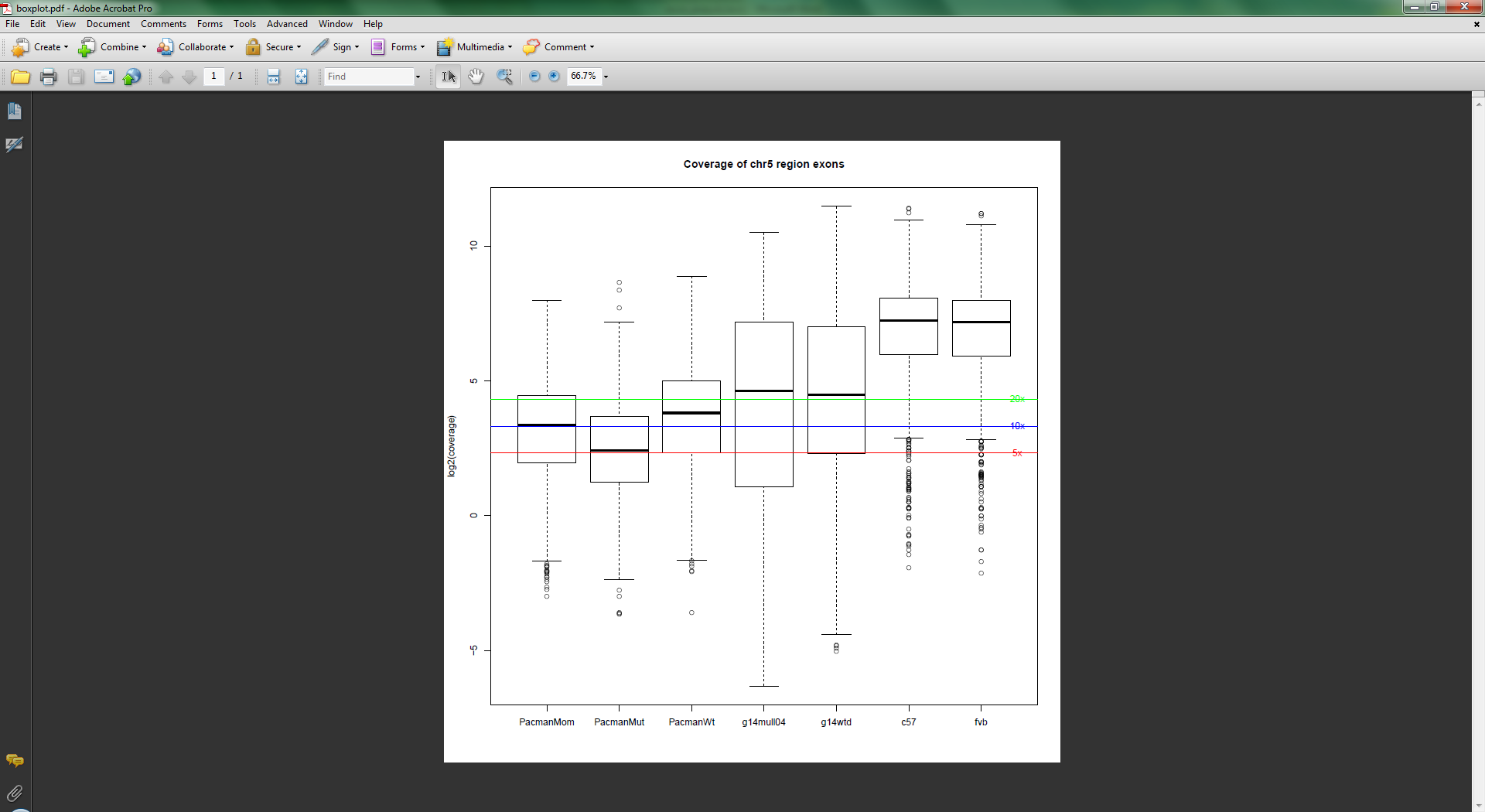
# Introduction

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)

# S:\Bioinformatics\analysis\Krumlauf\bone_de_kumar\bdk19\mcm_analysis\maplots.pnghttp://had.co.nz/ggplot2/graphics/55078149a733dd1a0b42a57faf847036.pnghttp://2.bp.blogspot.com/-38wWn7KD6v0/TgTJskG1ujI/AAAAAAAAC5g/0k33b0L5fL8/s1600/heatmap.pnghttp://genomebiology.com/content/figures/gb-2006-7-7-r66-4.jpg



# 1. Getting Started

## Download and install R

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

Select a mirror site near you

Windows

Download and Install R

base

(click exe of latest version)

OSX

MacOS X

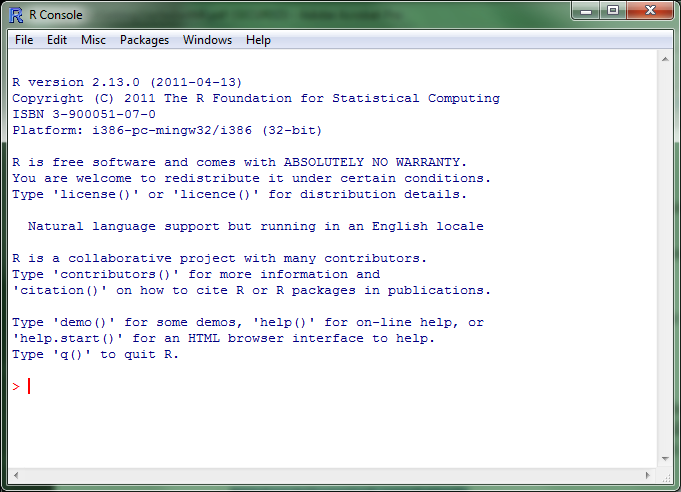
(click .pkg of latest version)

Optional: Download, install, and run Rstudio

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

## Start R

This is the R console

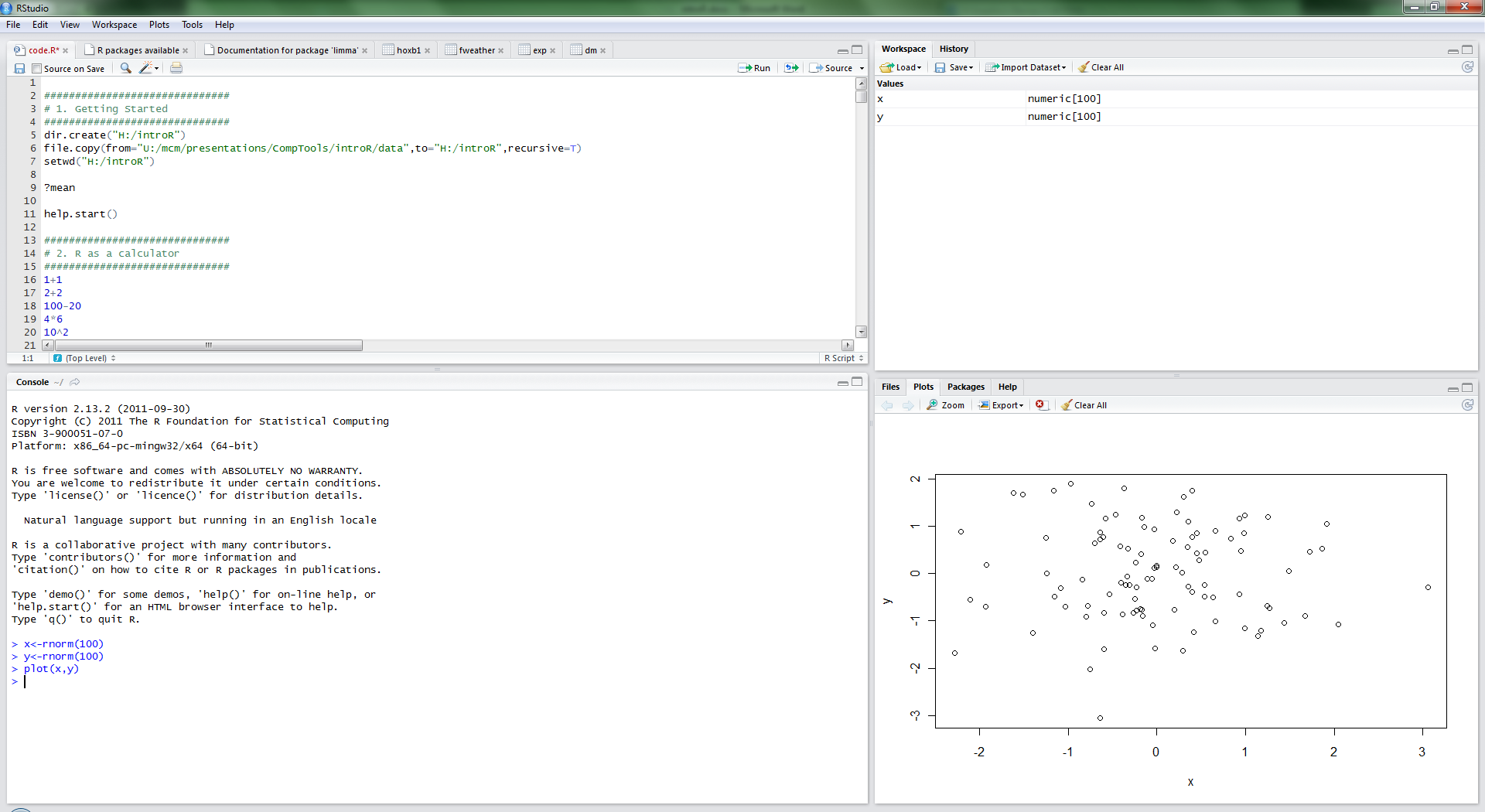


Type or paste code here.

## 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).

## If you use Rstudio



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

Data shows up here

Plots or help shows up here

This is your text file with code. Ctrl-Enter to send to the console

## 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, 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/code.R")

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 (though not strictly required) 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()

## 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>

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 groundwork 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 touch on 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(2), 2 is the argument. Or when you say setwd("H:/introR"), "H:/introR" is the argument.

# 2. R as a calculator

## Entering commands

You can use R like a simple calculator:

1+1

[1] 2

Try some of these:

2+2

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. Vectors

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

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

[1] 1 3 6 8 13

If you would like to save that vector so you can use it for something, you can use the assignment operator "*<-*" to assign it to the variable x.

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

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 see a # in front of something, it's just a comment and is ignored by R.

x

[1] 1 3 6 8 13

#create another vector, y.

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

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

y

[1] 2 5 4 7 12

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

x[4]

[1] 8

Create some more vectors:

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

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

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

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

If you mix strings and numbers, the numbers will be treated as strings.

v1 <- c(6, 5,"hi")

[1] "6" "5" "hi"

A vector of two vectors gets combined into one vector.

z <- c(x,y)

z

[1] 1 3 6 8 13 2 5 4 7 12

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.

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)

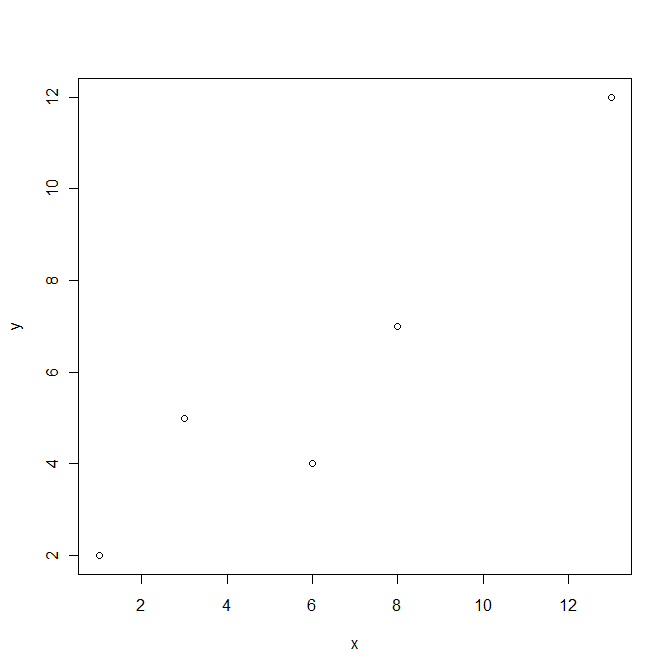
*what is the median of numbers 4 to 7?*

*what is the correlation of the square root of numbers 1-100 with the log2 of the numbers 1-100?*

*which elements of z are greater than 7?*

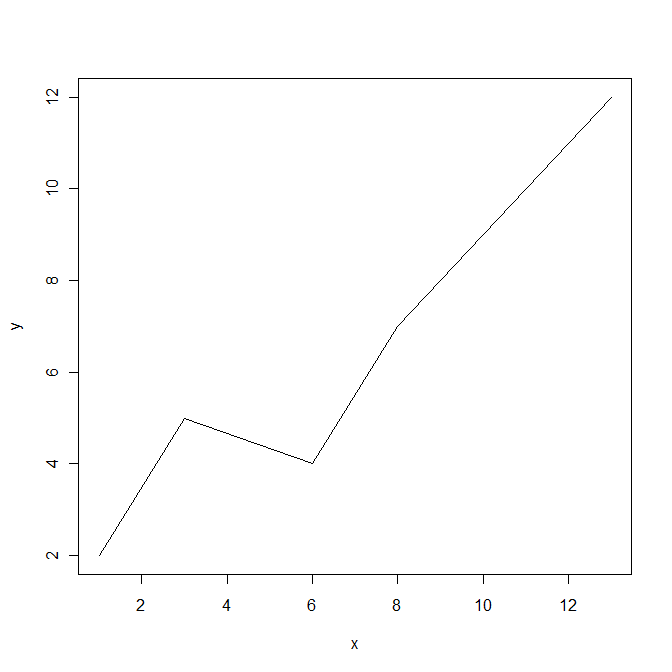
## Plot x vs y

plot(x,y)



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

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



More on this later!

*Try plotting just x by itself. What happens?*

*plot the log2 of numbers 1 to 100 by the square root of numbers 1 to 100.*

# 4. Data from Files

## Read in some data

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

#or mac

df <- 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"
* quote=""
* comment.char='""
* strip.white=T
* 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) with R or cran
* If it breaks on a specific line, look at that line in the file

## 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, which you can check using

class(df)

[1] "data.frame"

Each column of the data frame is a vector, and each element of the data frame will be whatever type it is. You can access an individual row:

df[1,]

column:

df[,4]

or element:

df[2,7]

as well as little sections:

df[1:10,]

head(df)

df[1:3,1:4]

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

df$gene

df[,"gene"]

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

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

df[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:

df[df[,4] == "YIL162W", ]

You could do something similar to find all overlapping genes:

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

Or all genes without a close neighbor

df[df$left\_gene\_dist > 5000 & df$right\_gene\_dist > 5000,]

You can sort a data frame by one column using *order()*

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

#note, this will sort alphabetically chr1, chr10, chr11.

#sort by chromosome, then start.

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

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

#Find the means of a bunch of columns

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

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

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

*Save the first 10 rows from the column into a new variable called "top".*

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

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

*Which gene is the furthest away from another gene on the left and how far is it? (Hint - use which.max)*

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

## 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=df)

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,]

Lists are useful for complex data types, but I'm not going to cover them in this beginner class.

## 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.

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

# 5. 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

fweather <- weather

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

Select only the weather for Des Moines, Iowa for a moment.

dm.iv <- fweather[,1]=="DES MOINES"

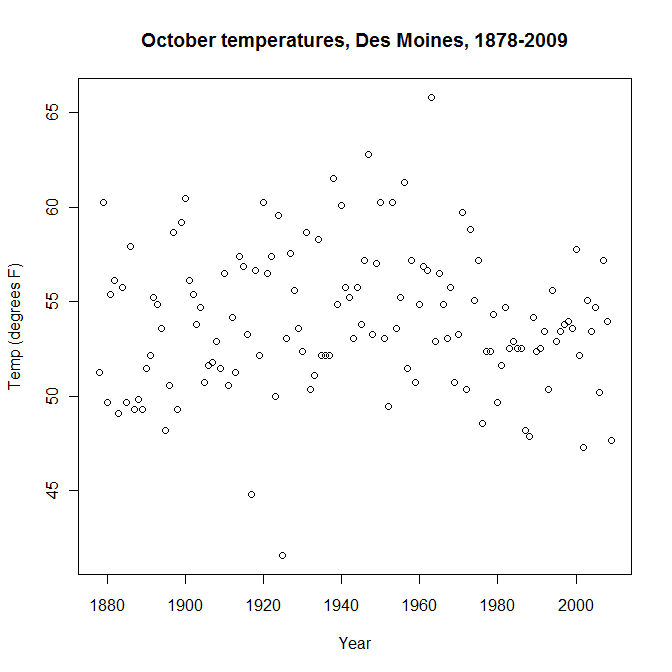
dm <- fweather[dm.iv,]

dm[,"Oct"]

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

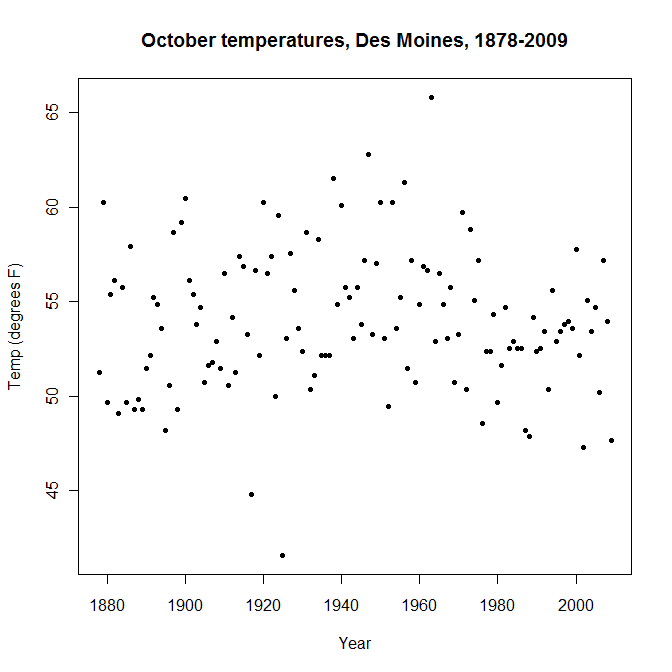
Add a title, label axes.

plot(dm[,"Period"],dm[,"Oct"],main="October temperatures, Des Moines, 1878-2009",xlab="Year",ylab="Temp (degrees F)")

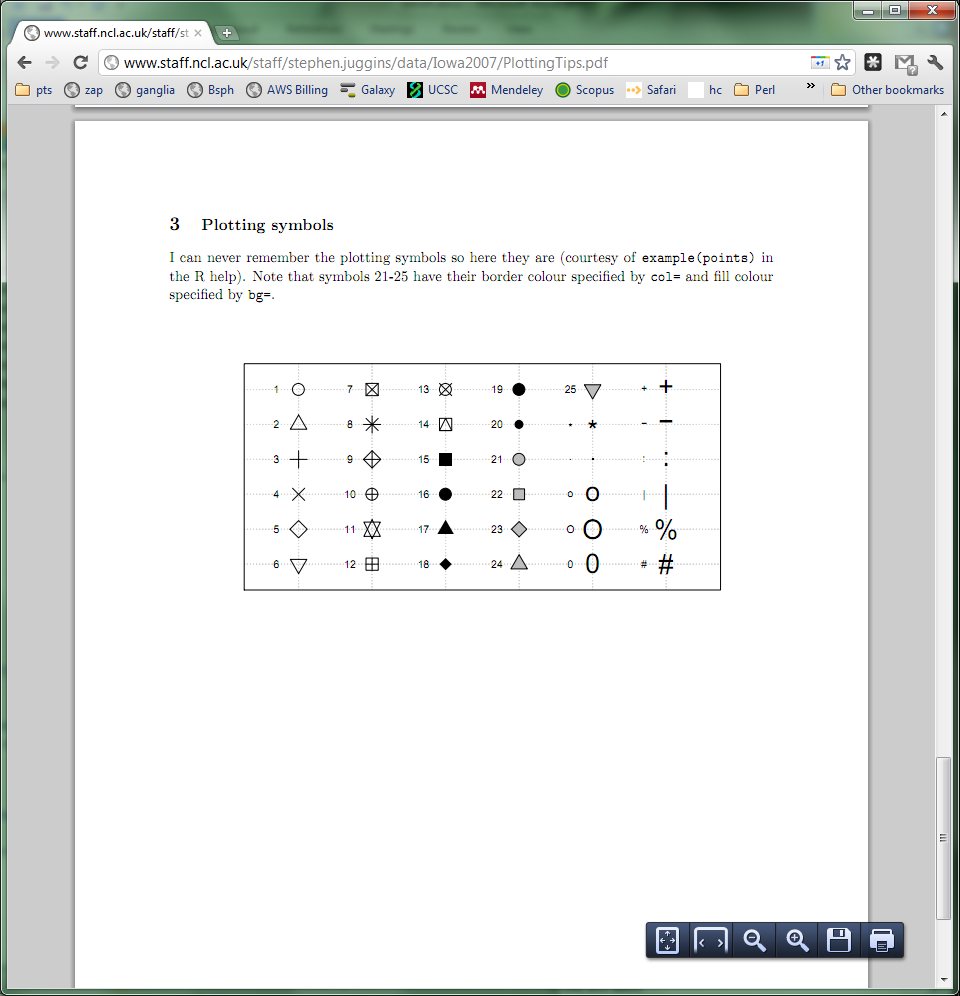


Change the plotting character with the pch argument

plot(dm[,"Period"],dm[,"Oct"],main="October temperatures, Des Moines, 1878-2009",xlab="Year",ylab="Temp (degrees F)",pch=20)



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

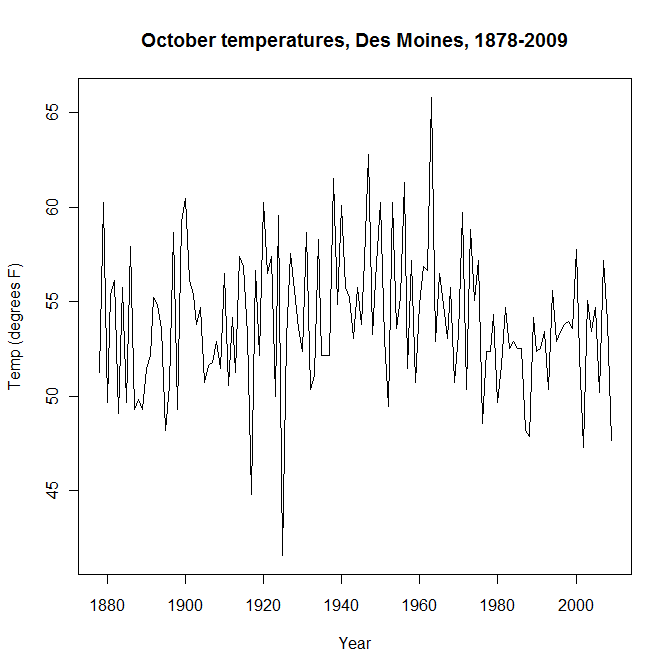


*Plot October temperatures vs. November temperatures from the dm data set. Experiment with different plotting characters.*

## 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(dm[,"Period"],dm[,"Oct"],main="October temperatures, Des Moines, 1878-2009",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*.*

iv <- dm$Period > 1900 & dm$Period < 2000

plot(lowess(dm[,"Period"][iv],dm[,"Oct"][iv],f=.1),main="Temperatures, Des Moines, 1900-2000",xlab="Year",ylab="Temp (degrees F)",type="l",ylim=c(10,70))

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

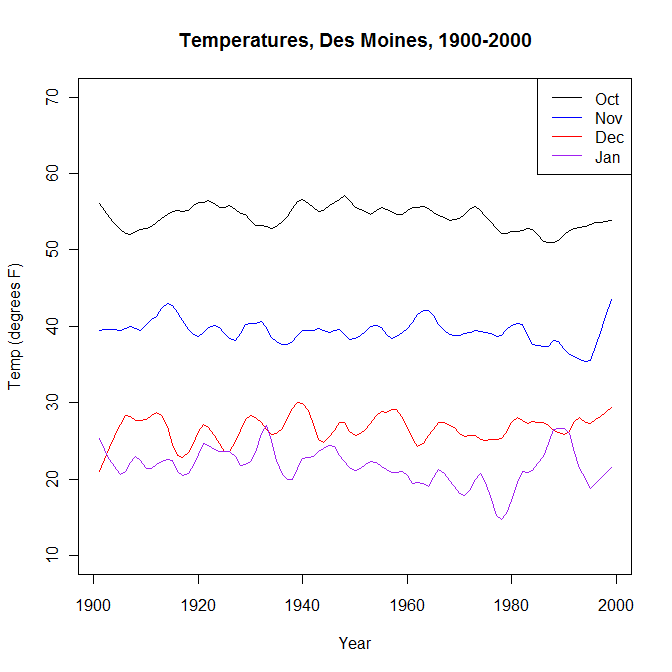
lines(lowess(dm[,"Period"][iv],dm[,"Dec"][iv],f=.1),col="red")

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

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"),lty=1)



*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 (hint:get rid of lty argument and add pch=1).*

*Try changing the plotting character of each month to O, N, D, J, and change the legend.*

Histogram

Make a histogram of the October mean temperatures in Des Moines.

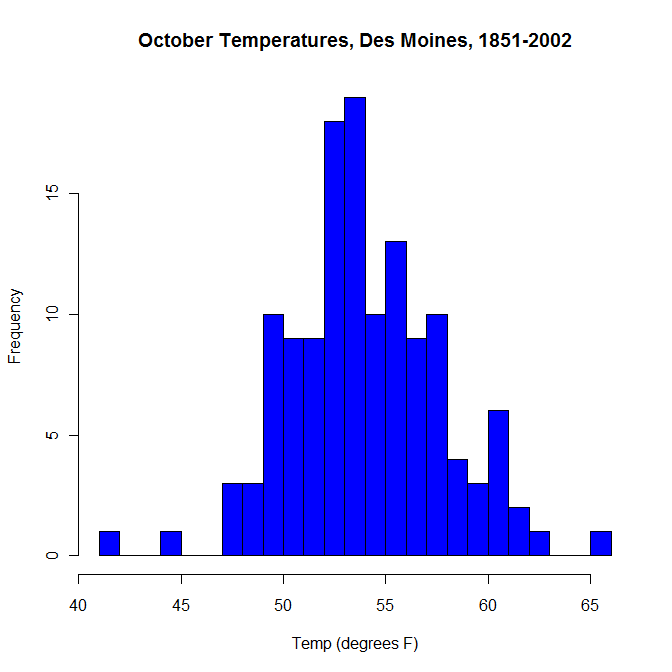
hist(dm[,"Oct"])

Make the histogram have more bins

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

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

hist(dm[,"Oct"],breaks=20,col="blue",main="October Temperatures, Des Moines, 1878-2009",xlab="Temp (degrees F)",ylab="Frequency")



*Make a histogram of December temperatures. Experiment with changing the value of the breaks argument to see what effect it has.*

## 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 column names and changing them to lowercase

colnames(hox) <- gsub("X","t",colnames(hox))

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]

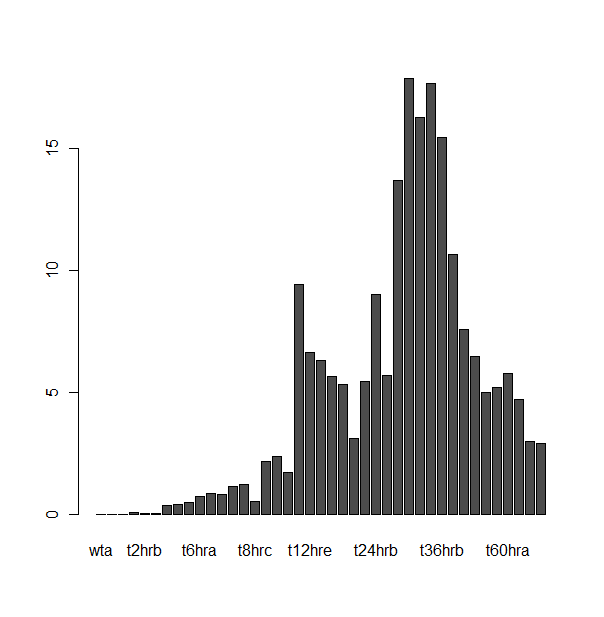
To make a barplot, R is built to expect a matrix or a vector. A matrix is similar to a data.frame, but with all the columns the same type - numbers, strings, or logical.

From the R barplot() documentation:

If height is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If height is a matrix and beside is FALSE then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked sub-bars making up the bar. If height is a matrix and beside is TRUE, then the values in each column are juxtaposed rather than stacked.

If your barplot doesn't work as you expect, try things like as.matrix(), t() to transpose it, or beside=T.

barplot(as.matrix(hoxb1))



## Improve the basic barplot

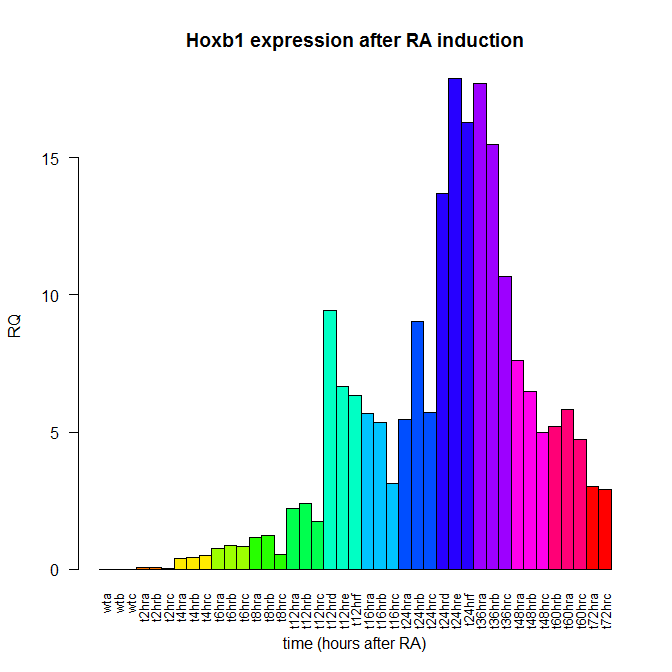
We made a basic barplot, but let's fix a few things. Let's make prettier colors, for one thing. *rainbow()* is a quick function to get a list of colors of a certain length. *rep()* is a way to repeat a certain sequence, in this case using the "each" argument to repeat each color 3 times (because we have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

cols <- rainbow(13)

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

Barplot with colors, make x labels horizontal, add axis labels and a title, shrink the label size.

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.*

# 6. 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

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)

}

Soon I'll show you a few ways to sensibly capture all those plots.

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="")

}

Of course, the more efficient way would be to do

c(3,9,4,7)^2

Loops can be slow in R and can often be replaced with one of the following functions: apply, tapply, lapply, mapply, etc. But this is for a more advanced class. [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.*

# 7. Multifigure Plotting

It's easy to put multiple plots on one figure

#mfrow is expecting # of rows, # of columns

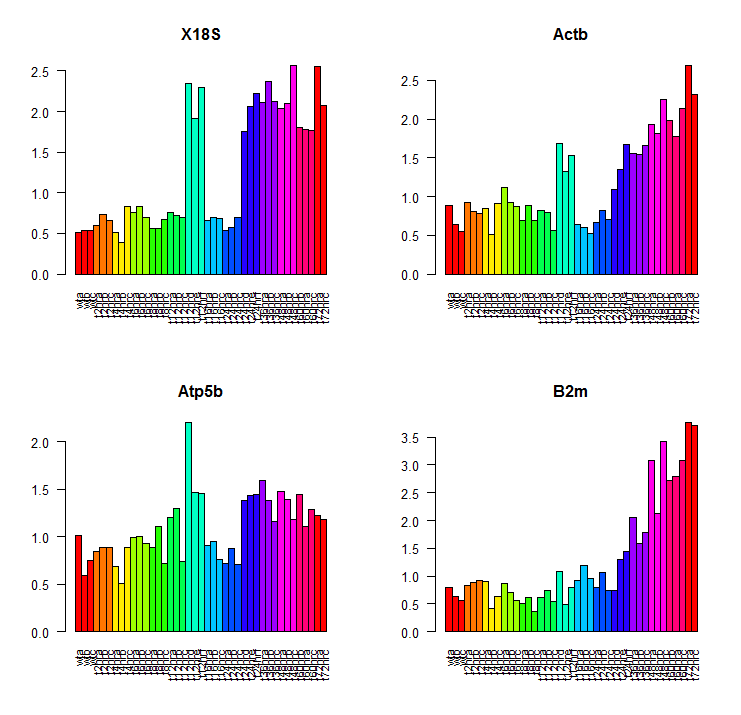
par(mfrow=c(2,2))

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.*

# 8. Getting plots out of R

To save a plot to a PDF, you could use the menu - File - Save As. (In Rstudio, Export)

A few words on images - When you are able, export graphs 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. If your plot has a large number of points, you may not want a vector format for initial presentation, as it will take awhile to render all the points.

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

Rather than saving your plots one by one after they are generated, you can generate them within your code on the fly.

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)

}

dev.off()

*Generate any plot from above and save it to a pdf.*

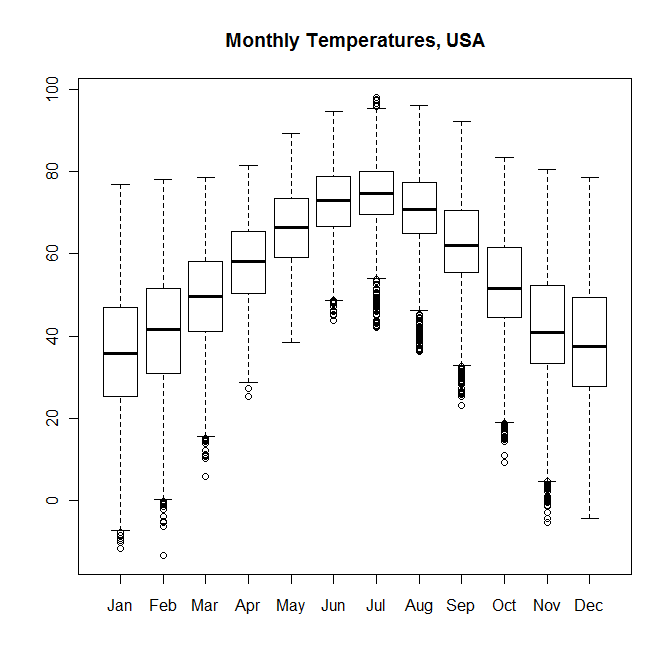
*Copy and paste it into a word document or powerpoint as a metafile.*

# 9. 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 Des Moines (dm) 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

To plot the average temperature by station, we could do the following:

boxplot(Jan~Station,data=fweather)



That's not incredibly useful because the order is all mixed up and the axis labels aren't readable.

Instead, we can calculate the January average by station and order the boxplot by that. This all is a little exotic for a beginner R class, so just know it's possible. Typically when working with R, you will start with a simple plot and keep adding and refining your code until you get it the way you like.

#calculate the average temperature in January for each Station, removing missing data

janavg <- tapply(fweather[,"Jan"],factor(fweather$Station),FUN=mean,na.rm=T)

#get the order of stations based on the January averages

coldToWarm <- names(janavg[order(janavg)])

#create a factor of the stations ordered by cold to warm

St <- factor(fweather$Station,levels=coldToWarm)

#set the margins so we can read the names

par(mar=c(9,3,3,3))

#make the boxplot, making labels perpendicular to axis, shrinking text size, adding colors from yellow to orange to red

boxplot(Jan~St,data=fweather,las=2,cex.axis=.6,main="Average January Temperatures by Station",col=colorRampPalette(c("yellow","orange","red"))(length(coldToWarm)))



# 10. 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 Des Moines?

lm(dm$Oct~dm$Period)

Call:

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

Coefficients:

(Intercept) dm$Period

57.571448 -0.001862

You could also store that in a variable if you want

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

What is that thing "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 nice way to examine a mysterious object in R.

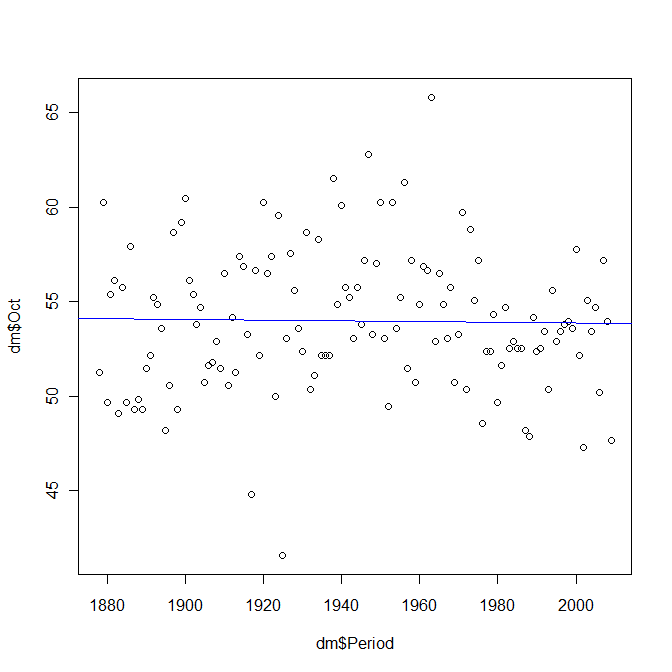
str(model)

Plot the regression line on the scatter plot.

plot(dm$Period,dm$Oct)

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

Not much increase in temperatures for Des Moines in October over the period.



Let's add a new column to our data frame with the mean monthly temperatures for each year. We can use the function *rowMeans()* to get the mean value for each row*.* To add a new column on, we can just assign it to a new name, in this case mn for mean.

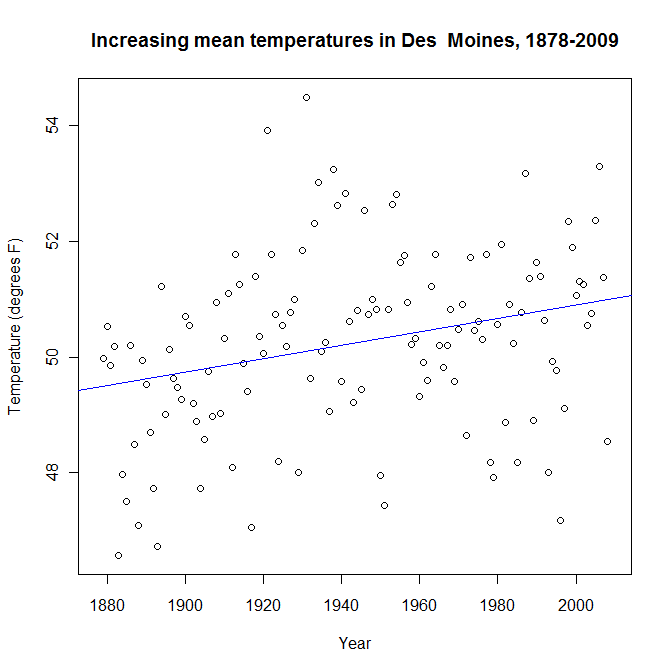
head(dm)

dm$mn <- rowMeans(dm[,7:18],na.rm=T)

If we look at the means of all the monthly temperatures, there is an increase.

plot(dm$Period,dm$mn, main="Increasing mean temperatures in Des Moines, 1878-2009",xlab="Year",ylab="Temperature (degrees F)")

abline(lm(dm$mn~dm$Period),col="blue")



*What is the slope of the regression line?*

*Use colMeans() to find the mean temperatures per month.*

## Highlighting points on a scatter plot

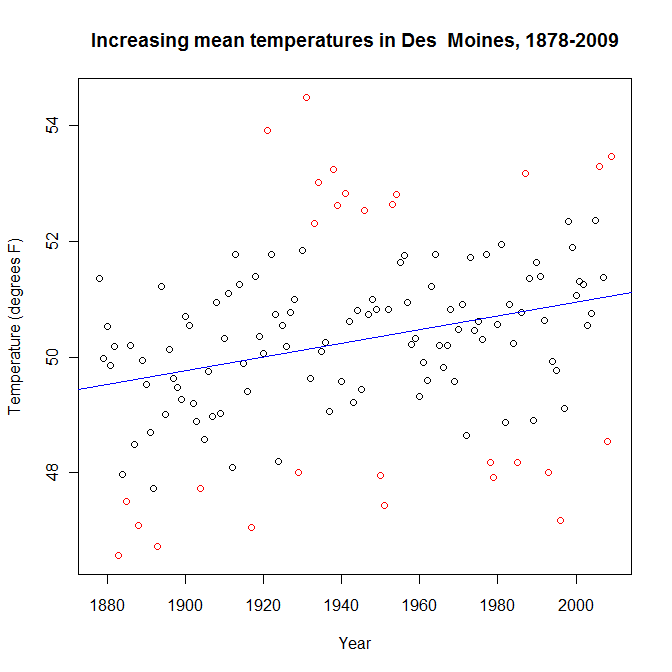
Highlight some outliers on the plot (based on residual, which is their distance from the regression line).

meanmodel <- lm(dm$mn~dm$Period)

iv <- abs(meanmodel$resid)> 2

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.

points(dm[iv,"Period"],dm$mn[iv], col="red")



Now that we have our index vector, we can use it to examine our data frame as well.

*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.*

dm[iv,]

# 11. Writing data out

Let's say we want to save some data out to a file. We can use write.table().

write.table(dm[iv,],"data.txt",sep="\t",quote=F,row.names=F)

# 12. T-test

Many statistical tests are available in R. A simple test is a t-test, which you can use to test whether the means of two samples are equal. Any difference between 2008 and 2009?

t.test(dm[dm$Period=="2008",7:18],dm[dm$Period=="2009",7:18])

Welch Two Sample t-test

data: dm[dm$Period == "2008", 7:18] and dm[dm$Period == "2009", 7:18]

t = -0.5701, df = 19.811, p-value = 0.575

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

-22.9758 13.1178

sample estimates:

mean of x mean of y

48.545 53.474

Just the p-value:

tt<- t.test(dm[dm$Period=="2008",7:18],dm[dm$Period=="2009",7:18])

tt$p.value

*Any significant difference between October and November? What's the p-value?*

# **13. 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.

dm[is.na(dm)]

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

# 14. Packages

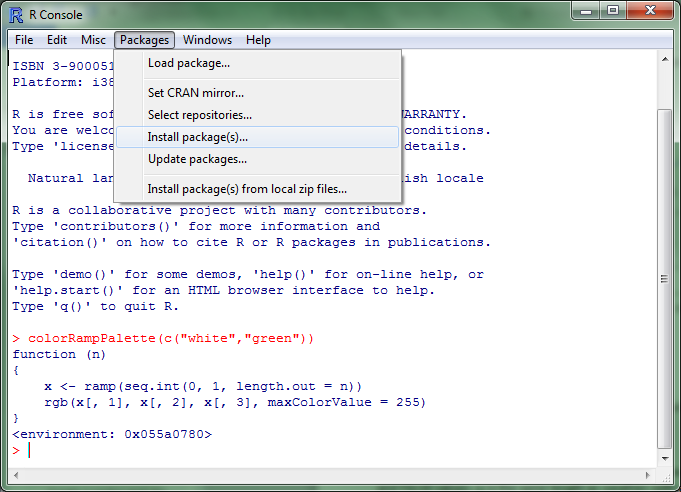
There is a lot of functionality built into R by itself (base), but there are also many interesting packages that have been created for R that you can use. In order to use these package, you will typically have to install them, and then call library(package) to load them into your current session.

There are two large repositories of packages you will likely use - cran and bioconductor. Cran is general 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 (warning, case sensitive), do the following: For a cran package:

install.packages("RColorBrewer")

Or through the menu (select nearby repository). In Rstudio, go to Tools, install packages.



For Bioconductor packages,

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

biocLite("limma")

Once a package is installed, load it into your environment with

library("limma")

You can see which packages you currently have available with

library()

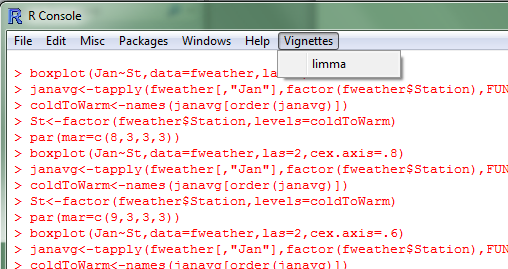
Or

sessionInfo()

You can get more info about a package with

help(package="limma")

Some packages have "vignettes" which show how to use the package.



*What is the ChIPpeakAnno package for?*

# 15. Colors

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

Colors can be referred to three different ways: text name ("yellow"), RGB values from 0 to 255 (255,255,0), or hexadecimal (#FFFF00). In this case, the hex number is a triplet where FF=255, FF=255, and 00= 0.

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

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

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

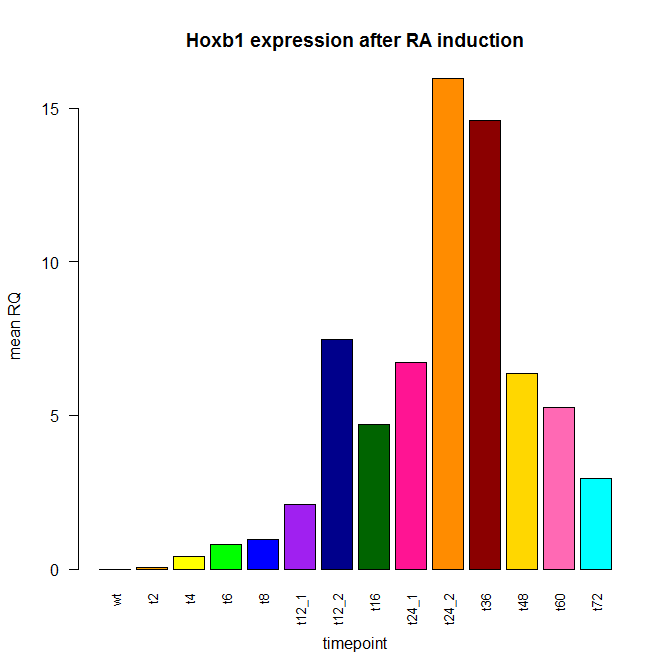
We need 14 colors for our plot.

length(hoxb1\_avg)

We could make our own palette from color names that we just look up

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

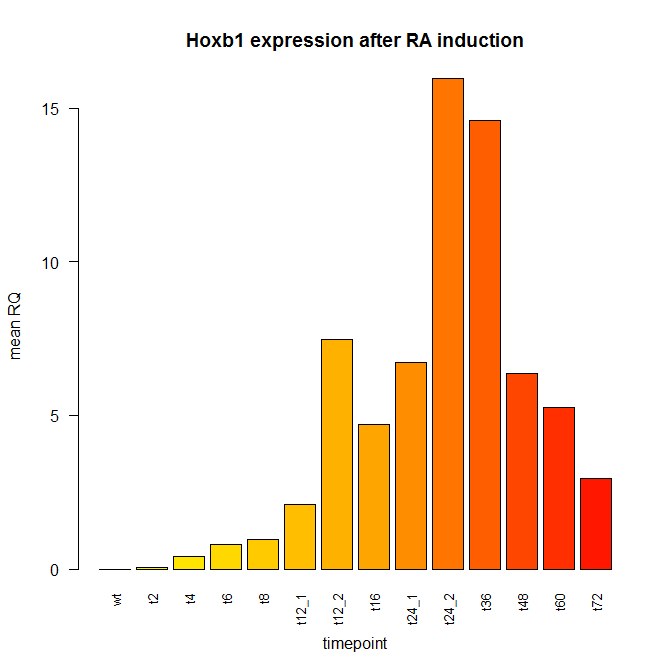
barplot(hoxb1\_avg,col=mypal,las=2,names.arg=names(hoxb1\_avg),ylab="mean RQ",xlab="timepoint",main="Hoxb1 expression after RA induction",cex.names=.8)



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 is colorRampPalette()(numcolors).

cols <- colorRampPalette(c("yellow","orange","red"))(15)

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 (in the training room). To install it, you would do:

install.packages("RColorBrewer")

Load the package into your current session:

library(RColorBrewer)

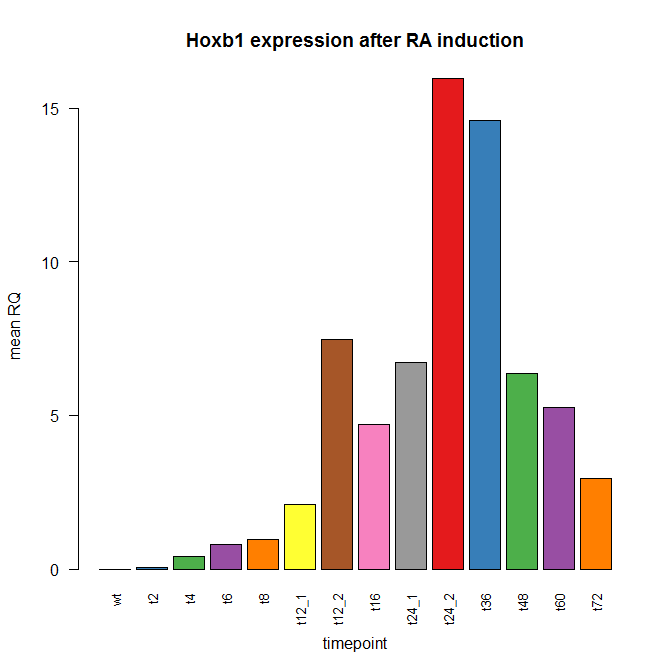
Show all the available palettes

display.brewer.all()

Let's use a categorical color palette from RColorBrewer:

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

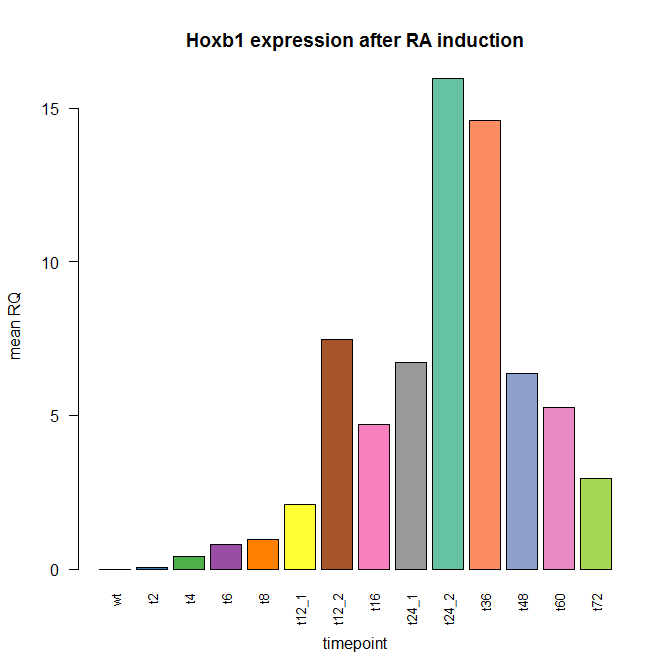
barplot(hoxb1\_avg,col=cols,las=2,names.arg=names(hoxb1\_avg), ,ylab="mean RQ",xlab="timepoint",main="Hoxb1 expression after RA induction",cex.names=.8)



Notice the colors are recycled if there are more variables than colors. 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 to pick out patterns. There are many possibilities to create clustered (or unclustered) heat maps in R using heatmap() and image().

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 a subset of columns from the data frame for plotting (I generated this using *dput(colnames(exp))*)

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)

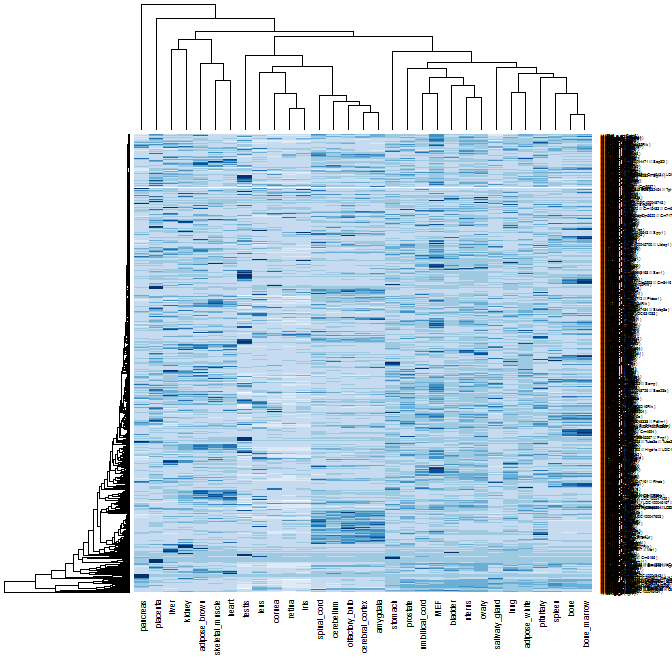
rownames(selected)<-paste(exp[,1],"(",names[match(names[,1],exp[,1]),2],")")

#set the colors

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

#create the heatmap

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



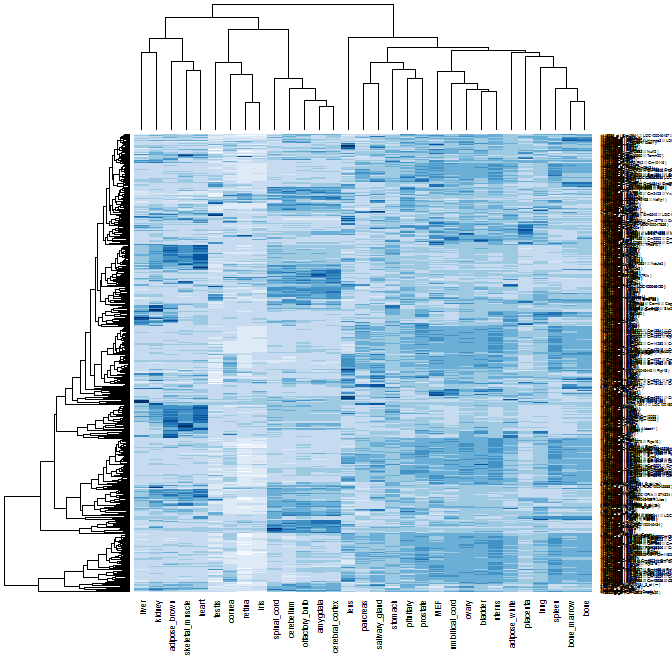
The default heatmap() command in R does a hierarchical clustering of the data and rearranges both rows and columns based on their distance from eachother. For more information, see ?heatmap, ?hclust and ?dist.

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 Des Moines weather data (dm[,7:18]). Don't forget as.matrix().*

# 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

Some examples taken from Gaye Hattem and Chris Seidel. Some other examples 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