Introduction to R

Contents

[1. Getting Started 2](#_Toc306954085)

[Download and install R 2](#_Toc306954086)

[Start R 2](#_Toc306954087)

[Working with R 2](#_Toc306954088)

[Set up for this tutorial 2](#_Toc306954089)

[Set your working directory 3](#_Toc306954090)

[Get help on a function 3](#_Toc306954091)

[Browse Documentation 3](#_Toc306954092)

[Web resources 3](#_Toc306954093)

[2. R as a calculator 4](#_Toc306954094)

[Entering commands 4](#_Toc306954095)

[Create a vector and store it in a variable 4](#_Toc306954096)

[Calculate some basic statistics on vectors 5](#_Toc306954097)

[Plot x vs y 6](#_Toc306954098)

[3. Data 6](#_Toc306954099)

[Read in some data 6](#_Toc306954100)

[Data frames 7](#_Toc306954101)

[Factors 8](#_Toc306954102)

[5. Scatter Plot, Histogram, Barplot 8](#_Toc306954103)

[More on scatter plots 8](#_Toc306954104)

[Multiple figure plots 9](#_Toc306954105)

[Histogram 9](#_Toc306954106)

[Create a barplot 10](#_Toc306954107)

[Improve the barplot 11](#_Toc306954108)

[4. For loops 12](#_Toc306954109)

[5. Multifigure Plotting 12](#_Toc306954110)

[5. Getting plots out of R 13](#_Toc306954111)

[Boxplots 14](#_Toc306954112)

[5. Linear Regression 16](#_Toc306954113)

[Plot the regression line on the scatter plot. 16](#_Toc306954114)

[Highlighting points on a scatter plot 18](#_Toc306954115)

[6. Packages 19](#_Toc306954116)

[7. Colors 19](#_Toc306954117)

# 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

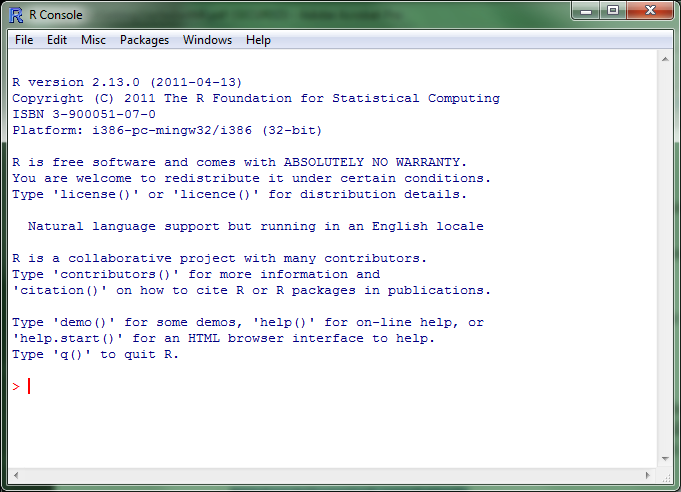
(click .pkg of latest version)

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 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 red should be what you enter into the R console, and in blue 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.

dir.create("H:/introR")

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

## 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 script you are currently working on.

setwd("H:/introR")

If you don’t explicitly set your working directory on windows, it will be set to

C:\Documents and Settings\mcm\My Documents *(Windows XP)*

or

C:\Users\mcm\Documents *(Windows Vista, Windows 7)*

## Get help on a function

*Try asking for help on setwd*

?mean

## Browse Documentation

help.start()

## Web resources

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

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

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

# 2. R as a calculator

## Entering commands

You can use R like a calculator

1+1

[1] 2

To explain the result, the [1] before the 2 is because R considers the result of a calculation a vector with one element. So the first element of the vector is labeled with [1].

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

*What is 2 to the 12th?*

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

?log2

## Create a vector and store it in a variable

A vector in R can be a list of numbers, of strings (letters/words), or of TRUE/FALSE values. You can use the c() function to **combine** elements into a vector.

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

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

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

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

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 flattened into one vector.

z<-c(x,y)

z

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

There is a shortcut in R, 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

*create a vector of words of your choosing*

*create a vector of numbers from 5 to 20*

m<-1:10

## Calculate some basic statistics on vectors

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

mean(x)

median(x)

min(y)

max(y)

#which.min returns the POSITION of the minimum element.

which.min(y)

which.max(y)

#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 vector the same size as x, with a TRUE everywhere that x is equal to 6.

x==6

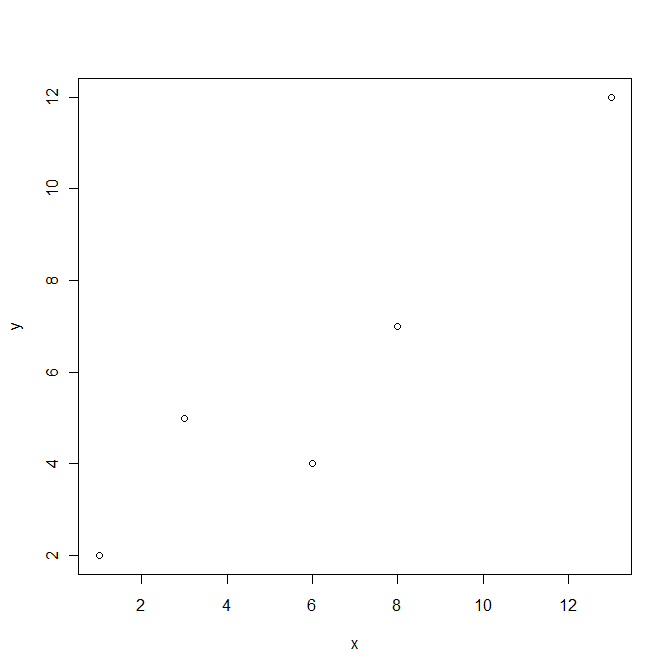
*what is the median of numbers 4 to 7?*

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

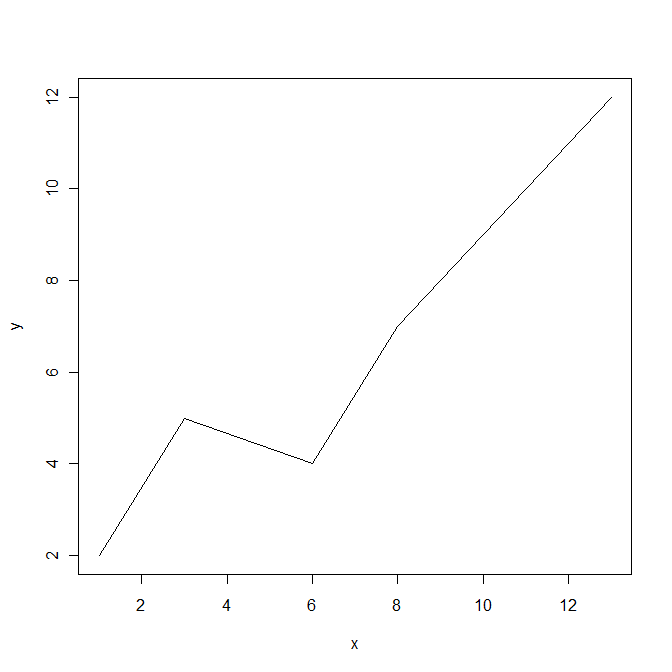
[1] FALSE FALSE TRUE FALSE FALSE

## Plot x vs y

plot(x,y)



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.*

# 3. Data

## Read in some data

df <-read.table("H:/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, you have some other options

* Edit it manually in Excel / Text editor if it's a text format and needs a minor change
* Edit it programmatically with perl / python / ruby
* Google refine? <http://code.google.com/p/google-refine/>
* Use a specialized R package to read in certain types of data (BAM, CEL, XML, etc)

At some point as you're learning R, you will likely run into problems reading in a data file. Try the following to troubleshoot:

* sep='\t'
* quote=""
* comment.char='""
* strip.white=T
* open in Excel or a text editor, look for weird characters or spacing problems, extra tabs, etc.
* Google the error
* If it breaks on a specific line, look at that line in the file
* try reading in the first few lines with nlines=5 or skipping the first line with skip=1

## 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:

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

The inner part df[,4]=="YIL162W" is actually 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 could do something similar to find all overlapping genes:

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

Or all genes without a close neighbor

*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)*

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

## 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. Scatter Plot, Histogram, Barplot

## More on scatter plots

Read in a data set listing temperature measurements across the US from 1851 until 2009.

weather<-read.csv("H:/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

read.csv is another function to read in a file specifically for comma-separated-values, you could also use read.table here with sep=','.

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

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

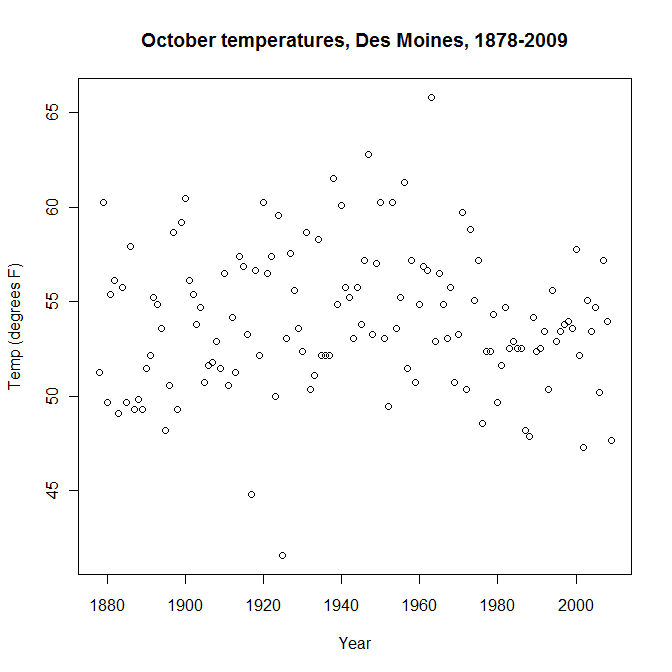
dm<-fweather[iv,]

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)")



## Multiple figure plots

You can plot multiple plots on one figure using par() to set a parameter before you start making the plots. The argument to use for multiple plots is mfrow.

par(mfrow=c(2,2))

## Histogram

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

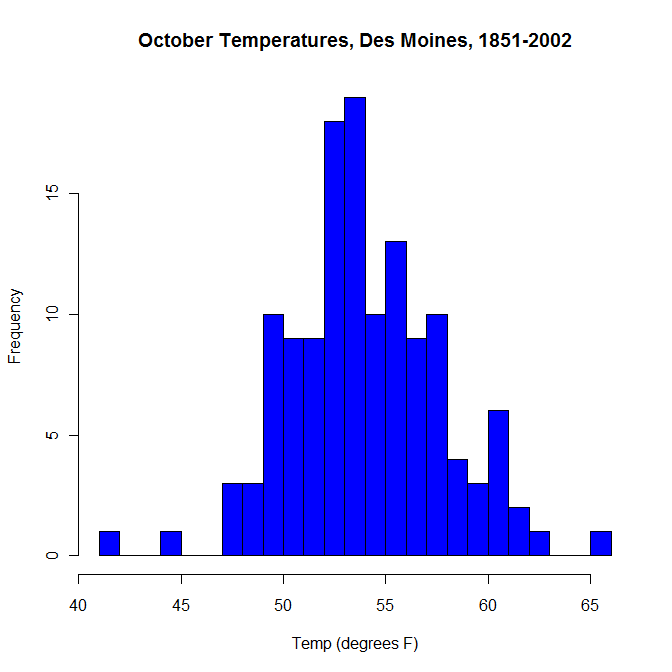
hist(dm[,"Oct"])

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")



## 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)

Change the column names (sorry, kind of messy)

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.

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

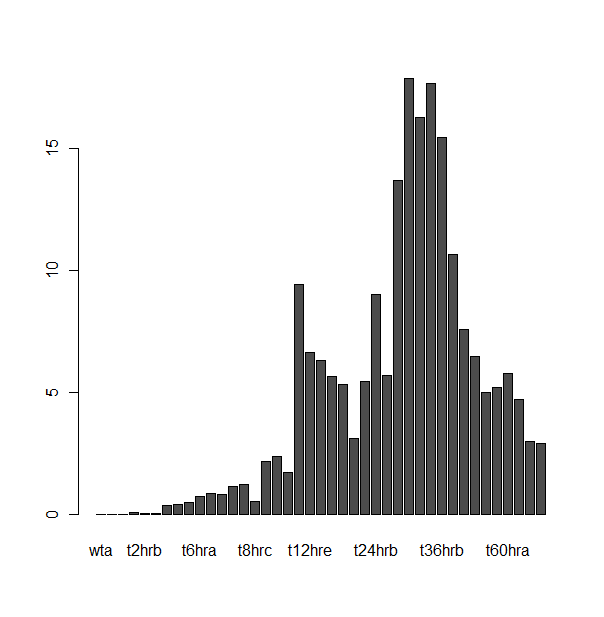
For a barplot, R expects a matrix or a vector. A matrix is like a data.frame, but all the columns are the same type - numbers or strings.

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.

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

barplot(as.matrix(hoxb1))



## Improve the barplot

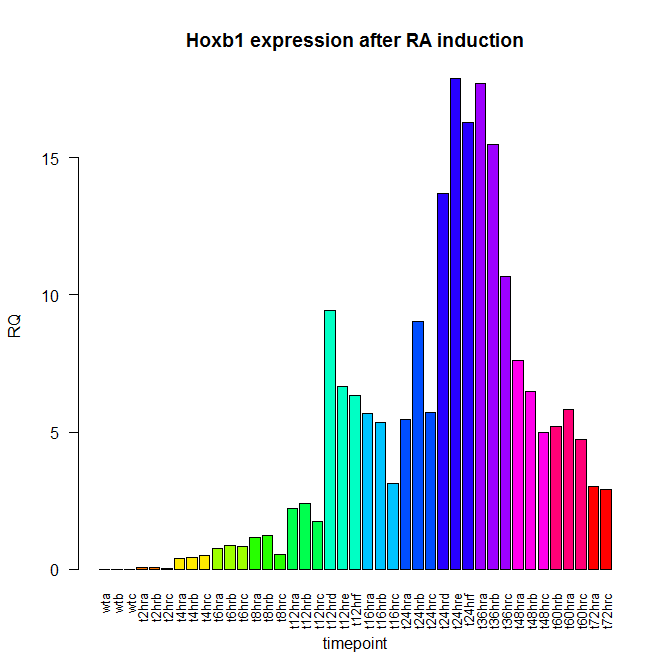
We made a basic barplot, but let's fix a few things. Let's make prettier colors, for one thing. rainbow() is a quick way to get a list of colors of a certain length (more about colors later). rep() is a way to repeat a certain sequence, in this case using the each parameter to repeat each color 3 times for each 3 replicates.

cols<-rainbow(13)

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

Barplot with colors, make labels horizontal, add axis labels and a title.

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



# 4. For loops

A for loop is a way of doing the same thing over and over on different columns or rows of a data set.

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)

}

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

# 5. Multifigure Plotting

It's easy to put multiple plots on one figure

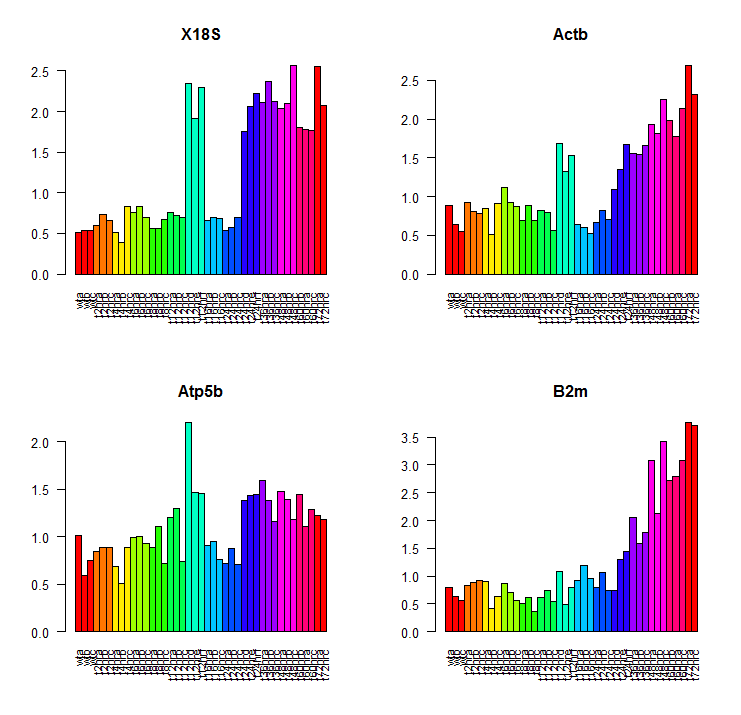
par(mfrow=c(2,2))

for(i in 1:4)

{

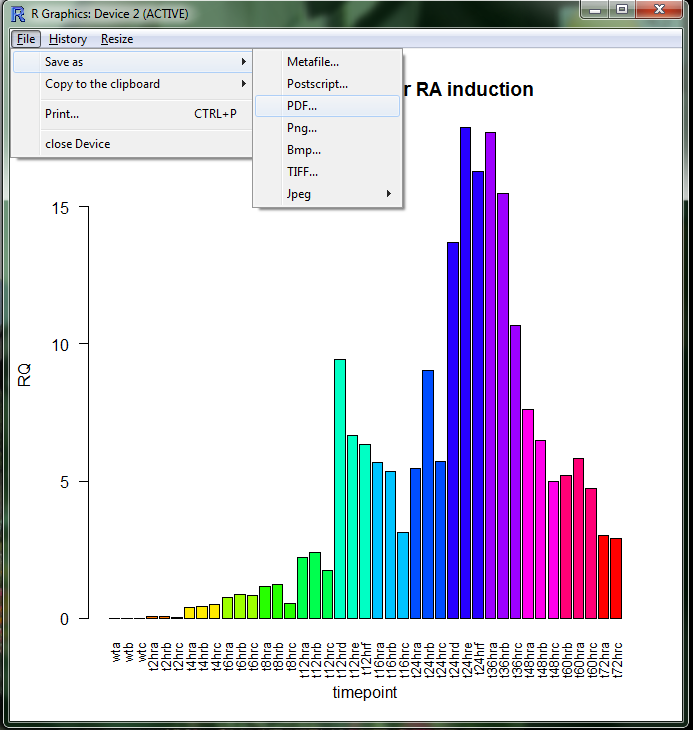
barplot(t(hox[i,2:42]), beside=T, names.arg=colnames(hox)[2:42], col=cols, main=hox[i,"gene"],las=2,cex.names=.8)

}

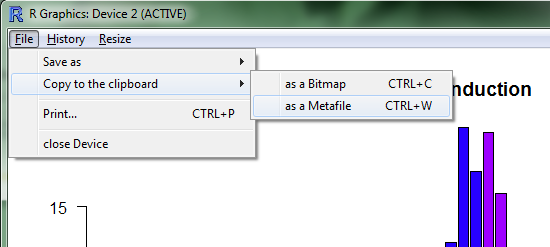


# 5. Getting plots out of R

To save a plot to a PDF, you could use the menu - File - Save As.



Or to copy to the clipboard.



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.

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

Rather than saving your plots 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)

}

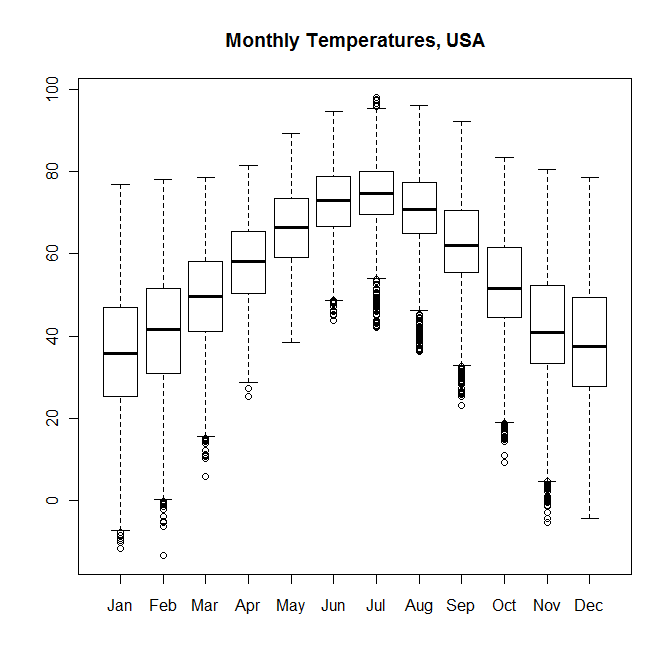
dev.off()

# 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.

The parts of a boxplot are:



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)))



# 5. 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)

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? An object.

class(model)

[1] "lm"

What is in it? str is a nice way to examine a mysterious object in R, we'll see more of those later.

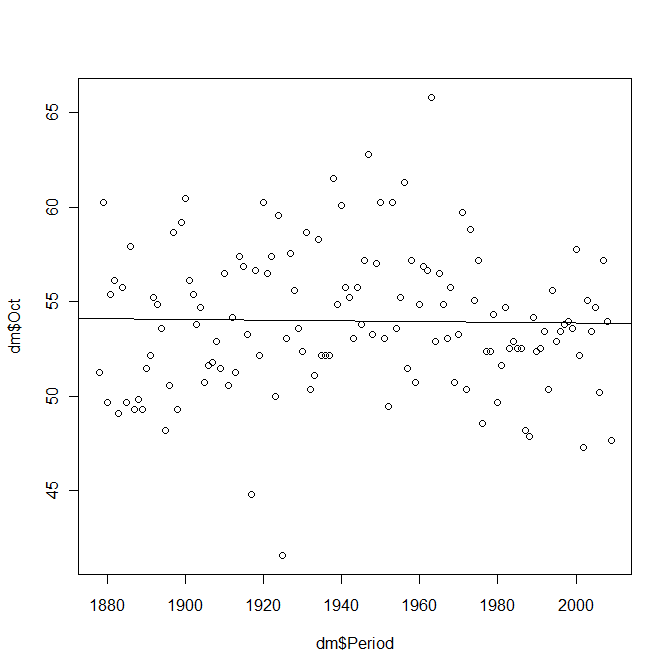
str(model)

## 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.

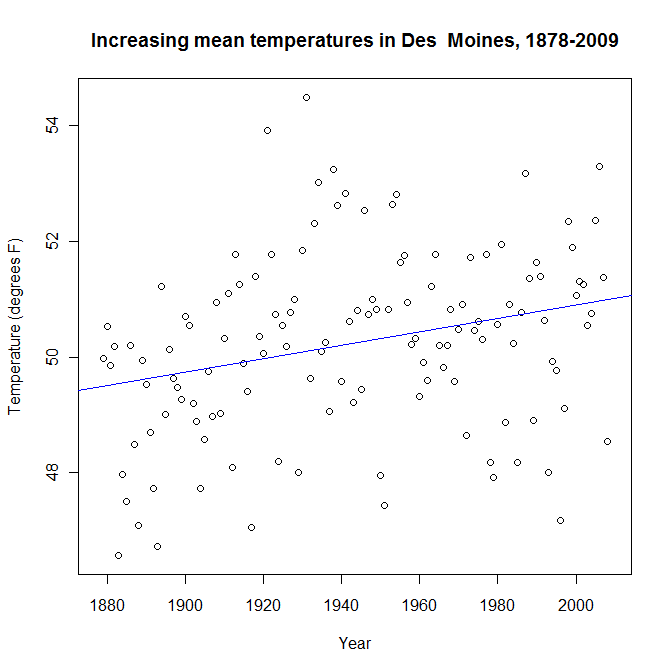
head(dm)

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')



## Highlighting points on a scatter plot

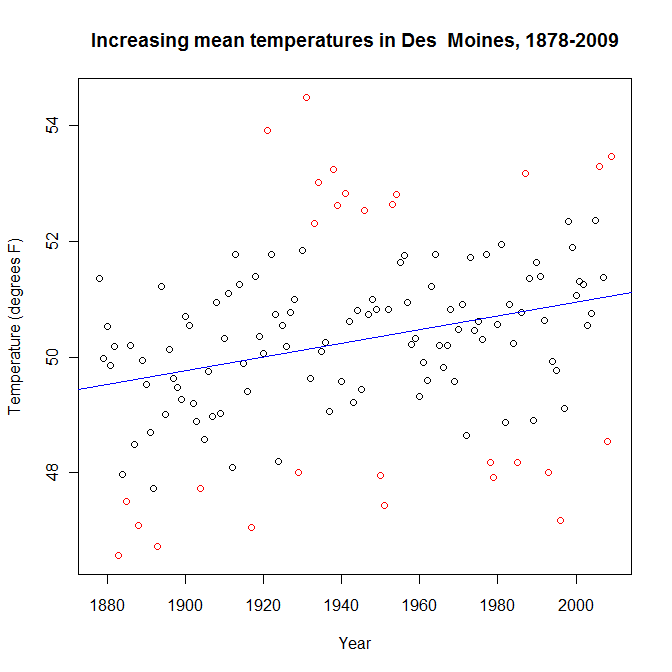
Highlight some outliers on the plot (based on residual, which is their distance from the regression line).

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

iv<-abs(meanmodel$resid)> 2

iv is what we call 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,]

# 6. Packages

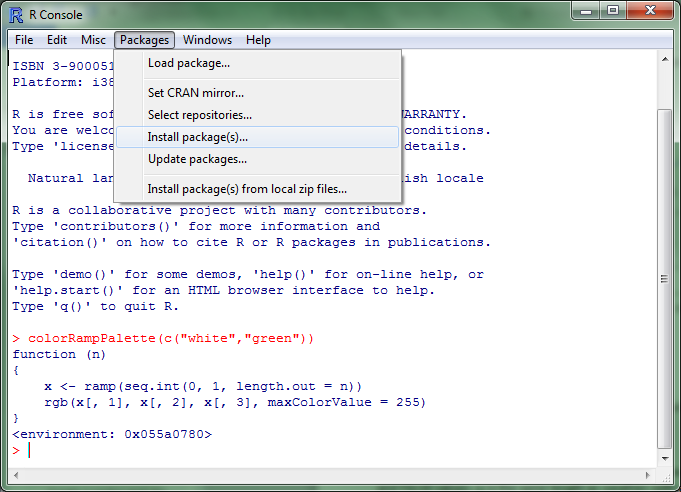
There is a lot of functionality built into R by itself, but there are also many interesting packages that have been created for R that you can use. In order to use these package, you will typically have to install them, and then call library(package).

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)



For Bioconductor packages,

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

biocLite("limma")

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

library("limma")

You can see which packages you currently have available with

library()

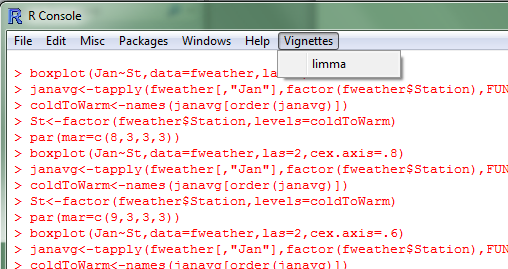
Or

sessionInfo()

You can get more info about a package with

help(package="limma")

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



# 7. Colors