R Bootcamp

Gene Hunt
NMNH, Smithsonian Institution
Analytical Paleobiology Workshop
Gainesville, FL July 2018

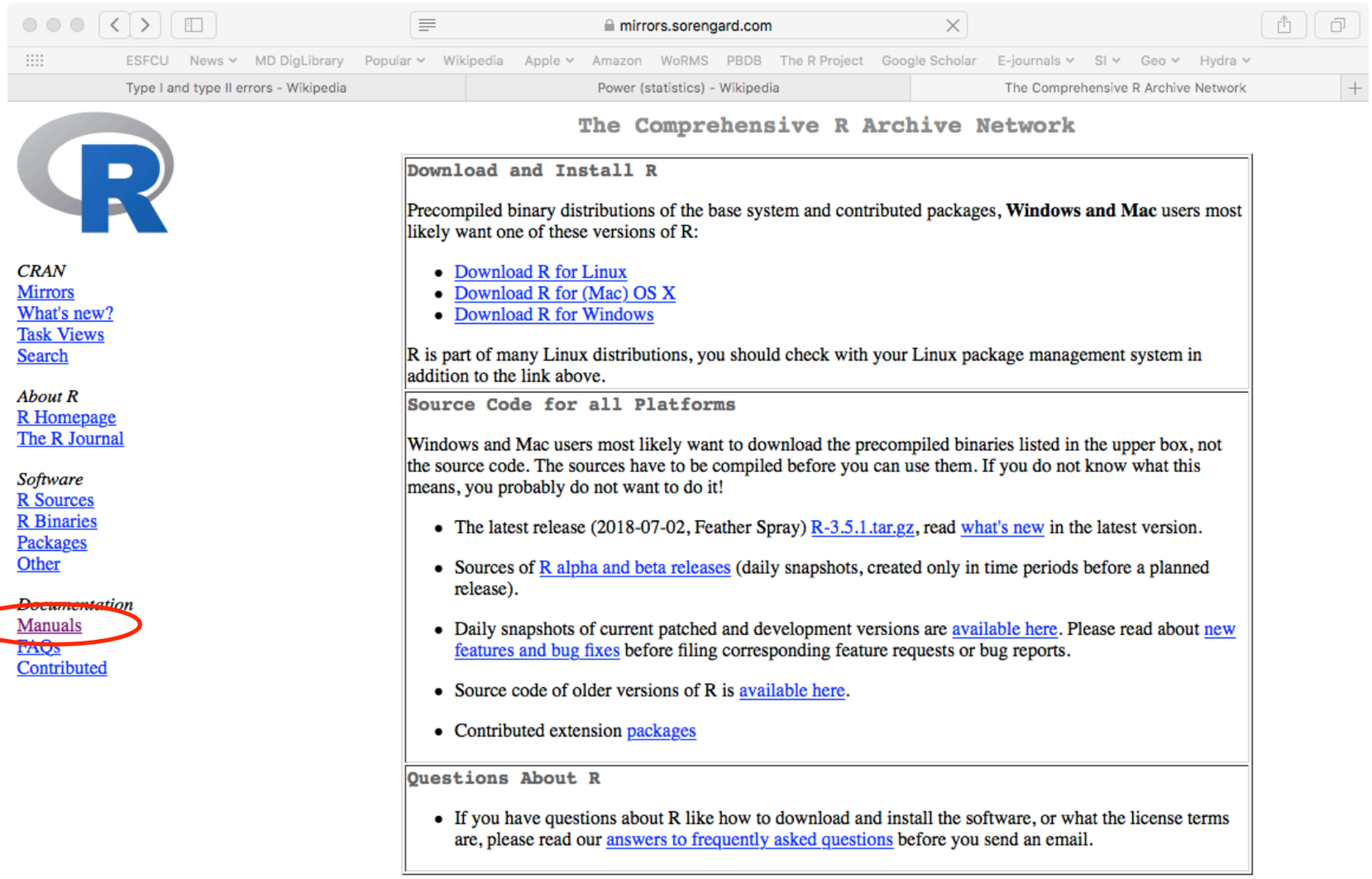
Purpose

- Get comfortable with R
- R commands, syntax, rules
- R programming

Background on *R*

- Descended from *S* (Bell Labs); both *S* and *R* are used heavily by statisticians
- Open source, maintained by a volunteer committee
- Practical Benefits:
  - Free, available for all major OS
- Scientific Benefits:
  - High level (powerful functions built-in)
  - Powerful statistical, graphics capabilities
  - Extendable (user contributed packages)

Getting additional help

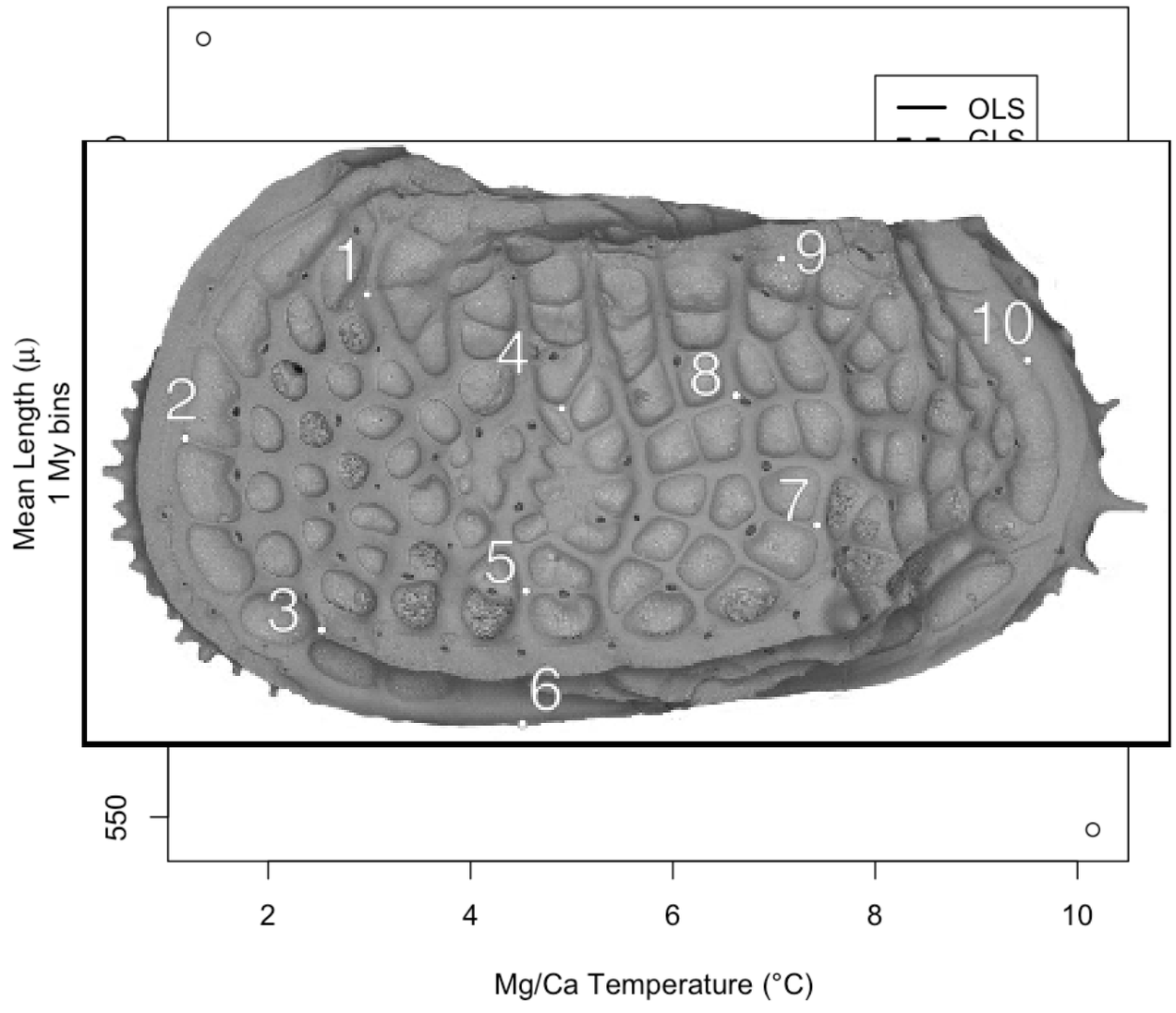


~~"An Introduction to R"~~

not very useful; go to "contributed documentation" > Maindonald

Ways to use *R*

- As a statistics package (ANOVA, nonparametrics, ordinations)
- As a programming language (resampling, morphometrics, likelihood approaches)
- Phylogenetic comparative methods
- For publication graphics



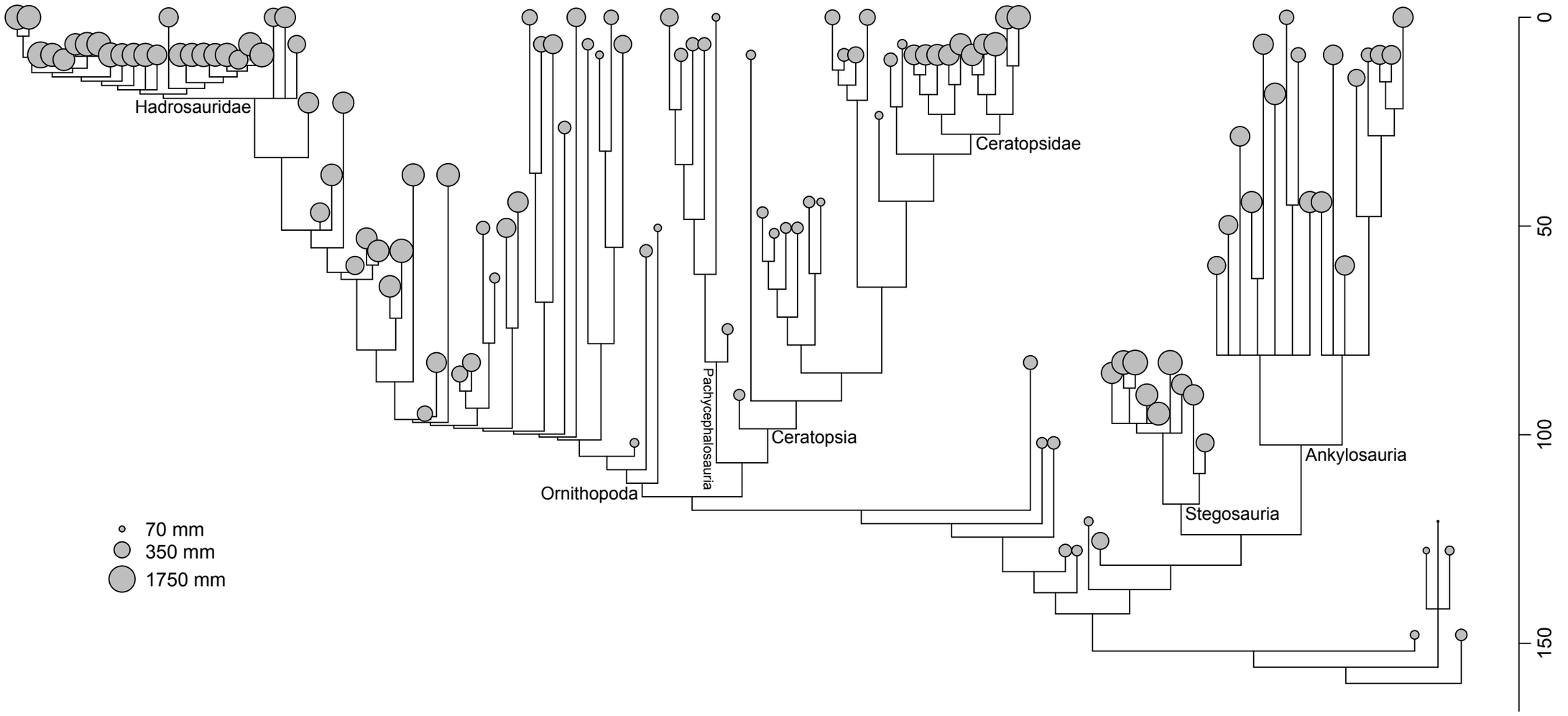
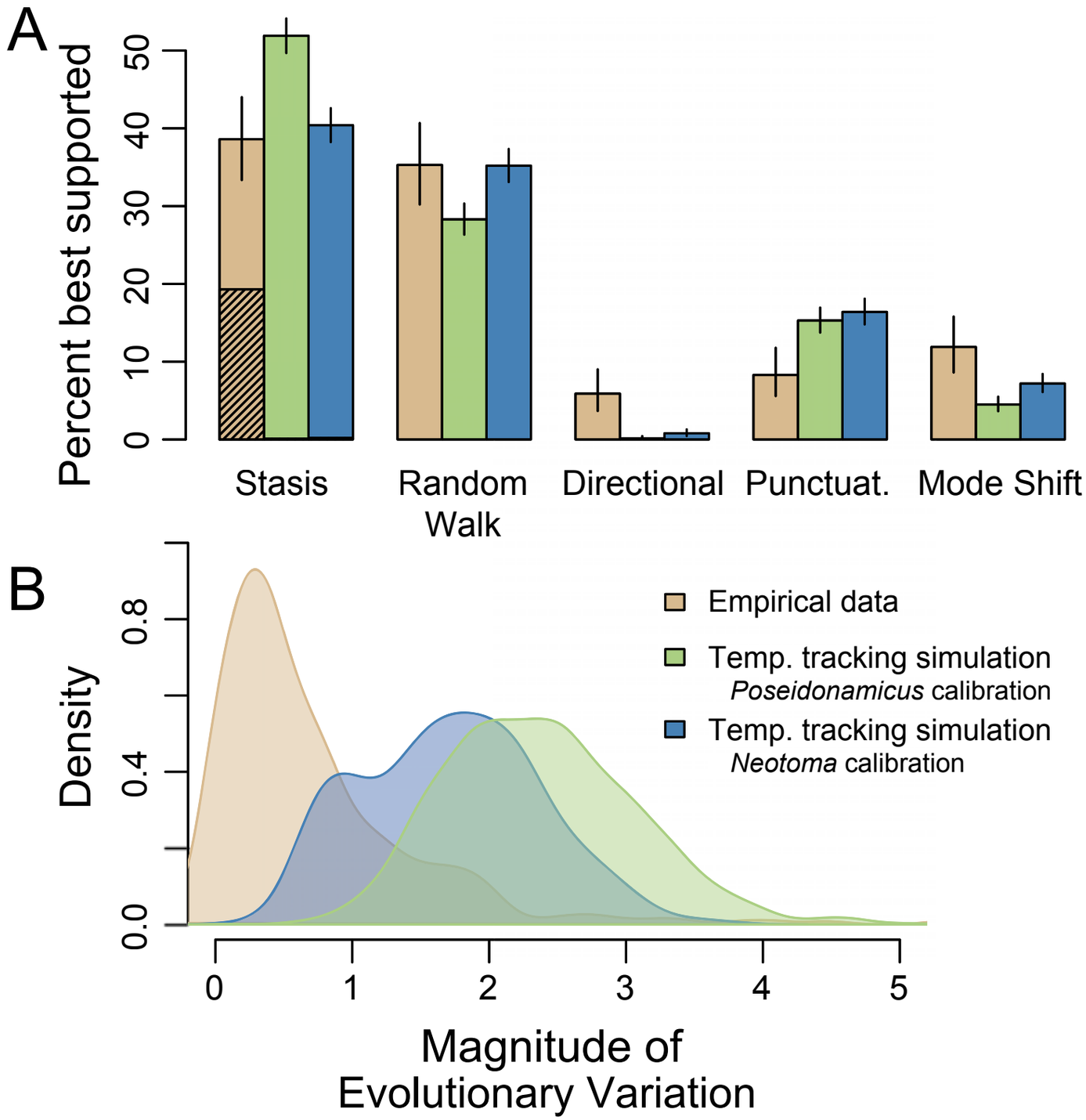
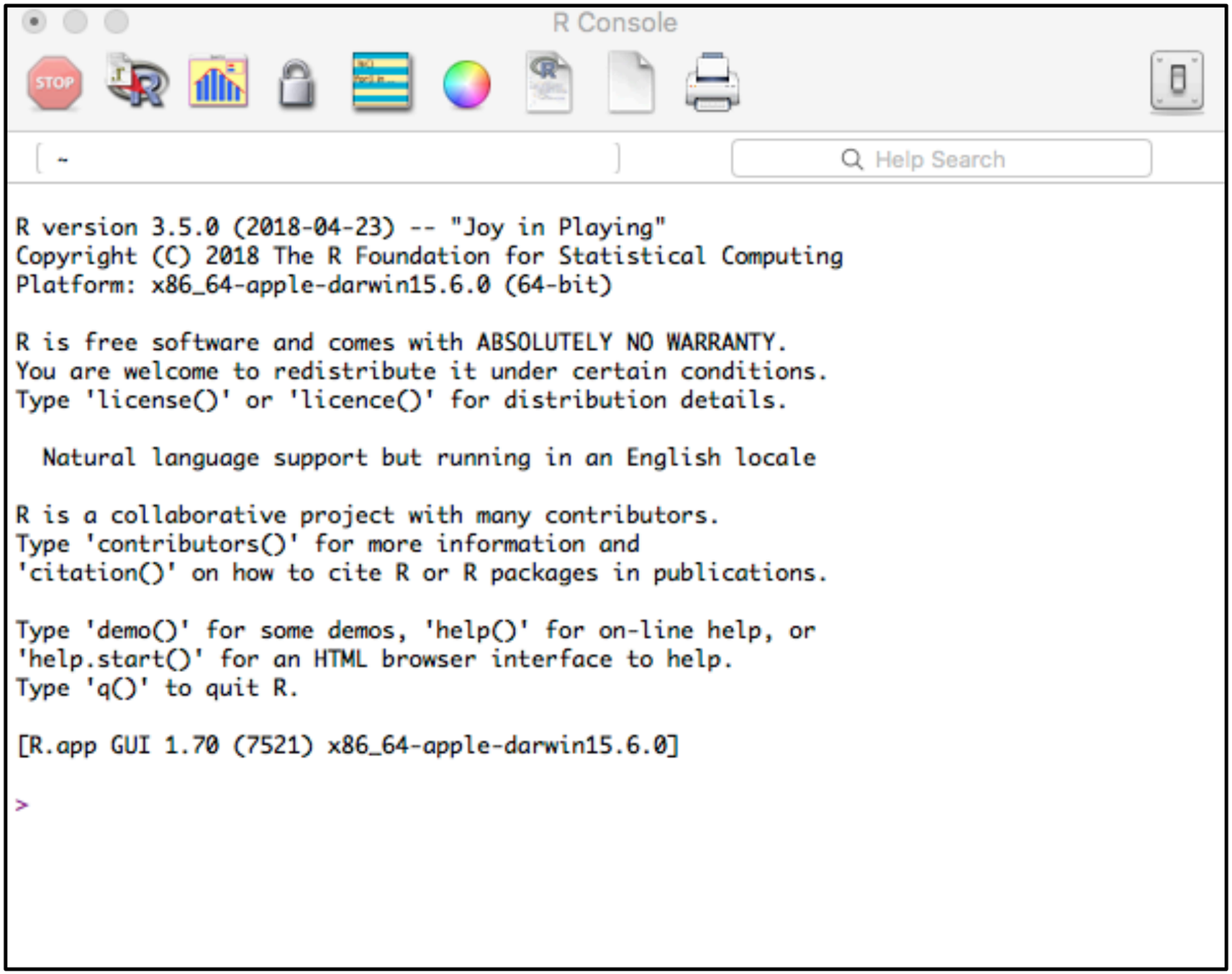


FIGURE *5*.—Phylogeny of ornithischian dinosaurs from the composite tree of Carrano (2006) modified according to other studies (Ford and Kirkland, 2001; Weishampel et al., 2003; Novas et al., 2004; Vickaryous et al., 2004; Averianov et al., 2006; Ryan, 2007; Carpenter et al., 2008; Maidment et al., 2008; You et al., 2008; Arbour et al., 2009; Boyd et al., 2009; Dalla Vecchia, 2009; Sues and Averianov, 2009; Butler et al., 2010), with branch lengths scaled to geological time. Taxon names are omitted for clarity, although a few of the larger named clades are labeled below the appropriate node. Symbol sizes are scaled to log femur length. Time axis extends backwards from the youngest terminal taxa, in millions of years.


A
Percent best supported
0
10
20
30
40
50
Stasis
Random Walk
Directional
Punctuat.
Mode Shift
B
Density
0.0
0.4
0.8
Empirical data
Temp. tracking simulation
Poseidonamicus calibration
Temp. tracking simulation
Neotoma calibration
0
1
2
3
4
5
Magnitude of Evolutionary Variation

The *R* Session



The Interactive R prompt

```
> 2                    2

> 2 + 2                4
```

Saving information as variables

```
x <- 8     # assigns variable, x, a value of 8
```

Assignment operator

```
x * 5          40

y <- “tooth”
```

Variable assignment

1. Can also use = and -> as assignment operator

```
y <- 18

y = 18

18 -> y
```

2. Names in *R* are case-sensitive

```
tri <- 18

Tri <- 25  # these are different variables
```

Types of variables

Mode	Example
Numeric	`5, 2.34`
Character	`“Ostracoda”, ‘b’`
Logical	`TRUE, FALSE, T, F`
Factor	`{for categorical data}`
Complex	`2 + 3i`

Action	Symbol	Example
Arithmetic	+ - * / ^	2 + 2 5 * 4 + 1 ^12 5 * (4 + 1)^12
Assignment	<-	x <- 2 + 3

- R uses standard rules for order of operations: ^ before */ before +-
- To group operations, or if unsure of the rules, use parentheses

Exercise 1. Variable Assignment

1. Create a variable, `x`, and assign it a value of 46. Create a new variable, `xx`, that is equal to the one-half of x, raised to the 4th power.
2. Create a variable, `y`, and assign it a value of TRUE. Then assign y a value of “Cambrian”.

Functions

Functions take information (=arguments), do something with it, and return a result

```
sqrt() #computes the square root of its argument
sqrt(25)  5

seq() generates regular sequences of numbers
seq(1,5)  1 2 3 4 5
1:5       1 2 3 4 5    # special shortcut
```

`seq(1,10)`	Arguments specified by order
`seq(from = 1, to = 10)` `seq(to = 10, from = 1)`	Arguments specified by name
`seq(10, 20, 2)`	Third argument is increment
`seq(10, 20, by = 2)`	Can mix order and name specifiers

Using the Help

- If you know the function name

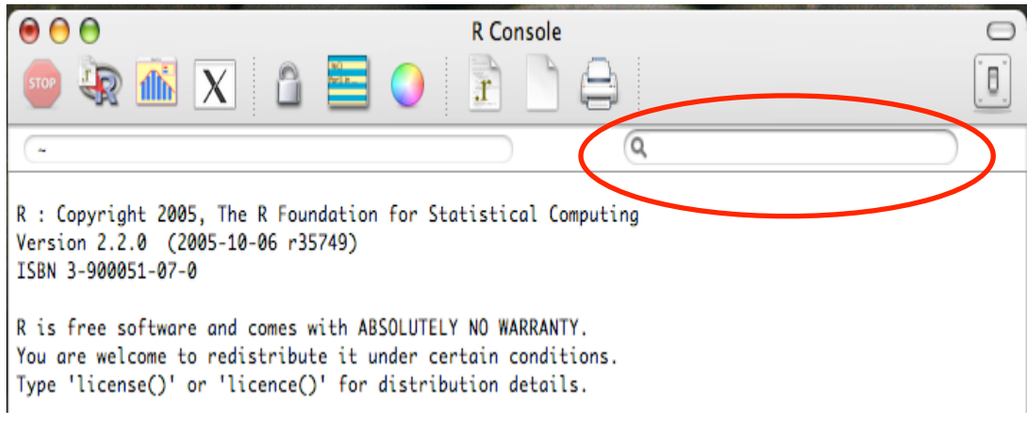
```
help(function)    # or, ?function
example(function) # shows the example from help
```

- To search for help on a topic

```
help.search("topic")  # e.g., help.search("covariance")
                      # same as ??covariance
```

Easier Way

- Mac OS: use Spotlight toolbar
- Windows: use HTML Help



function name {package}

short description

how to use it

what arguments mean

more info

example

R Help

< > Print Help Search

sd {stats} R Documentation

Standard Deviation

Description

This function computes the standard deviation of the values in `x`. If `na.rm` is `TRUE` then missing values are removed before computation proceeds.

Usage

```
sd(x, na.rm = FALSE)
```

this argument has a default

Arguments

`x`	a numeric vector or an R object which is coercible to one by `as.double(x)`.
`na.rm`	logical. Should missing values be removed?

Details

Like `var` this uses denominator *n - 1*.

The standard deviation of a zero-length vector (after removal of NAs if `na.rm = TRUE`) is not defined and gives an error. The standard deviation of a length-one vector is NA.

See Also

`var` for its square, and `mad`, the most robust alternative.

Examples

```
sd(1:2) ^ 2
```

[Package *stats* version 3.3.2 Index]

Combining elements into arrays

- Arrays are regular arrangements of multiple elements
- Must be the same type (e.g., all numbers)
- Vectors are 1D arrays, matrices are 2D arrays.

```
x <- c(10,14,35,50)  # concatenate function
```

1st 2nd 3rd 4th elements of x

```
length(x)  4     # number of elements in x
```

Extracting/subsetting element(s) from a vector using [ ]

```
x[2]        14
x[2:3]      14 35
x[c(2,4)] 14 50
```

c() is needed to specify a vector

2D Arrays: Matrices

```
X <- matrix(1:6, nrow=3, ncol=2)
```

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

```
X[1,2] element in 1st row, 2nd column
X[1, ] whole 1st row of X
X[ ,2] whole 2nd column of X
```

Some useful functions

```
nrow(X)   3    # number of rows
ncol(X)   2    # number of columns
dim(X)    3 2     # dimension = c(nrow, ncol)
```

```
X <- matrix(1:6, nrow=3, ncol=2)
X <- array(1:6, dim=c(3,2))  # same result
```

Operations on Arrays

- Many operations and functions can be applied to numbers, vectors and matrices.
- Operations are usually done element-wise

```
x <- 1:4        1  2  3  4
x+5             6  7  8  9
sqrt(x)         1  1.41  1.73  2
```

Action	Example
Combine elements into an array	`c(1,2,10,22)`
Make a matrix	`matrix(1:4, nrow=2, ncol=2)` `array(1:4, dim=c(2,2))`
Subset a vector	`y[3]  # 3rd element of vector y`

Subset a matrix	`X[1,4] # 1st row, 4th column`
	`X[3, ]  # whole 3rd row`
	`X[ ,4]  # whole 4th column`

Exercise 2. Arrays and matrices

1. Create a matrix, `XX`, with 3 rows and 4 columns from the numbers `1:12`. Take the 3rd row of this matrix, and assign it to a new variable, `y`.
2. Create a new vector of same size as `y`, with each element 10 times the corresponding value in `y`.
3. What does the following command give you: `dim(XX)[2]`? Reason it out first, then check your answer.

Lists

- Used to combine different types of information into a single object

```
w <- list(“yes”, 32, TRUE)

w[2]   32   # just like a vector
```

- List elements can be accessed by name

```
me <- list(firstname = “Gene”, id=40172)

me$firstname    “Gene”
me$fir          “Gene”   # can shorten, if unambiguous
me[1]           “Gene”
me[[1]]         “Gene”   # drops names attributes
```

Dataframes

- Rectangular table of information, not necessarily of the same type (numbers, text, etc).
- Usually, this is the form your data will be in when you import it

	habitat	taxon1	taxon2	taxon3
site1	forest	0	20	10
site2	forest	34	3	44
site3	grassland	23	1	112
site4	grassland	0	5	67

Column names

`colnames()`

Row names

`rownames()`

abund =

	habitat	taxon1	taxon2	taxon3
site1	forest	0	20	10
site2	forest	34	3	44
site3	grassland	23	1	112
site4	grassland	0	5	67

- Rows and columns can be accessed by number, like a matrix

```
abund[1,3]     10
abund[2, ]     # all of 2nd row
```

- Columns can be accessed by name, like a list

```
abund$taxon2   # all of 2nd column
```

- Use `attach()` to access variable names directly

```
attach(abund)
taxon2         # all of 2nd column
detach(abund) # undoes the attach()
```

- Some datasets are built-in to R for purposes of illustration.
- They can be accessed using the data() function, which makes the objects available in the **workspace**.

Exercise 3. Dataframes

1. Make the dataset `mtcars` available by typing `data(mtcars)`. This dataset summarizes information about models of cars in 1973. Take a look a the data by typing `mtcars`. Note that row names are the car models and the columns are attributes of the car models (a description of what the variables mean can be found by typing `?mtcars`).
2. How many rows and columns are there in this dataset?
3. Use the functions `mean()` and `median()` to calculate the mean and median of fuel efficiency (`mpg`) and car weights (`wt`, in thousands of pounds).
4. How would you save to a different variable the following subsets of this dataframe: (a) columns 1 and 2; (b) columns 1 and 4; (c) the first ten rows; (d) the first ten rows and the first three columns.
5. Make a new vector called `RelPower` which is equal to the horsepower of a car (`hp`), divided by its weight (`wt`).

Testing Relationships

Greater than, less than	> , <
Greater or equal to, less than or equal to	>= , <=
Equal, not equal	== , !=
AND, OR	& , |

```
x <- 4

x > 10          FALSE

x <- c(4,8,30,52)

x > 10                  FALSE FALSE TRUE TRUE
x < 50                  TRUE  TRUE  TRUE FALSE
x > 10 & x < 50         FALSE FALSE TRUE FALSE
```

Subsetting vectors

```
x <- c(4,8,30,52)
```

What if we want only those x *greater than 10?*

1. Choose by their indices

```
x[c(3,4)]          30 52
x[c(4,3)]          52 30  # order is respected
```

2. Using logical (T/F) vectors

```
x[c(FALSE,FALSE,TRUE,TRUE)]          30 52

x > 10             FALSE FALSE TRUE TRUE

x[x > 10]          30 52
```

Subsetting vectors

3. Using the elements' names

```
x<- 1:4
names(x)<- c('a', 'b', 'c', 'd')  # letters[1:4]
x     a b c d    # prints with names above
      1 2 3 4
x['b', 'c']       b  c  # prints with names above
```

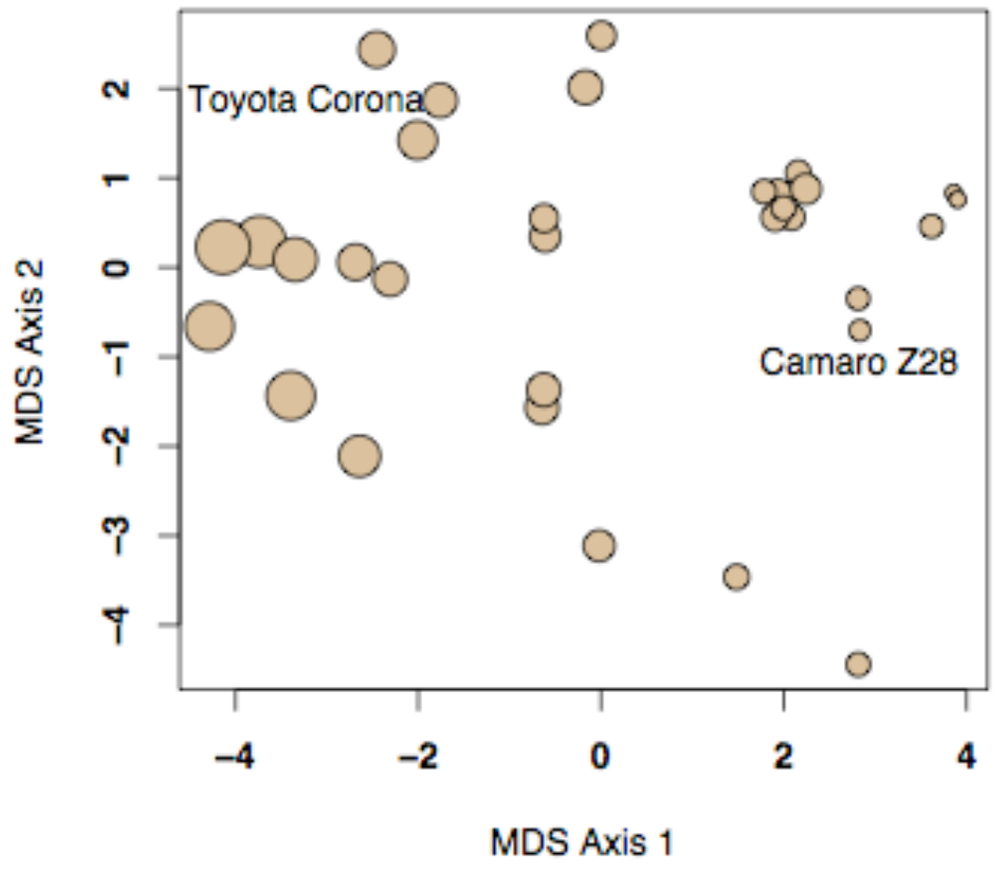
Greater than, less than	`>, <`
Greater or equal to, less than or equal to	`>=, <=`
Equal, not equal	`==, !=`
AND, OR	`&, |  (&&, ||)`
Subset by indices Subset by logical	`x[c(1,4)]` `x[x > 10]`

Exercise 4. Subsetting

1. Continuing with the `mtcars` dataset, compute separately the `mean` fuel efficency (`mpg`) of big cars (>3,000 pounds), and small cars (<3,000 pounds). Note that `wt` is in units of thousands of pounds.
2. Extract a vector of horsepower (`hp`) values for cars that are both small (`wt`<3) and fuel-efficient (`mpg`>=25).
3. Create a logical vector called `muscle` that is `TRUE` when a car has high horsepower (`hp>200`). The function `rownames()` returns the row names for a matrix or data table. Use this function, along with your `muscle` vector, to return the model names of the powerful cars.

Graphing

- Powerful graphing capabilities
- Can be saved as vector graphics (PDF, postscript)
- Can add to a plot, but can't edit what's already there (not clickable)



MDS ordination of 1973 Car data

`dot size ∝ mpg`

0th rule of data analysis
Plot your data! (J. Sepkoski)

Making Graphs

Generate some fake data

```
x <- rnorm(n=20,mean=0,sd=1)          # random normal numbers
                                      # same as rnorm(20,0,1)
                                      # same as rnorm(20)

y <- rnorm(n=20,mean=100,sd=10)
```

Try some plots…

```
plot(x)                               plot(x,y)
plot(x, type="l")                     plot(x,y, pch=2)
plot(x, type="b")                     plot(x,y, pch=3, col="red")
```

Other common graphing functions

```
hist()          pie()
barplot()       contour()
boxplot()
```

Common arguments to graphing functions

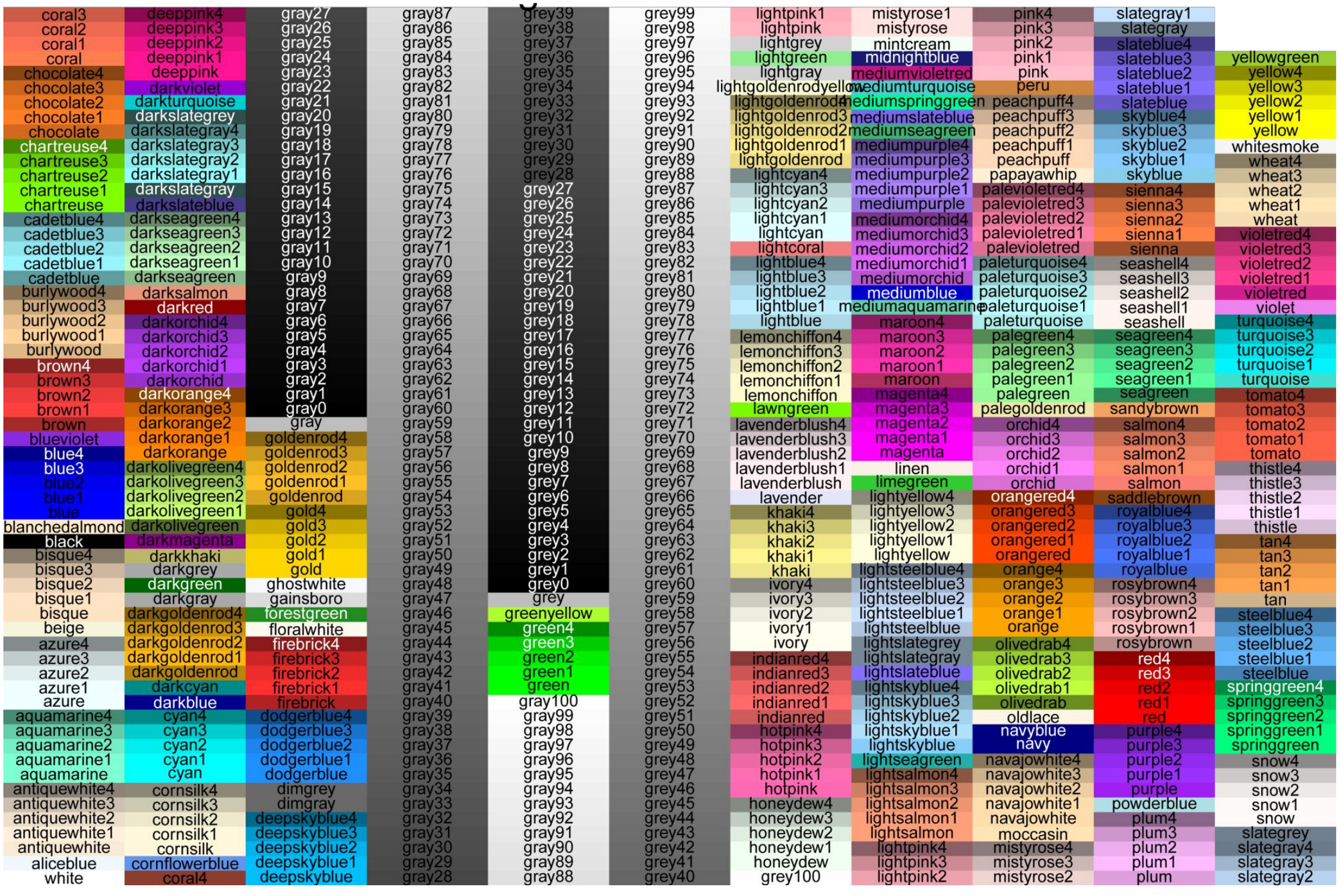
`col`	Color	`col = "red", col = 2`
`cex`	Character size	`cex = 2   # twice as big`
`pch`	Plotting symbol	`pch = 5   # diamonds`
`lty`	Line type	`lty = 2   # dashed`
`log`	Log-scale axes	`log = "x", log = "xy"`

Things you can add to existing plots

```
title()         legend()
abline()        arrows()
points()        segments()
text()          rect()
polygon()       symbols()
```

More about colors

```
colors()   # gives list of 657 color names
```



See R color cheat sheet

More about colors

Colors and RGB

```
rgb(red, green, blue, alpha)  # specify color by RGB components

col2rgb(“pink”)
```

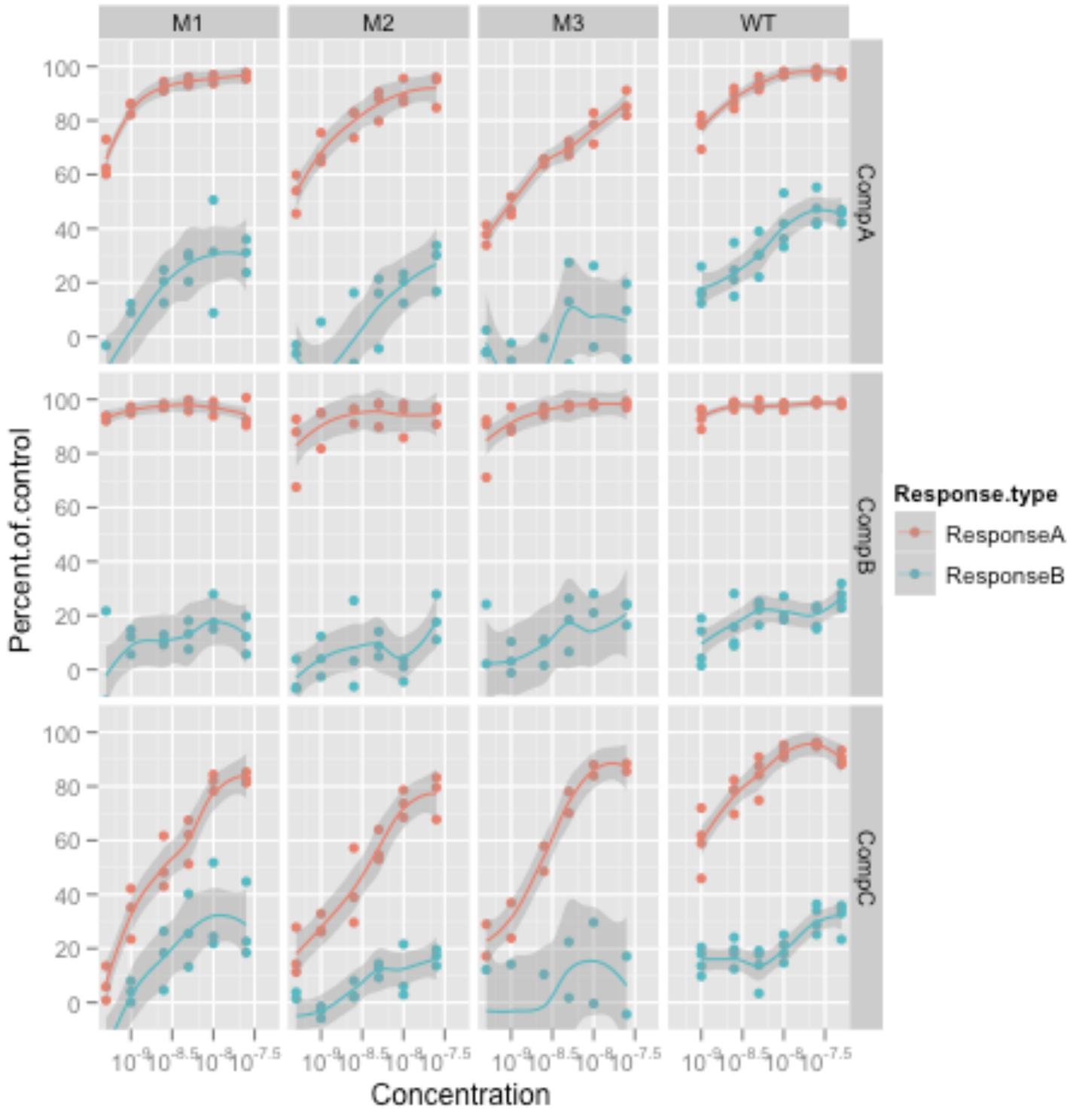
Color Palettes

```
palette()                     # tells you what color is 1, 2, etc.
                              # can also use to set these values

example(rainbow)              # sets color ranges
                              # see also colorRamp()
```

See R color cheat sheet

Alternative paradigm: package ggplot2 of the ‘tidyverse’



Interacting with plots

`locator()`	Gives x,y coordinates of clicked location

`identify()`	Identifies data point nearest to click
	`identify(x,y) # returns index of clicked points`
	`# when done, right-click (Win) or`
	`# ESC (Mac)`

Varying parameters for different points

Many graphing arguments can be a vector of the same length as the data.

This feature can be used to designate different plotting symbols, colors, etc. for different groups.

```
data(mtcars)
plot(mtcars$mpg, mtcars$wt, col = mtcars$gear)
legend("topright", legend = 3:5, col = 3:5, pch =1)
```

Exercise 5. Graphing

1. Attach the `mtcars` dataset, if necessary.
2. Plot horsepower (`hp`) as a function of fuel efficiency (`mpg`). Try varying the plotting symbol, symbol color, and symbol size.
3. To figure out what numbers for `pch` correspond to which plotting symbols, try this: `plot(1:25, pch=1:25)`. Do it again, but make the symbols larger (`cex = 2`) so that you can more clearly see them, and use a background color (`bg = 'red'`) because some of the symbols are filled.
4. Repeat the plot for question 2, but now use different colors for manual (am = 1) versus automatic transmissions (am = 0).
5. Try this: `plot(mpg, hp, type='n')`. The `type='n'` argument says to set up the axes, but not to plot the data. Try the following: `text(mpg, hp, cyl)`. This tactic can be useful for controlling precisely what is plotted.
6. Plot `qsec` (the time the car takes to complete 1/4 mile) as a function of `mpg`. Use `identify()` to figure out: (a) which car is the really slow one at the top middle of the plot, and (b) which are the two cars with terrible fuel efficiency on the left edge of the plot.
7. The function `identify()` takes an optional argument called `labels` which can be used to label points on a graph. Identify a few points, using the argument `labels=rownames(mtcars)`. Note that the label is placed to whatever side of the point that you click.

Statistical tests

Generally, statistical tests are performed using built-in functions

```
x <- rnorm(20, mean = 10, sd = 1)
y <- rnorm(20, mean = 11, sd = 1)
t.test(x,y)
```

Most return several pieces of information as a list

```
w <- t.test(x,y)
w$p.value    # extracts p-value from test
str(w)       # compactly shows structure of w
```

Some commonly used statistical tests

```
t.test()      wilcox.test()    # same as Mann-Whitney U
cor.test()    ks.test()        # Kolmogorov-Smirnov
var.test()    fisher.test()    # Fisher's exact test
prop.test()     # Test a specified proportion
```

Exercise 6. Additional exercises

1. Call up the help for `seq()`. Look at the different ways of specifying the arguments. Based on this, generate (a) a sequence from 0 to 20, in steps of 5, and (b) a sequence from 0 to 200 that consists of 30 numbers.

2. Create a variable `x <- rnorm(20,50,4)`. Create a new variable, `xn`, which is a normalized version of `x` (i.e., it has a mean of 0, and a standard deviation of 1) by first subtracting its mean and then dividing by its standard deviation (search the help to find the name of the function that computes standard deviations). Compare `xn` to `scale(x, center=TRUE, scale=TRUE)`. Mean-centering and normalizing are common procedures in multivariate analyses.

3. Type the following: `library(MASS)`. This allows access to a package called `MASS` (we'll cover packages later). In this package is a dataset called `Animals`. Access it using the `data()` function, and then `attach` it. This data set consists of body mass (`body`) and brain mass (`brain`) for a set of animals. How many rows and columns are in this dataset?

4. Plot `brain` as a function of `body`. Because the data vary by many orders of magnitude, log scales would help. Plot the data again, this time scaling both axes logarithmically with the argument `log="xy"`. Use the function `title()` to add an informative title to the top of the plot (check its help information, if necessary, but you can probably guess its usage).

5. Use the `identify()` function to figure out which taxa correspond to the three outliers with large bodies and relatively small brains. One way to to this: assign the results of the `identify()` function to a variable called `dumb`, and subset rows of `Animals` with `dumb`. For kicks, now try: `plot(body[-dumb], brain[-dumb], log="xy")`. This illustrates yet another way of subsetting data--can you figure out how this works?

Data Manipulation

Sorting

```
x <- c(3,1,3,10)
sort(x)    # sorts vector (does not replace x)
order(x)   # gives ranks
rank(x)    # gives ranks (averages ties)
```

Selecting

```
subset(data)   # subsets cases/variables of a dataframe
which(tf)      # gives indices for which elements are TRUE
which(mpg<15)    # indices of gas guzzlers
```

Combining

```
rbind()   # combines rows of vector/matrix
cbind()   # combines columns of vector/matrix
merge(d1, d2)   # dataframe join (like databases)
```

Data Manipulation

Tabulating

```
table(f1)          # counts per unique value of f1
table(f1, f2)       # cross tabulation
```

```
data(mtcars)        # car data again
attach(mtcars)
table(cyl)
table(cyl, am)  # tabulate number cylinders vs.
           # transmission type (am=1 is auto)
```

```
> table(cyl, am)
     am
cyl  0  1
  4  3  8
  6  4  3
  8 12  2
```

Vector & Matrix Operations

Addition element-wise, A & B same shape

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 10 \\ 10 \end{bmatrix} \qquad \mathbf{x} + \mathbf{y} = \begin{bmatrix} 11 \\ 12 \end{bmatrix}$$

```
x + y
```

Scalar Multiplication

$$\mathbf{G} = \begin{bmatrix} 1.0 & 0.8 \\ 0.8 & 1.0 \end{bmatrix} \qquad 2\mathbf{G} = \begin{bmatrix} 2.0 & 1.6 \\ 1.6 & 2.0 \end{bmatrix}$$

```
2*G
```

Matrix Multiplication

$$\mathbf{G} = \begin{bmatrix} 1.0 & 0.8 \\ 0.8 & 1.0 \end{bmatrix} \quad \beta = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \mathbf{G}\beta = \begin{bmatrix} 1.0 \\ 0.8 \end{bmatrix}$$

```
G %*% beta
NOT: G*beta
```

Vector & Matrix Operations

Transpose swap rows and columns

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \mathbf{A}^{\mathrm{T}} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

`t(A)`

Inverse $AA^{-1} = A^{-1}A = I$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \mathbf{A}^{-1} = \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix}$$

`solve(A)`

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Variance-Covariance Matrix

$$\mathbf{S} = \begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(x,y) & Var(y) \end{bmatrix}$$

`cov(X)`

Exercise 7. Data Manipulation, Vectors & Matrices

1. Load and attach the data `mtcars` again if needed. Save a subset of this dataframe to a new variable, `car.sub`. Include in this only cars with eight cylinders and the columns `mpg, hp, qsec`. Now, create the covariance matrix for this reduced dataset. Covariances are more interpretable when they are scaled as correlation coefficients, so use the `cor()` function to create a correlation matrix as well. What does this tell you about the variables?
2. For looking at correlations among variables, `pairs()` is a useful plotting function. Try `pairs(car.sub)`.
3. Create a new dataset, `x<- rnorm(30)`. Create a second variable, `y`, as `x + rnorm(30,mean=0, sd=0.1)`. Make a scatterplot of x versus y. Does the result make sense? Test, using the function `cor.test()` if the two are significantly correlated. Now, compute `cor.test(rank(x), rank(y))`. This also called the Spearman rank correlation coefficient.

If / else statements

Commands can be executed depending on some condition being TRUE, using if() and else

```
x <- 4

if (x == 4)  print("Oh")     "Oh"

if (x == 3)  print("Oh")     # Nothing happens

if (x == 3)  print("Oh") else print("Hey")    "Hey"
```

Multiple commands within a for() or if() statement need to grouped with curly braces {}.

```
if (x==4){
  print ('Wow, x is not equal to 5!')
  print ('Gee, x is not equal to 3, either!')
  }
```

Writing Functions

- There are many functions built-in to R
- Sometimes, need to do something for which no function exists
- For example: people who wrote "vegan" wanted to rarefy and compute diversity metrics
- If it is a general enough task, it can be useful to write your own function

A totally unnecessary function…

The function to make functions is called `function()`

```
times5 <- function(x)
  {
     result <- x*5
     return (result)
  }
```

function name

argument

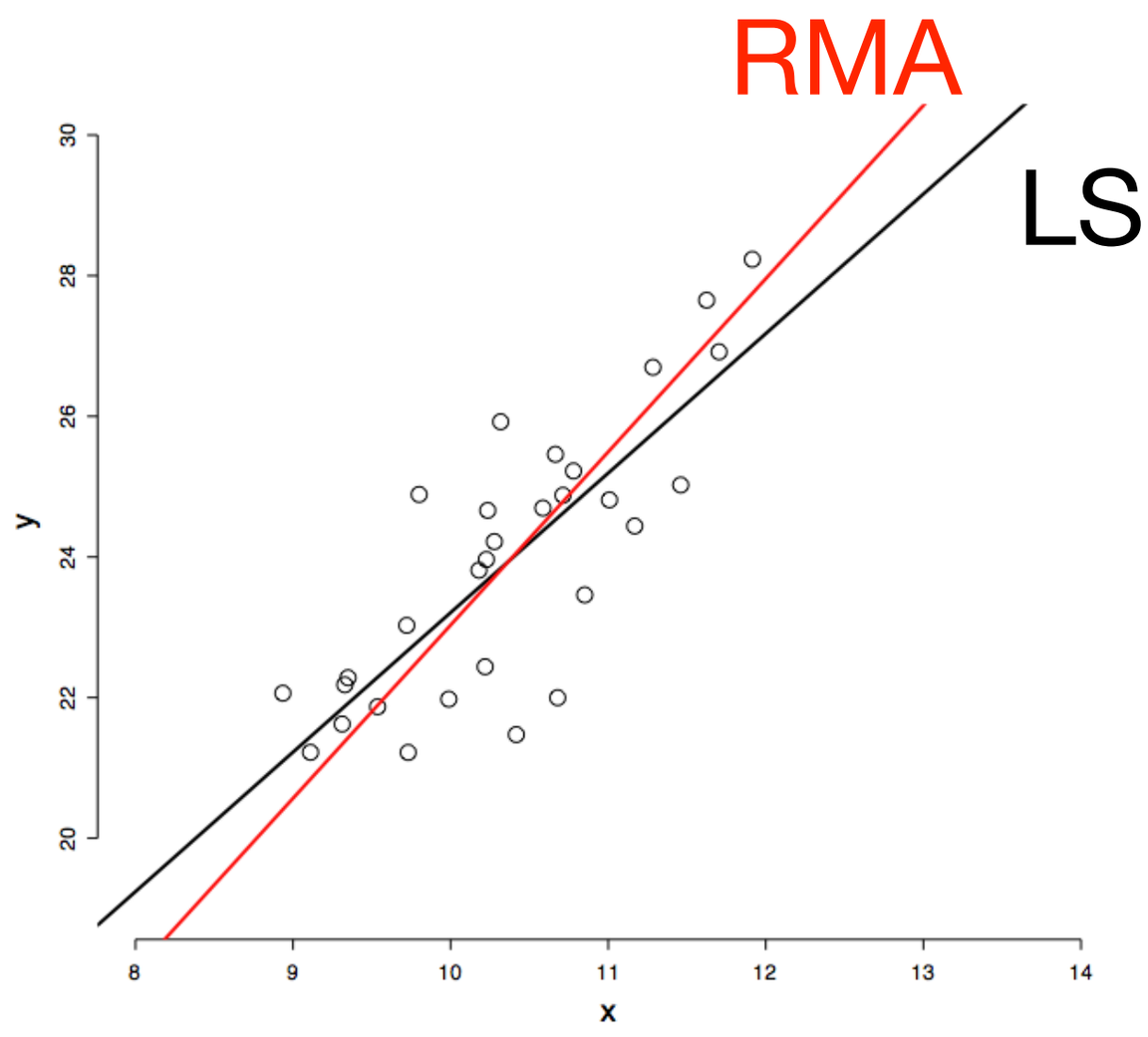
`result` gets returned as the function output

Once defined, it can be used just like built-in functions

```
times5(10)   50
```

A more useful function: RMA

- Ordinary Least-squares regression assumes all error is in y-variable
- Often, x-variable has error too
- Reduced Major Axis draws a line that allows for error in both x and y



slope $b_1 = \pm s_y / s_x$

intercept $b_0 = \bar{y} - b_1 \bar{x}$

Sourcing R scripts

- We have been entering commands, one at a time, at the interactive R prompt
- We can also write commands in a text file, and then tell R to do all of them consecutively
- The text file of commands = “script”

	Mac	Win
Open Script	File > Open Document	File > Open Script
Source Script	File > Source File	File > Source R Code

Sourcing R scripts

Sample.R

<functions>

Help search

```
1  ## example of a script
2
3  x <- rnorm(20)
4  y <- rnorm(20)
5  x
6  plot(x,y, xlab="Fake x", ylab="Fake y", pch=3, col="blue")
7
```

One difference: expressions not printed by default

Side note: syntax highlighting is really nice!

Win Users: Rstudio (Mac, too)

Script style

Rule	Do	Do Not
Variable names concise & meaningful	`ceph_len` `cephLen`	`x` `cephalon_length`

Do not overwrite existing variables/functions		`c <- 10`
		`t <- 10`
		`F <- TRUE`

Put spaces around most operators except for : ,	`x <- 1:10`
	`a < 40`
	`x <- Y[1,3]`
	`x <- Y[ ,3]`
		`x<-1:10`
		`a<40`
		`x <- Y[,3]`

Can use spaces for alignment	`total <- a + b` `mn    <- c + d`	
Add comments for why, not what	`# total has within- and between group parts`	`# add a and b`

Use commented spacer lines	`############`
	`#----------#`
	`#==========#`	

Line length <= 80 characters; indent with two spaces		*From Wickham (2015)*

Exercise 8. Conditionals, and Functions

1. Write a function to compute a Reduced Major Axis. Have it accept as arguments two vectors, x and y, and have it return a vector of two elements: the intercept and slope of the RMA. Here are a few hints. Look at the earlier slide, and figure out all the quantities you need to know to compute the intercept and slope. You'll need the `sign()` function in order to know if the slope is positive or negative (check its help entry).
2. Create a fake data set as follows: `x<- rnorm(100, 0, 1)` and `y<- x+rnorm(100,0,0.8)`. Make a scatterplot of `x` and `y`. Use your RMA function on these data, saving the result to `res.rma`. Now type `abline(res.rma)`, and look again at your plot. Note: `abline(res.rma)` will only work if you have your RMA function return a vector of c(intercept, slope) as suggested in #1.
3. Repeat #2, but with `y <- rnorm(100, 0, 1)`, which will make `y` uncorrelated with `x`. Does the resulting RMA slope surprise you?
4. The function for computing a least-squares regression is `lm()`, which is short for linear model. Use this `res.ls<- lm(y~x)`. Don't worry too much about this syntax now because we'll come back to it later on. Add the least-squares regression as a dashed, blue line: `abline(res.ls, lty=2, col='blue')`. The RMA slope should be steeper than the least squares slope.

Importing Data (preliminaries)

- Preliminary: change the working directory via menu
  - Mac: **Misc**, **Change Working Directory**
  - Win: **File**, **Change dir**
- The data should be a clean rectangular matrix
  - Must be plain text (ASCII), usually exported from Excel or a database program.
  - It is advisable to have the first row be a header with names of the variables
  - The first column may optionally be row names (labels)
  - Example: look at cope.txt (Hunt & Roy 2006, PNAS)

Importing Data

Workhorse function: `read.table()`

```
cope<- read.table(file="cope.txt", header=TRUE)
```

It's a good idea to look at it to make sure import worked OK

```
cope            # shows whole dataframe
cope[1:10,]     # look at first 10 rows of dataframe
head(cope)      # head shows top of any R object
```

Importing Data

Other variants: `read.csv()`, `read.delim()`; `read.xls()` from package `{gdata}`

(useful if there are spaces *within* some fields)

Handy function: `file.choose()`

```
# navigate to file
cope<- read.table(file=file.choose(), header=T)

attach(cope)  # we'll use it for a while
```

Dataframe `cope`

Row names

Column names

	species	age	depth	mg.temp	n	valve.length
ant-M1	anteropunctatus	19.87	1962	6.31	2	673
ant-T1	anteropunctatus	5.97	1545	3.75	3	663
ant-T2	anteropunctatus	1.09	1545	1.65	1	706
ant-E1	dinglei	37.06	1478	9.73	21	607
ant-E2	dinglei	34.11	1478	8.46	16	580
din-T3	dinglei	33.91	2318	8.37	2	623

For ostracode genus *Poseidonamicus*, gives species assignment, age (Ma), bathymetric depth (m), paleotemperature (°C), sample size, and body size (valve length, in μm) for 106 populations.

Factors

- Used to represent categorical data; by default, read.table() converts columns with characters into factors
- Factors look like strings, but are treated internally as indexes (integers) with factor names

```
species  # example of a factor
```

- Factors have levels, which are the unique values it takes

```
levels(species)  # returns levels
nlevels(species)  # returns number of levels
```

- Factor levels may be ordered (e.g., low, med, high), which is important in some analyses (see `?factor` and `?ordered`)

Type conversion

- Sometimes want to convert one kind of data to another, e.g., a factor to character:
  `as.character(species)`
- Functions: `as.xxx()`, where `xxx` = type

```
as.numeric()   as.character()
as.logical()   as.factor()
```

- Useful trick using type conversion
  - When logical converted to numeric, T⇒1; F⇒0
  - Summing a logical vector yields number of TRUE's

```
sum(species == "dinglei")   # counts number of pop's
                            # assigned to "dinglei"
```

Missing Data

Represented by a special character in *R*: `NA`

Many functions have an argument `na.rm`

- If TRUE, NA's are removed
- FALSE is usually default (function returns NA)
  - `median(x, na.rm=TRUE)`

```
read.table(file, na.strings="NA")
  na.strings="-999" # here, missing data are -999
```

Useful function: `complete.cases(cope)`

Returns logical vector, T for rows without missing data

```
cc<- complete.cases(cope) # see also na.omit()
cope.cc<- cope[cc,]    # only rows w/o NA
```

Exercise 9. Factors, types, and missing data

1. Plot valve.length versus age in *Poseidonamicus* using a trick involving type conversion to easily give a different plotting symbol to each species.
2. How many of the populations in `cope` date from before 20 million years ago? Figure this out by applying `sum()` to a logical vector.
3. Create a new variable, `x<- valve.length`. Compute the `median` of `x`. Change the first observation of x to be missing data by `x[1]<- NA`. Compute the `median` of `x` now. How can you get the function `median` to ignore missing data in computing a median?
4. Warning: factors and factor conversions can be tricky! Remember that, internally, they are integers, not strings. Take a look at `cbind(cope$species, as.character(cope$species)` to see the difference.

Packages

- R is modular, with functions and other objects separated into units called packages
- Some packages are always available: e.g., `base, stats, graphics`
- Others need to be made available using the function `library()`, or via the menus
- Many packages are installed by default, more can be downloaded and installed from within *R*.

CRAN Task Views

http://cran.r-project.org/web/views/

Google

SI Webaccess Apple Yahoo! Google Maps YouTube Wikipedia News (37) Popular Apache and ...n Mac OS X Ride-on

CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
gR	gRaphical Models in R

To automatically install these views, the ctv package needs to be installed, e.g., via

```
install.packages("ctv")
library("ctv")
```

and then the views can be installed via `install.views` or `update.views` (which first assesses which of the packages are already installed and up-to-date), e.g.,

```
install.views("Econometrics")
```

or

```
update.views("Econometrics")
```

CRAN Task Views

Exercise 10. Packages, and additional exercise

1. Figure out how to install and load packages using the menus on your operating system. Practice by installing the the gdata package and the packages from the tidyverse: `install.packages("tidyverse")`.
2. Practice some plotting using the cope data. Plot `valve.length` separately as a function of `mg.temp` and `depth`. Use the `symbols()` function to plot `valve.length` versus `mg.temp`, plotting circles proportional to depth. Use the argument `inches` to control to size of the circles (check the help function).
3. Add the RMA line (using your function) to the cope plot of valve.length versus mg.temp. Add it to the plot as a blue line. Compute the least-squares slope and add it as a dotted (lty=3), red line.

Statistical Models

- *R* has special syntax for statistical models, and a lot of built-in capabilities
- Models represent (hypothesized) relationships among variables, usually one response (*y*) and one or more predictor (*x*) variables

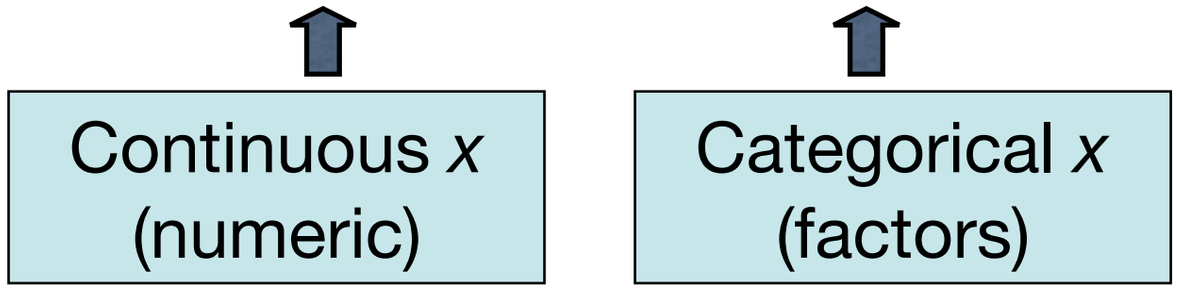
<u>example</u>

Linear regression: $y = \beta_0 + \beta_1 x$

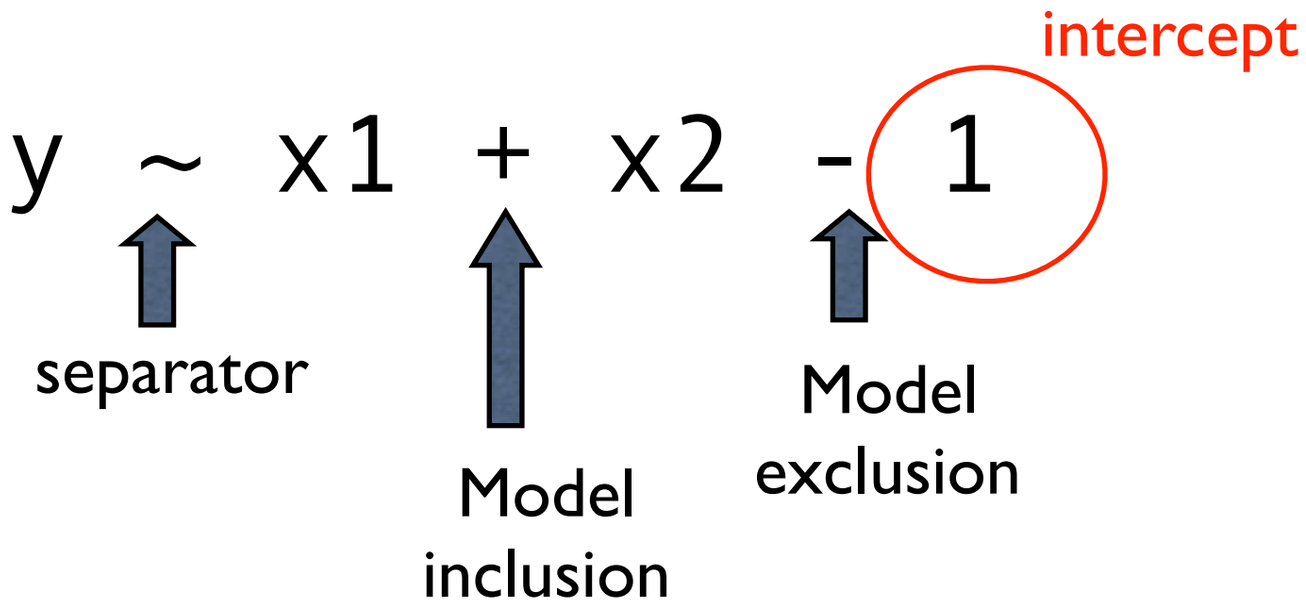
intercept slope

Linear Models

- Response variable is a linear function of predictor variable(s)
- Very common, includes regression & ANOVA



Model Notation:



The function `lm()`

- Evaluates linear models for best fitting coefficients
- Returns a list with lots of information
- Example: linear regression

```
plot(mg.temp, valve.length)
w <- lm(valve.length ~ mg.temp, data = cope)


 w      # gives only coefficients & formula
 abline(w)  # adds regression line to plot
 summary(w) # gives much more info: stats, P, etc.
 str(w)   # gives terse view of all elements of w
```

```
> summary(w)      ## summary of linear regression

Call:
lm(formula = valve.length ~ mg.temp)

Residuals:
     Min       1Q   Median       3Q      Max
-115.369  -44.431    8.367   45.486  104.234

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  764.639     10.125  75.522  < 2e-16 ***
mg.temp      -19.530      2.466  -7.918 2.77e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 57.77 on 104 degrees of freedom
Multiple R-Squared: 0.3761, Adjusted R-squared: 0.3701
F-statistic:  62.7 on 1 and 104 DF,  p-value: 2.774e-12
```

Some information-extracting functions

`resid(w)` `coef(w)` `fitted(w)` `confint()`

ANOVA

Make fake groupings

```
gg <- rep(c("a", "b"), length.out=nrow(cope))
ggf<- factor(gg)    # convert from character to factor
```

Compute linear model

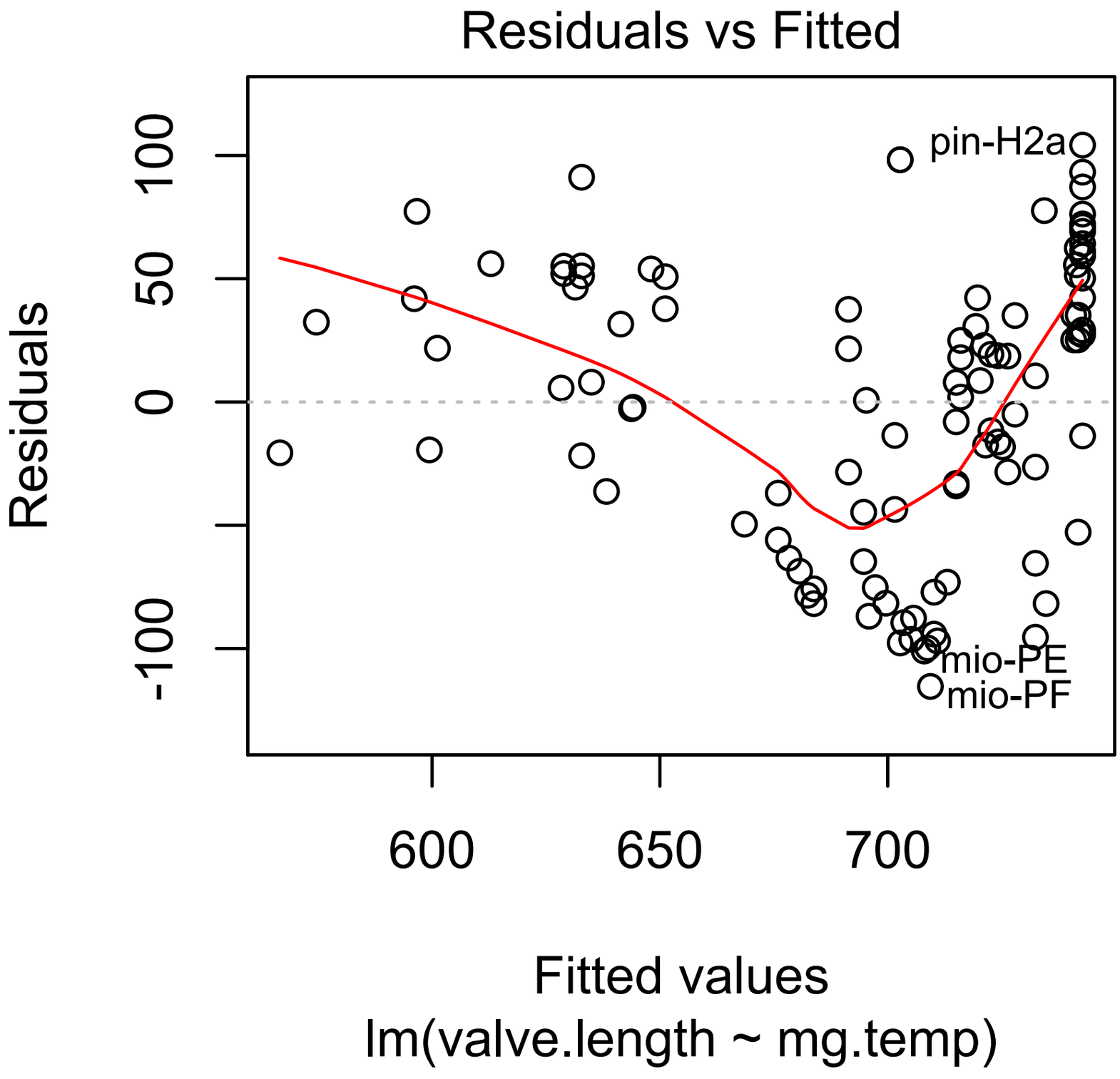
```
w.a <- lm(valve.length ~ ggf, data=cope)
summary(w.a)    # coefficients and p-values
anova(w.a)      # standard ANOVA table (SSQ, MSQ, F-test)
```

Compute linear model

```
w.sp <- lm(valve.length ~ species, data=cope)
summary(w.sp)    # coefficients and p-values
anova(w.sp)      # standard ANOVA table (SSQ, MSQ, F-test)
```

Exercise 11. Statistical models

1. Model notation is sometimes used in functions that do not directly involve statistical models. To see two examples, try the following commands: `plot(valve.length ~ mg.temp)`, and `plot(valve.length ~ species)`.
2. Look at the plot of `valve.length` as a function of `mg.temp`. Use `abline()` to add the line corresponding to the linear regression (w) to the plot. Does the linear regression seem adequate? In fact, the residuals are nonrandomly distributed. One common regression diagnostic is to plot the residuals versus the fitted values. Try this using the `fitted()` and `resid()` functions; put the fitted values on the horizontal axis. Use `abline(h=0, lty=3)` to put a dotted horizontal line at zero. From left to right, notice that the residuals are mostly positive, then mostly negative, then mostly positive.
3. Repeat the symbol plot from exercise 10.3: `symbols(mg.temp, valve.length, circles=depth, inches=0.3)`. What does this show? Add the regression line for w again. This plot seems to suggest that depth is having some effect on body size (note the size of the points above and below the line).
4. To incorporate `depth`, perform a multiple regression in which `mg.temp` and `depth` are jointly used to predict `valve.length`; save the results to a variable called `w.td`. Is `mg.temp` still significant? How about `depth`? Redo the diagnostic plot looking at fitted and residual values. Does adding the extra variable result in better-looking residuals? Note that, in terms of interpretation, both of these variables reflect temperature: `mg.temp` is a proxy for global deep-water temperature, and, at any point in time, deeper waters are colder than shallower waters.
5. FYI: try plot(w), which shows several useful diagnostic plots for regressions.

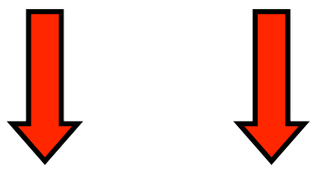

Residuals vs Fitted
Residuals
100
50
0
-100
600
650
700
pin-H2a
mio-PE
mio-PF
Fitted values
lm(valve.length ~ mg.temp)

Looping Basics

Situation
You have a set of objects (sites, species, measurements, etc.) and you want to do some computation to each one

The function that makes loops is `for()`

iterator **vector over which i iterates**



```
for (i in 1:5){
    print(i)
}
```

Sets up multiple passes, one for each element of the vector.

Here, `i` is 1 on the first pass, 2 on the second pass, and so on.

Looping Basics

Usually, you want to save information with each pass, and so you need to set up an array to store that information

```
femur <- c(10, 8, 14, 22, 32)    # 5 femora lengths
```

```
1 log.femur <- array(dim=5)    # set up new variable

          2
for (i in 1:5){
                                                    3
    log.femur[i] <- log(femur[i])
}
```

1 set up variable to save computations

2 figure out indices to loop over

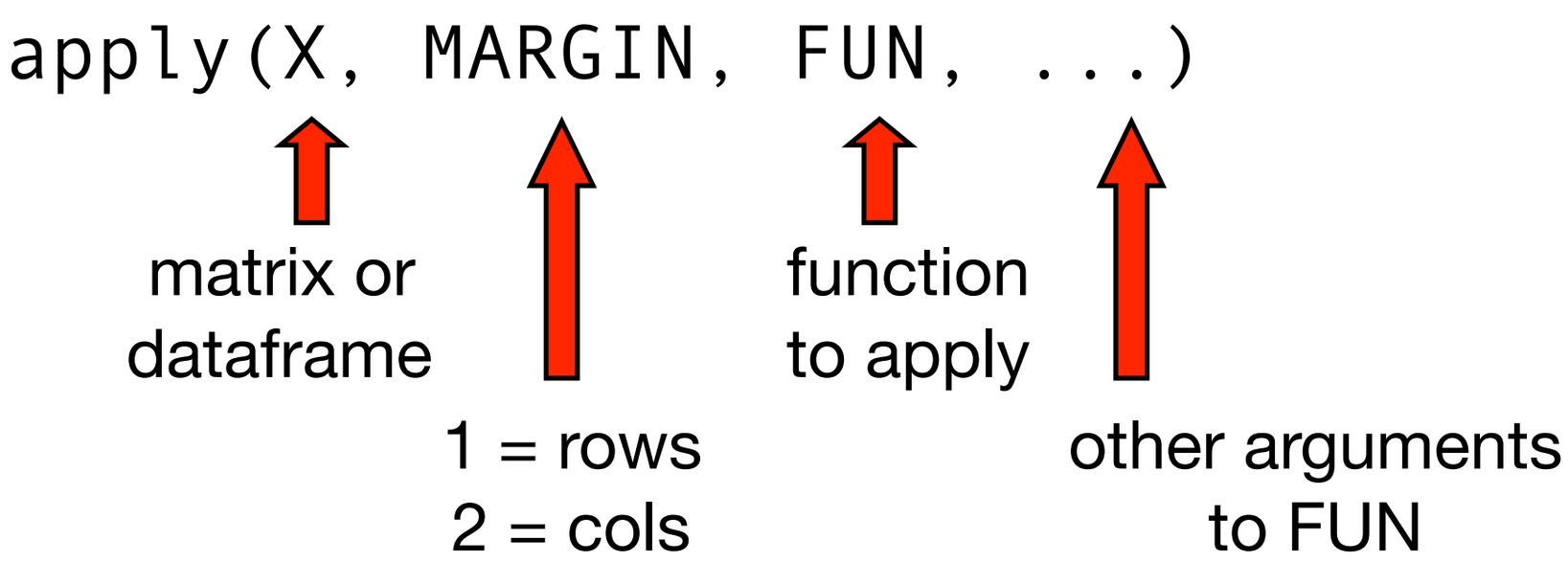
3 do calculations; save into results array

Here, looping is not necessary

```
log.femur <- log(femur)
```

Apply-family of functions

- Set of functions that allow one to perform operations over chunks/subsets of the data
- Avoids loops (can be much faster)
- Different functions for different data structures



apply(X, MARGIN, FUN)

```
X <- matrix(1:12, nrow = 3, ncol = 4)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
apply(X, MARGIN = 2, FUN = sum)
```

```
6 15 24 33
```

```
apply(X, MARGIN = 1, FUN = sum)
```

```
22 26 30
```

```
# shortcuts
colSums(X)
colMeans(X)
rowSums(X)
rowMeans(X)
```

Other versions

`lapply()` and `sapply()` operate over lists - repeatedly call `FUN` to each element of a list

```
## make a list of t-tests
tl <- list()
for(i in 1:100){
  tl[[i]] <- t.test(rnorm(100), rnorm(100))
}


## look at str(tl[[1]]) to see components
ff <- function(x) x$p.value   # extract p.value element
pvl <- lapply(tl, ff)
pvs <- sapply(tl, ff)    # same, but converts to vector

pvs <- sapply(tl, “[[“, “p.value”)    # another way to do extraction
```

Other versions

version	**notes**
`sapply(X, FUN)`	like `lapply()` but **s**implifies output to vector or matrix
`tapply(X, INDEX, FUN)`	applies FUN over subgroups defined by factor INDEX
`mapply(FUN, …)`	**m**ultivariate version - accepts more than one list or vector as arguments, drawn from …
`mclapply(X, FUN)`	parallel (**m**ulti**c**ore) version of lapply(); in package {parallel}. [not Windows, see `parLapply()` instead.]

Exercise 12. Looping

1. Use a loop to go through all the species in the Cope dataset and compute the mean valve.length for each species. Save these mean valve lengths to a vector.
2. Question #1 can be done much more simply using `tapply()`. Do so, referring to the help page as needed.
3. Go back to `mtcars`. Compute average `mpg` separately for each 4, 6 and 8 cylinder cars.
4. Apply and similar functions are sometimes -- but not always -- faster than loops. Test this with a large matrix, `M <- matrix(rnorm(1e6), nrow = 1e2, ncol = 1e4)`. There is a helper function, `system.time(expr)`, that tells you how much time it takes R to execute the R command `expr`. Use this to time how long it takes to compute column sums using apply, and then using a for loop. You'll want to write a new function to do the column sums using a loop so that it can be used as the argument to `system.time()`.

Classes in R

- R is sometimes referred to as an ‘object-oriented’ language
- Everything you create in R is an object: functions, variables, etc.
- Classes are categories of objects. Many standard ones: numbers, character strings, arrays, dataframes, lists.
- The `class()` function tells you an object’s class.

```
class(1.4)   # “numeric”
class(cope)  # “dataframe”
class(TRUE)  # “logical”
```

Classes in R

- There are also more specialized classes

```
x <- rnorm(50); y <- rnorm(50)
w <- lm(y ~ x)
class(w)  # "lm"
```

- Some functions are **generic**, meaning they call specialized versions depending on the class of the argument
- Common generics include: `plot()`, `print()`, `summary()`; specialized versions are of the form `function.class()`

```
plot.lm()
print.lm()
```

Exercise answers follow

Exercise Answers

Ex 1: (1) x<- 46; xx<- (x/2)^4 (2) y<- TRUE; y<- “Cambrian”

Ex 2: (1) XX<- matrix(1:12, nrow=3, ncol=4); y<- XX[3,] (2) y2<- 10*y (3) number of columns of XX (=4)

Ex 3: (2) dim(mtcars) (3) mean(mtcars$mpg); median(mtcars$mpg); (4) mtcars[,1:2]; mtcars[,c(1,4)]; mtcars[1:10,]; mtcars[1:10, 1:3] (5) RelPower<- mtcars$hp/mtcars$wt

Ex 4: (1) attach(mtcars); mean(mpg[wt>3]); mean(mpg[wt<3]) (2) hp[wt<3 & mpg>=25] (3) muscle<- hp>200; rn<- rownames(mtcars); rn[muscle]

Ex 5: (1) attach(mtcars) (2) plot(mpg, hp, pch=3, col=”red”) (4) pcol=rep(“red”, times=nrow(mtcars)); pcol[am == 1] <- “blue”; plot(mpg, hp, pch=21, bg=pcol, cex=2) OR: plot(mpg, hp, pch=21, bg=am+1, cex=2) (6) plot(mpg, qsec); identify(mpg, qsec) (7) identify(mpg, qsec, labels=rownames(mtcars))

Ex 6: (1) seq(0,20,5); seq(0,200, length.out=30) (2) xn<- (x-mean(x))/sd(x) (3) 5% of the time (3) dim(Animals) (4) attach(Animals); plot(body, brain, log=”xy”); title(“Brain size allometry”) (5) dumb<- identify(body, brain)

Ex 7: (1) car.sub<- mtcars[mtcars$cyl==8, c(‘mpg’, ‘hp’, ‘qsec’)]; SS<- cov(car.sub); RR<- cor(car.sub)

Ex 8: See next page.

Ex 9: (1) plot(age, valve.length, pch=as.numeric(species)); legend(x="topright", pch=1:19, levels(species), cex=0.6) (2) sum(cope$age>20) (3) median(x, na.rm=TRUE)

Ex 10: (2) symbols(cope$mg.temp, cope$valve.length, circles=cope$depth, inches=0.3)

Ex 11: (2) plot(fitted(w), resid(w)) (4) w.td<- lm(valve.length ~ mg.temp + depth)

Ex 12: (1) See next page. (2) tapply(valve.length, INDEX = species, FUN = mean) (3) tapply(mtcars$mpg, INDEX=mtcars$cyl, FUN=sum) (4) See next page.

```r
 # Reduced major axis (Ex. 8-1)

RMA<- function(x,y)
{
 # compute needed summary statistics
 mx <- mean(x)
 my <- mean(y)
 sx <- sd(x)
 sy <- sd(y)
 rxy <- cor(x,y)

 # compute slope and intercept
 b1 <- sy / sx * sign(rxy)
 b0 <- my - b1 * mx

 # combine slope and intercept into a vector
 result<- c(b0,b1)
 return(result)     # abline(result) will work!
}
```

```r
 ## speed of loops versus apply (Ex. 12-4)
 M <- matrix(rnorm(1e6), nrow=1e2, ncol=1e4)

 system.time(apply(M, 2, sum))  # 0.033 sec on my machine
 system.time(colSums(M))        # much faster! 0.001 sec

 # now, using loops
 loopColSum <- function(M){
    nc <- ncol(M)
    cm <- array(dim=nc)   # vector to store colsums
    for(i in 1:nc)  cm[i]<- sum(M[,i])  # loop thru columns, get each sum

    return(cm)
 }

system.time(loopColSum(M)) # faster than apply, 0.02 sec on my machine
```

```r
# Looping example (Ex. 12-1)
lsp<- levels(species)    # all species names
nsp<- nlevels(species)   # number of species
mn.len<- array(dim=nsp)  # vector to store mean lengths by species
names(mn.len)<- lsp      # give the vector elements names for the species
for(i in 1:nsp){
   sub <- species == lsp[i]
   mn.len[i] <- mean(valve.length[sub])
}
```