Computational Skills for Biostatistics I: Lecture 1

Amy Willis, Biostatistics, UW

September 28, 2017

Welcome!

- Biost 561 is an add-on to Biost 514
- 514 covers basic R commands, 561 covers more advanced R and other programming skills
- As a MS/PhD student in Biostats, you will do some serious programming!
- Good programming practices will help you with research, collaborating, your job search, and your long-term career... whichever path you choose!

Structure and expectations

- Weekly lectures
- Weekly homeworks
- Weekly office hours: HSB F-657. Tuesday 3-4?
- Office hours by appointment

Public folder of course materials at
`https://github.com/adw96/biostat561`

Topics

Subject to change

- 9/28 Lecture 1: Advanced classes, methods, objects, debugging
- 10/5 Lecture 2: Efficient loops, functions
- 10/12 Lecture 3: Pipes
- 10/19 Lecture 4: ggplot
- 10/26 Lecture 5: latex *
- 11/2 Lecture 6: Markdown *
- 11/9 Lecture 7: unix, shell, UW cluster computing *
- 11/16 Lecture 8: version control, git, github, packages
- 11/23 no class; Thanksgiving
- 11/30 Lecture 9: Calling C/C++ in R *
- 12/7 Lecture 10: Python *

* indicates guest lecture by one of your classmates. A great opportunity to learn the latest and greatest!

Resources

- Available via github
  `https://github.com/adw96/biostat561`
  - Syllabus
  - Slides (& source code)
  - Examples
  - Homeworks
- Available via github classroom
  - Homework submission
- Available via email
  - Announcements

Expectations

What you should expect of me

- I will make your learning a priority
- I will give you timely feedback on your homeworks
- I will treat you as adults, I will treat you with respect
- I will talk slowly (tell me if I'm speaking too fast!)
- I will try to make class engaging and fun!

Expectations

What I will expect of you

- You make attending class a priority
- You submit your best work for homework: your own work, on time
- You engage in classroom discussion and quizzes!
- You learn from the class; you learn to teach yourself programming skills
- You treat me, guest lecturers, and each other with respect

Assessment

The only assessment in this course is homework.

- 10 or fewer homeworks
- You must submit a good attempt at every homework to receive credit for this course

I won't record attendance, but if you consistently do not show up you will not receive credit.

About me

AmyW
adw96

statistician, biodiversity enthusiast, assistant professor

Seattle, WA
http://faculty.washington.edu...

Organizations

Overview | Repositories 11 | Stars 8 | Followers 13 | Following 21

Pinned repositories — Customize your pinned repositories

breakaway
Species richness with high diversity
R ★5 1

ConfidenceSets
Code accompanying my confidence sets research
R ★1

stamps
Course materials for STAMPS @ MBL
R ★4 5

CatchAll
Species richness estimation in R
R

TreeUncertainty
Incorporating uncertainty in tree space
R ★1

biostat561
Course materials for BIOST561
HTML

82 contributions in the last year — Contribution settings

Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep
Mon
Wed
Fri

Learn how we count contributions. Less More

About you

Everyone is here with different backgrounds in programming and computing. Let's get statistical!

[Analyse survey here]

Class

There are many different *classes* of objects in R

```
x <- c(1, 2, 5)
y <- c("a", "b")
z <- as.factor(y)
c(class(x), class(y), class(z), class(c))
```

```
## [1] "numeric"   "character" "factor"    "function"
```

Others include `logical` (`TRUE`, `FALSE`), complex numbers...

Modes

R has different modes. The mode tells the way a variable is stored.

```
mode(x)
```

```
## [1] "numeric"
```

```
mode(y)
```

```
## [1] "character"
```

```
mode(z) # factors are stored as numerics
```

```
## [1] "numeric"
```

```
mode(c)
```

```
## [1] "function"
```

Modes

`is.[class]` asks about the class. Normally you will be interested in the class, not the mode.

```
is.numeric(x)
```

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

```
is.factor(z)
```

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

```
is.numeric(z) # we asked about class, not the mode
```

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

Data structures

R can store data in various *objects*

- vector: one-dimensional, all data points have same mode
- matrix: two-dimensional, all data points have same mode
- data frame: two-dimensional, all data points in same column have same mode
- list: one-dimensional, data points can be of any type

Matrices

Matrices vs data frames: all elements have same mode in matrices

```
cbind(c(1,2), c("a", "b"))
```

```
##      [,1] [,2]
## [1,] "1"  "a"
## [2,] "2"  "b"
```

Matrices

Be very careful with matrix operations!

```
aa <- matrix(c(1, 2, 3, 5), nrow = 2, byrow = T)
bb <- c(0.5, 2)
aa
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    5
```

```
bb
```

```
## [1] 0.5 2.0
```

Matrices

```
aa %*% bb # matrix multiplication
```

```
##      [,1]
## [1,]  4.5
## [2,] 11.5
```

```
aa * bb # careful! pointwise
```

```
##      [,1] [,2]
## [1,]  0.5    1
## [2,]  6.0   10
```

Data frames

```
data.frame(c(1,2), c("a", "b"))
```

```
##   c.1..2. c..a....b..
## 1       1           a
## 2       2           b
```

```
dd <- data.frame("ID"=c(1,2), "name"=c("a", "b")) # better
dd
```

```
##   ID name
## 1  1    a
## 2  2    b
```

```
str(dd) # structure: compact info about frame & variables
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ ID  : num  1 2
##  $ name: Factor w/ 2 levels "a","b": 1 2
```

vectorisation

vectorisation: doing many calculations with a single command

```
x <- c(0.5, 2, 3, 6)
x^2
```

```
## [1]  0.25  4.00  9.00 36.00
```

```
y <- c(3, 1, 2, 1)
x^y
```

```
## [1] 0.125 2.000 9.000 6.000
```

vectorisation

The slow way: with loops. Avoid where possible!

```
z <- rep(NA, 4)
for (i in 1:4) {
  z[i] <- x[i]^y[i]
}
z
```

Code is easier to read, and usually easier to debug, when vectorised

recycling

In many tasks, R recycles elements of one input until it has enough to match the other

```
aa
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    5
```

```
bb
```

```
## [1] 0.5 2.0
```

```
aa + bb # vectors are treated as columns!
```

```
##      [,1] [,2]
## [1,]  1.5  2.5
## [2,]  5.0  7.0
```

```
cc <- c(1, 2, 3, 4)
aa + cc # the silent killer
```

Speed comparison

Vectorisation can cause major speed-ups, because task is optimised and precompiled in C/Fortran, not interpreted R.

```
dd <- matrix(rnorm(1e6), nrow = 1000)
cor(dd)

ee <- matrix(NA, 1000, 1000)
for (i in 1:1000) {
  for (j in 1:1000) {
    ee[i, j] <- cor(ee[, i], ee[, j])
  }
}
```

Speed up factor of vectorisation: 36!

Lists

Lists store information of many different types. Names are optional, but recommended!

```
amy <- list(office.num = 657, pets = TRUE,
            pets.names = c("Princess Jaws", "Friendly", "Mohawk",
                           "Canada", "USA", "Regina George"),
            is.cat = c(TRUE, rep(FALSE, 5)))
amy
```

```
## $office.num
## [1] 657
##
## $pets
## [1] TRUE
##
## $pets.names
## [1] "Princess Jaws" "Friendly"      "Mohawk"        "Canada"
## [5] "USA"           "Regina George"
##
## $is.cat
## [1]  TRUE FALSE FALSE FALSE FALSE FALSE
```

Lists

Double square brackets pull out individual elements. Single square brackets pull out subsets of the list. I recommend using names wherever possible!

```
amy[[3]] # subset third element
```

```
## [1] "Princess Jaws" "Friendly"      "Mohawk"        "Canada"
## [5] "USA"           "Regina George"
```

```
amy[3] # third element -- a list!
```

```
## $pets.names
## [1] "Princess Jaws" "Friendly"      "Mohawk"        "Canada"
## [5] "USA"           "Regina George"
```

Lists

```
amy[2:3] # second and third elements -- a list!
```

```
## $pets
## [1] TRUE
##
## $pets.names
## [1] "Princess Jaws" "Friendly"      "Mohawk"        "Canada"
## [5] "USA"           "Regina George"
```

```
amy$office # can also refer by name
```

```
## [1] 657
```

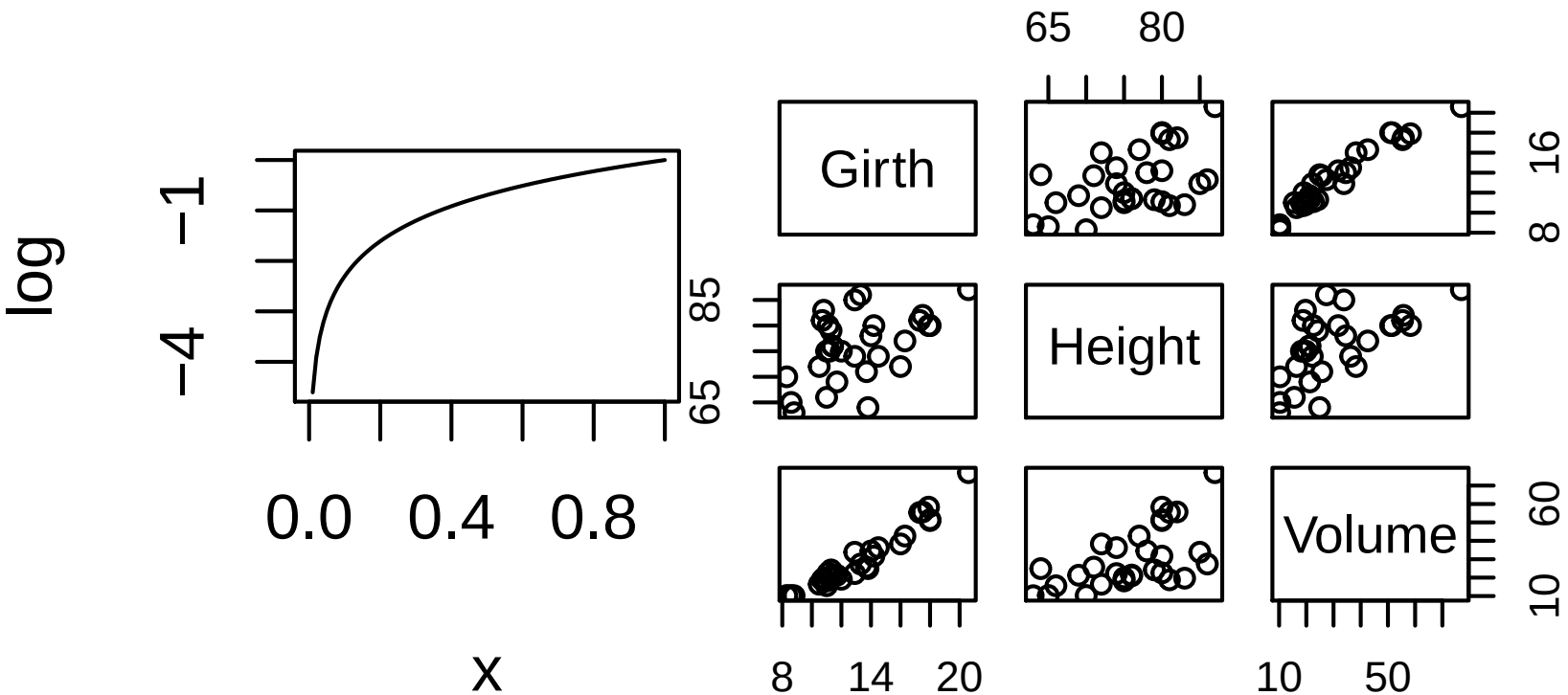
Generic functions

The same function can apply to objects of different classes. How does R know what to do?

```
c(class(log), class(trees))
```

```
## [1] "function"   "data.frame"
```

```
layout(t(1:2), widths = c(3,1)); plot(log); plot(trees)
```



Generic functions

`plot` is a *generic* function. Generic functions don't do anything themselves – they call *methods*, which are tailored to the class.

```
plot
```

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x7f94433b7b50>
## <environment: namespace:graphics>
```

```
methods("plot") # lists all types R knows how to plot
```

```
##  [1] plot.acf*              plot.data.frame*    plot.decomp
##  [4] plot.default           plot.dendrogram*    plot.densit
##  [7] plot.ecdf              plot.factor*        plot.formul
## [10] plot.function          plot.hclust*        plot.histog
## [13] plot.HoltWinters*      plot.isoreg*        plot.lm*
## [16] plot.medpolish*        plot.mlm*           plot.ppr*
```

Generic functions

To find the functions that apply to a class

```
methods(class = "lm")
```

```
##  [1] add1             alias            anova            case.n
##  [5] coerce           confint          cooks.distance   devian
##  [9] dfbeta           dfbetas          drop1            dummy.
## [13] effects          extractAIC       family           formul
## [17] hatvalues        influence        initialize       kappa
## [21] labels           logLik           model.frame      model.
## [25] nobs             plot             predict          print
## [29] proj             qr               residuals        rstand
## [33] rstudent         show             simulate         slotsF
## [37] summary          variable.names   vcov
## see '?methods' for accessing help and source code
```

Generic functions

To see the code for a generic function, type `[function].[class]`

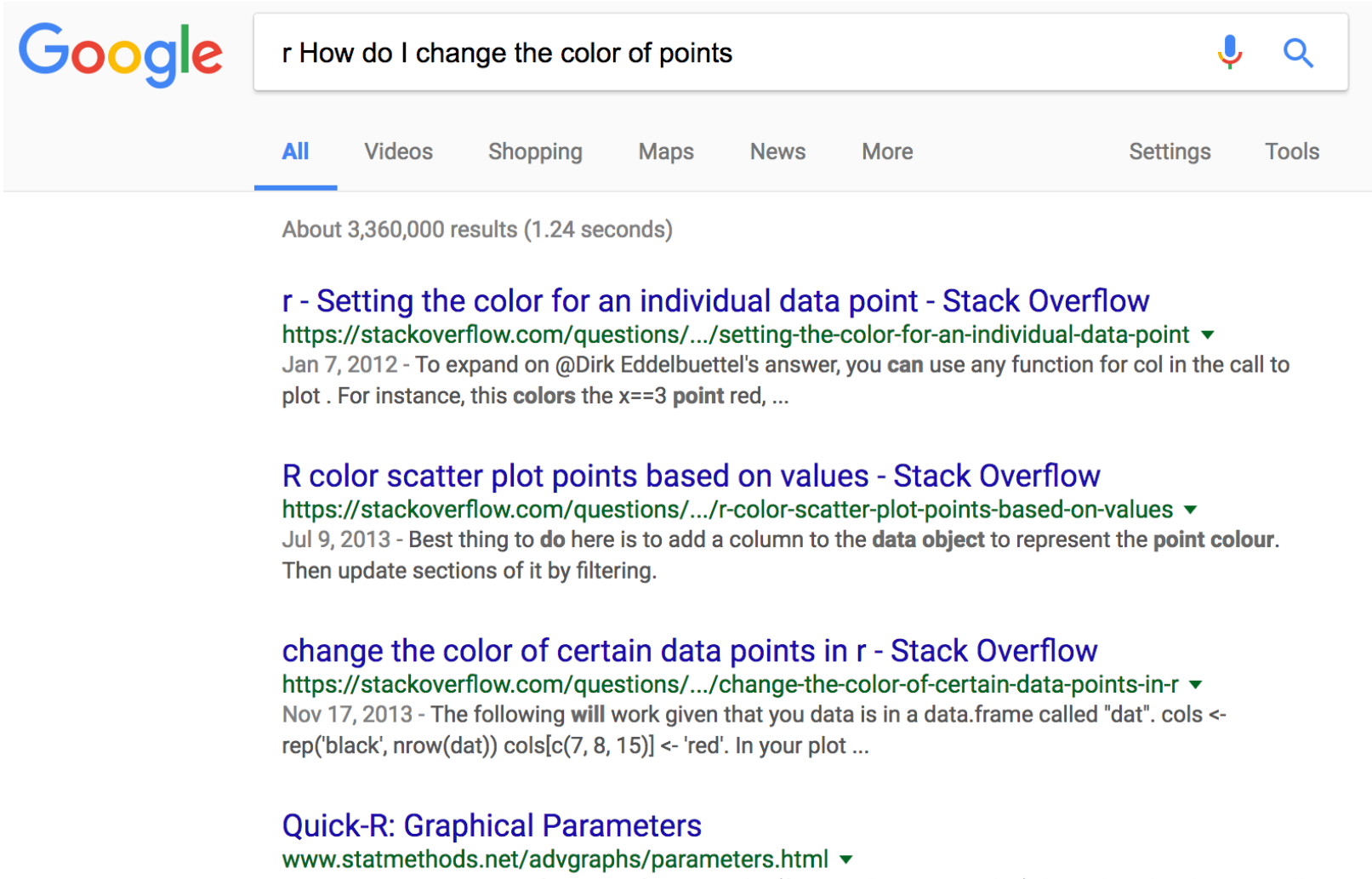
```
dimnames.data.frame
```

```
## function (x)
## list(row.names(x), names(x))
## <bytecode: 0x7f94460a20e8>
## <environment: namespace:base>
```

Help

There are many ways to get help with using functions or debugging code

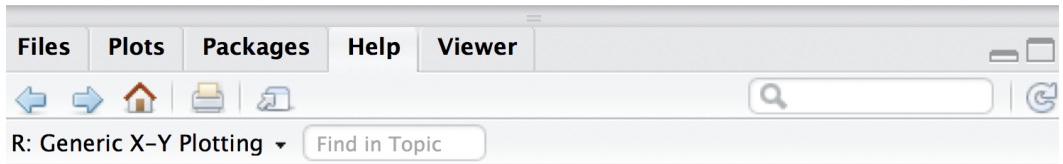
1. The internet



Help

2. `?fn` shows the documentation for `fn`... if it exists!

```
> ?plot
>
```



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

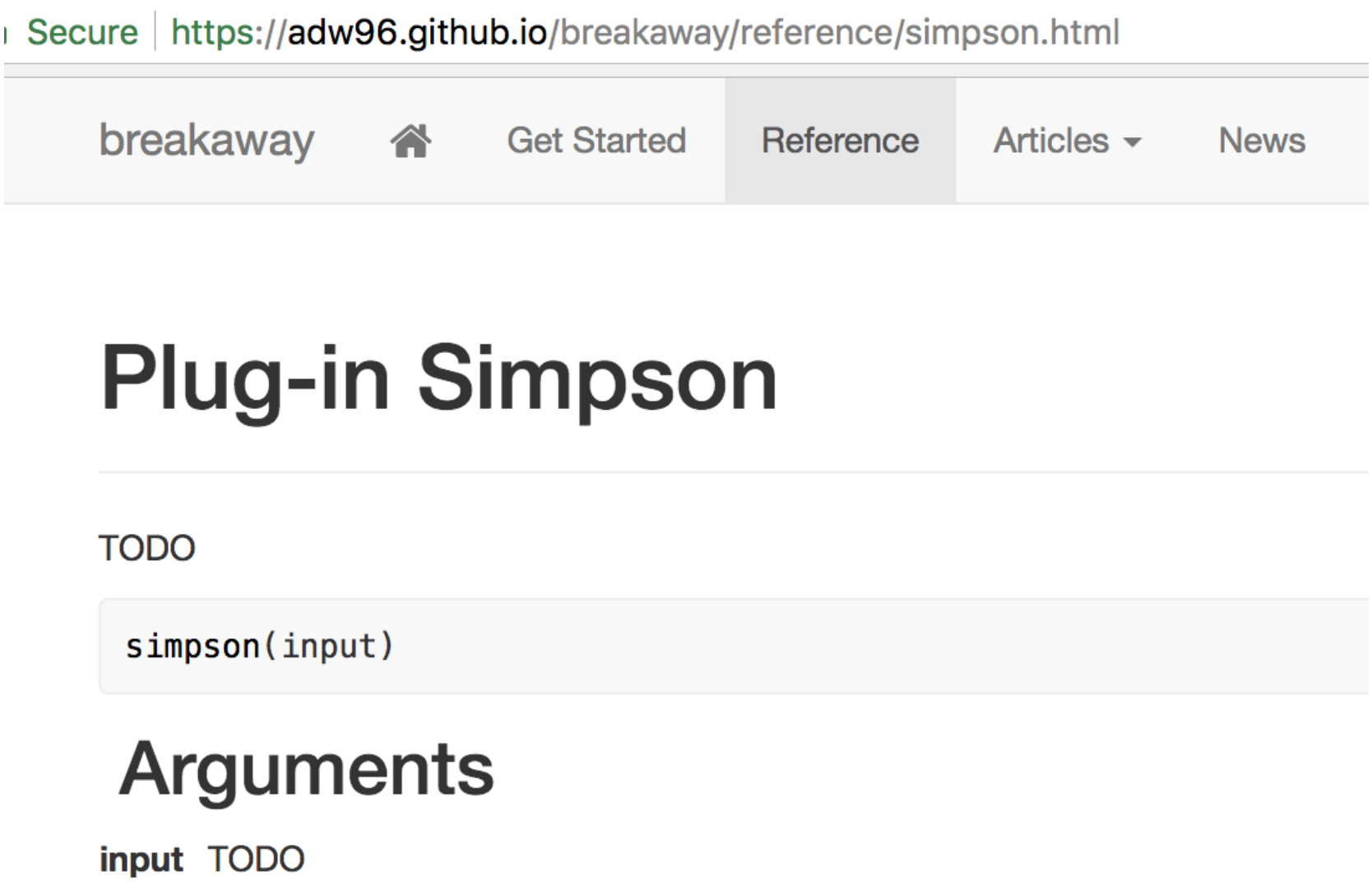
```
plot(x, y, ...)
```

Arguments

x	the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a plot method* can be provided.
y	the y coordinates of points in the plot, *optional* if x is an appropriate structure.
...	Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

Help

2. . . . if it exists!



Help

3. `help.search("topic")` searches help pages for "topic"

```
> help.search("points")
>
```

Search Results

Code demonstrations:

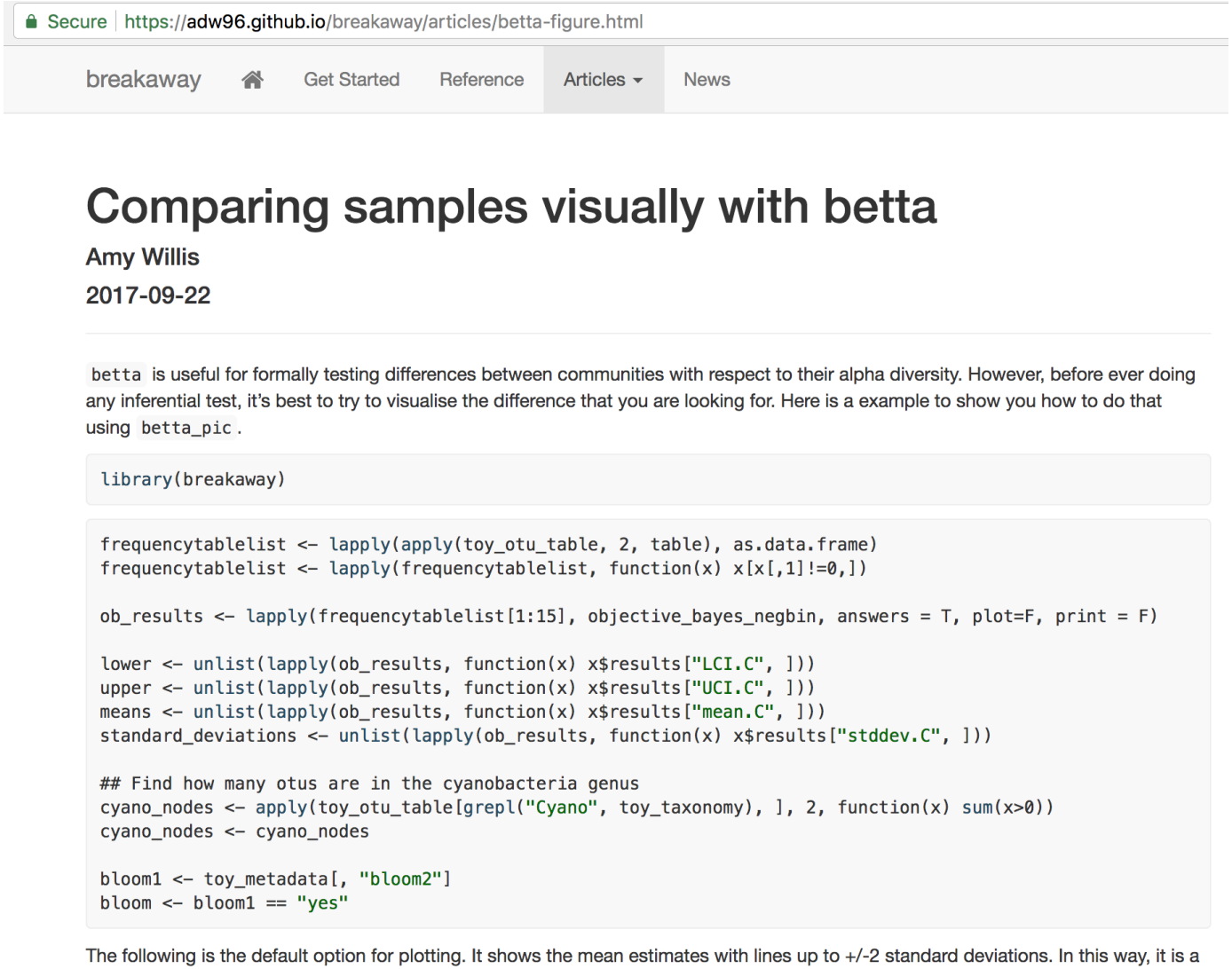
tcltk::tkcanvas	Creates a canvas widget showing a 2-D plot with data points that can be dragged with the mouse.	(Run demo in console)

Help pages:

ade4::ichtyo	Point sampling of fish community
ade4::procuste	Simple Procruste Rotation between two sets of points
ade4::s.class	Plot of factorial maps with representation of point classes
ade4::triangle.class	Triangular Representation and Groups of points
ade4::triangle.plot	Triangular Plotting
base::pretty	Pretty Breakpoints
base::utf8ToInt	Convert Integer Vectors to or from UTF-8-encoded

Examples

The documentation pages often show examples (`example(plot)`) and have demos (`demo(plotmath)`). `vignette()` opens longer worked examples that are great for playing with new packages.



Debugging

1. Stare at it until you identify the problem a.k.a. psychic debugging
2. Breakdown the components until you find the problem (bisection method converges linearly!)
3. `traceback()` – covered later in the course

Homework 1 and next week

- Slides:
  - `https://github.com/adw96/biostat561/lecture1/lecture1.pdf`
- Homework 1 is due next Thursday at 2 p.m.
  - `https://github.com/adw96/biostat561/lecture1/homework1.pdf`
- Complete Question 0 by next Tuesday so you can attend office hours if you have trouble
- Submission via github classroom (instructions included):
  - `https://classroom.github.com/classrooms/32249780-biost-561`