Computational Skills for Biostatistics I: Lecture 1

Amy Willis, Biostatistics, UW

28 March, 2019

Why bother?

- As a statistician working in any capacity, you will need to know some programming
- 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

Welcome!

Biost 561 covers modern/advanced R and other programming skills

- It is designed for graduate students in Biostatistics, and is tailored to their statistics and programming background
- If you don't know a lot of R already, you will learn it today and in Homework 1

Structure and expectations

- Weekly lectures
- Weekly homeworks
  - Posted within 24 hours of lecture
  - Due at the beginning of lecture via GitHub classroom
- Weekly office hours: HSB F-657, Mondays 2:30-3:30

Resources

- Available via GitHub `github.com/adw96/biostat561`
  - Syllabus
  - Slides
  - Homeworks (approximately 10)
  - Policies: inclusivity, accessibility, academic integrity
- 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!
- I will teach you the way that I program, and (contemporary) alternatives

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 treat me, guest lecturers, and each other with respect
- You engage in classroom discussion
- You learn from the class; you learn to teach yourself

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

Block or report user

statistician, biodiversity enthusiast, assistant professor

University of Washington
Seattle, WA
statisticaldiversitylab.com

Overview | Repositories 24 | Projects 0 | Stars 24 | Followers 87 | Following 38

Pinned

breakaway
Species richness with high diversity
R ★18 ⑂8

CatchAll
Species richness estimation in R
R ★1 ⑂1

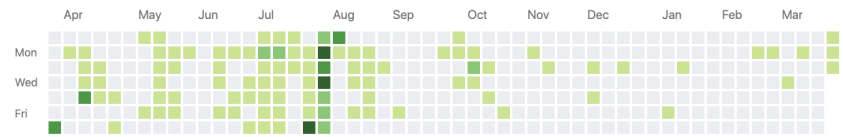
TreeUncertainty
Incorporating uncertainty in tree space
R ★4

biostat561
Course materials for BIOST561
★97 ⑂24

DivNet
diversity estimation under ecological networks
R ★13 ⑂4

stamps2018
Amy's teaching materials for STAMPS @ MBL in 2018
R ★5 ⑂2

358 contributions in the last year



Learn how we count contributions.
Less More

About you

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

?? responses to class survey

- ?? Mac-Windows users
- ??% have used R
- ??% use the apply() family
- ??% pipe, write packages
- ??% use git for **version control**

Is data missing-not-at-random?

Today's class

1. Version control with git
2. Intro to base `R` (types, methods)
3. RStudio & RStudio projects
4. Writing loops and functions

Version control

What problems can you see with the following approach to version control?

- papersims-v1.R
- papersims-v2.R
- papersims-thea-comments.R
- papersims-hellfire.R
- papersims-v5.R
- papersims-final.R

Version control

1. How many versions until this becomes intractable?
2. *Date Modified* sorting does not always help
3. Tracking changes is very difficult; reversion is even harder
4. Exponential file number growth with multiple collaborators!
5. The dreaded computer crash

Dropbox can help with some of these issues, but generally not (3) or (4)!

Git

- git is an open source version control system (VCS):
  - Track changes to code and documents. *What* changes by *who* and *when*?
  - Share code and collaborate
- GitHub is a website that uses git's VCS:
  - Collaborate with others effectively
  - Distribute code
  - Solicit improvements (*pull* requests)
  - Track issues & feature requests
  - (Build your coding portfolio!)

Git with R

git & GitHub are popular with R developers

- Integration with RStudio
- Easy distribution of packages
  - Circumvents CRAN moderators; for better or worse
- Always get the latest features (`devtools`; `install_github`)

Git with life

- Collaborations: syncing across multiple computers
- Portability: including for you
- TODOs as Issues: including your own
- Teams: `fork`, `pull` request
  - Like a package but hate the default figures? `fork` it, edit it, use your own version!
- Visibility: social/professional network

Getting started with git

- Download/update: `git-scm.com/downloads`
- Intro: `guides.github.com/activities/hello-world`
- Do this with Homework 1 and a blank pdf, not hello-world
- Questions? Error messages? The internet is a great resource!
- *git is a great habit to get into early! Start now!*

Homework 1 Question 0 will get you started using git, GitHub and GitHub Classroom.

Getting started with git

The workflow that I want you to follow for downloading the latest lecture notes is `git pull` (see Question 0.5 of Homework 1)

The standard workflow for uploading a new file or updating an old one (e.g., for your homework submission):

```
git pull
git add hw1-response.pdf
git commit -a -m 'question 2 part b response'
git push
```

You must have a git repository set up already to do this (e.g. with `git init` or `git clone ...`)

R

A fancy calculator

```
sqrt(50)
```

```
## [1] 7.071068
```

```
log(2.7)
```

```
## [1] 0.9932518
```

```
log(2.7, base=10)
```

```
## [1] 0.4313638
```

```
var(c(1, 2, 5))
```

```
## [1] 4.333333
```

Object classes

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...

Object classes

`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)
```

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

Object classes

You can define new classes in multiple ways (**S3** is the most common).

```
class(x) <- c("my_new_class", class(x))
x
```

```
## [1] 1 2 5
## attr(,"class")
## [1] "my_new_class" "numeric"
```

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

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

```
is.my_new_class(x)
```

```
## Error in is.my_new_class(x): could not find function "is
```

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
- array: n-dimensional, all data points have same mode
- data frame: two-dimensional, all data points in same column have same mode
- list: one-dimensional, elements 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

```
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

Be very careful with matrix operations!

```
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

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

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

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

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

```r
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
```

Vectorization

Vectorization: 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
```

`c()` is for "combine"

Vectorization

The slow way: with loops. Avoid where possible!

```
# initialise empty vector:
z <- vector(mode="numeric", length=4)
z
```

```
## [1] 0 0 0 0
```

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

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

Modes versus classes

R has different “mode”s as well as different “class”es

- `numeric`, `character`, `function`, `list`
- Mode concerns storage; you will deal with class moss often than mode

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

Vectorization 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 vectorization: 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.

```
amy$pets.names
```

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

```
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"
```

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

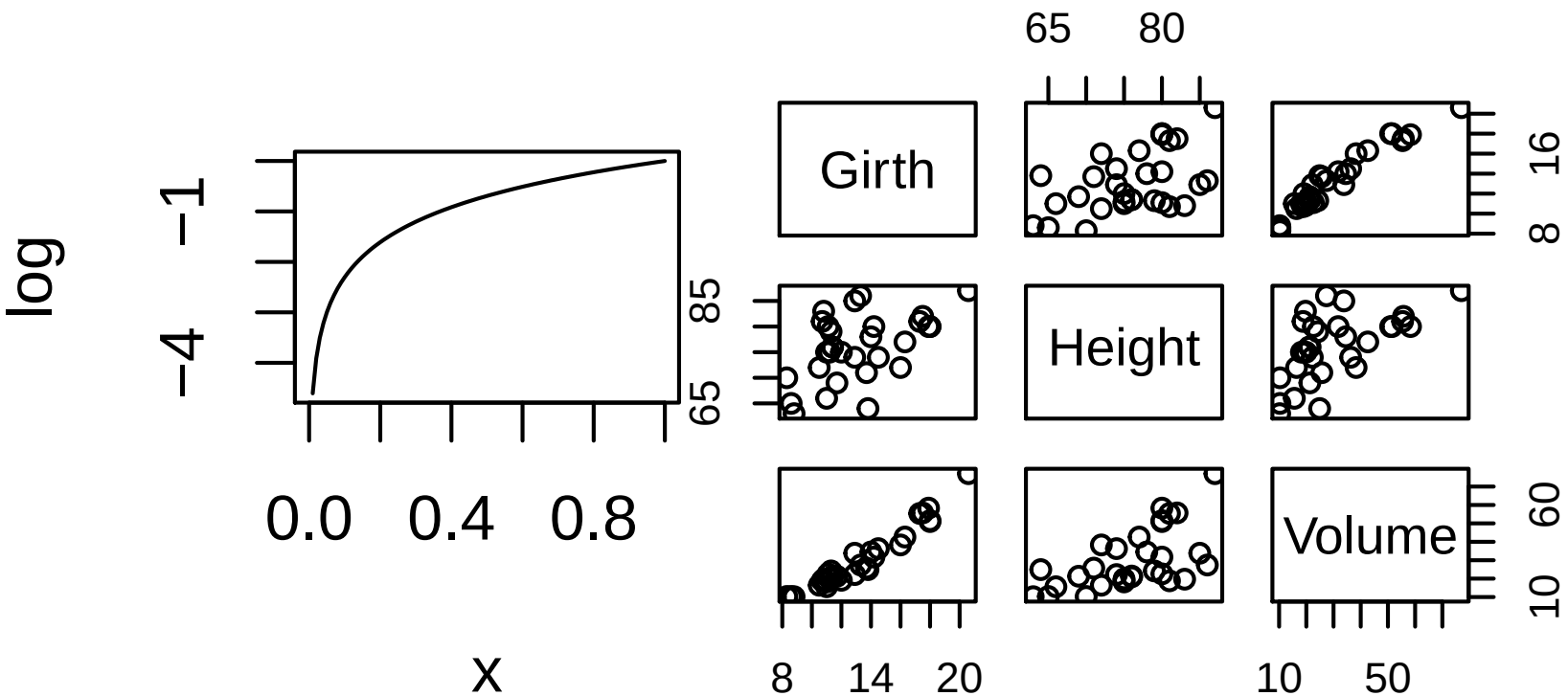
Generic functions

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: 0x7fcdff8705b8>
## <environment: namespace:graphics>
```

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

```
##  [1] plot.acf*           plot.data.frame*    plot.decomposed.ts*
##  [4] plot.default        plot.dendrogram*    plot.density*
##  [7] plot.ecdf           plot.factor*        plot.formula*
## [10] plot.function       plot.hclust*        plot.histogram*
## [13] plot.HoltWinters*   plot.isoreg*        plot.lm*
## [16] plot.medpolish*     plot.mlm*           plot.ppr*
## [19] plot.prcomp*        plot.princomp*      plot.profile.nls*
## [22] plot.raster*        plot.spec*          plot.stepfun
## [25] plot.stl*           plot.table*         plot.ts
## [28] plot.tskernel*      plot.TukeyHSD*
```

Generic functions

To find the functions that apply to a class

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

```
##  [1] add1             alias            anova            case.names
##  [5] coerce           confint          cooks.distance   deviance
##  [9] dfbeta           dfbetas          drop1            dummy.coef
## [13] effects          extractAIC       family           formula
## [17] hatvalues        influence        initialize       kappa
## [21] labels           logLik           model.frame      model.matrix
## [25] nobs             plot             predict          print
## [29] proj             qr               residuals        rstandard
## [33] rstudent         show             simulate         slotsFromS3
## [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: 0x7fce00413120>
## <environment: namespace:base>
```

Naming conventions

Because of the `[function].[class]` syntax, naming variables with dots in them is discouraged.

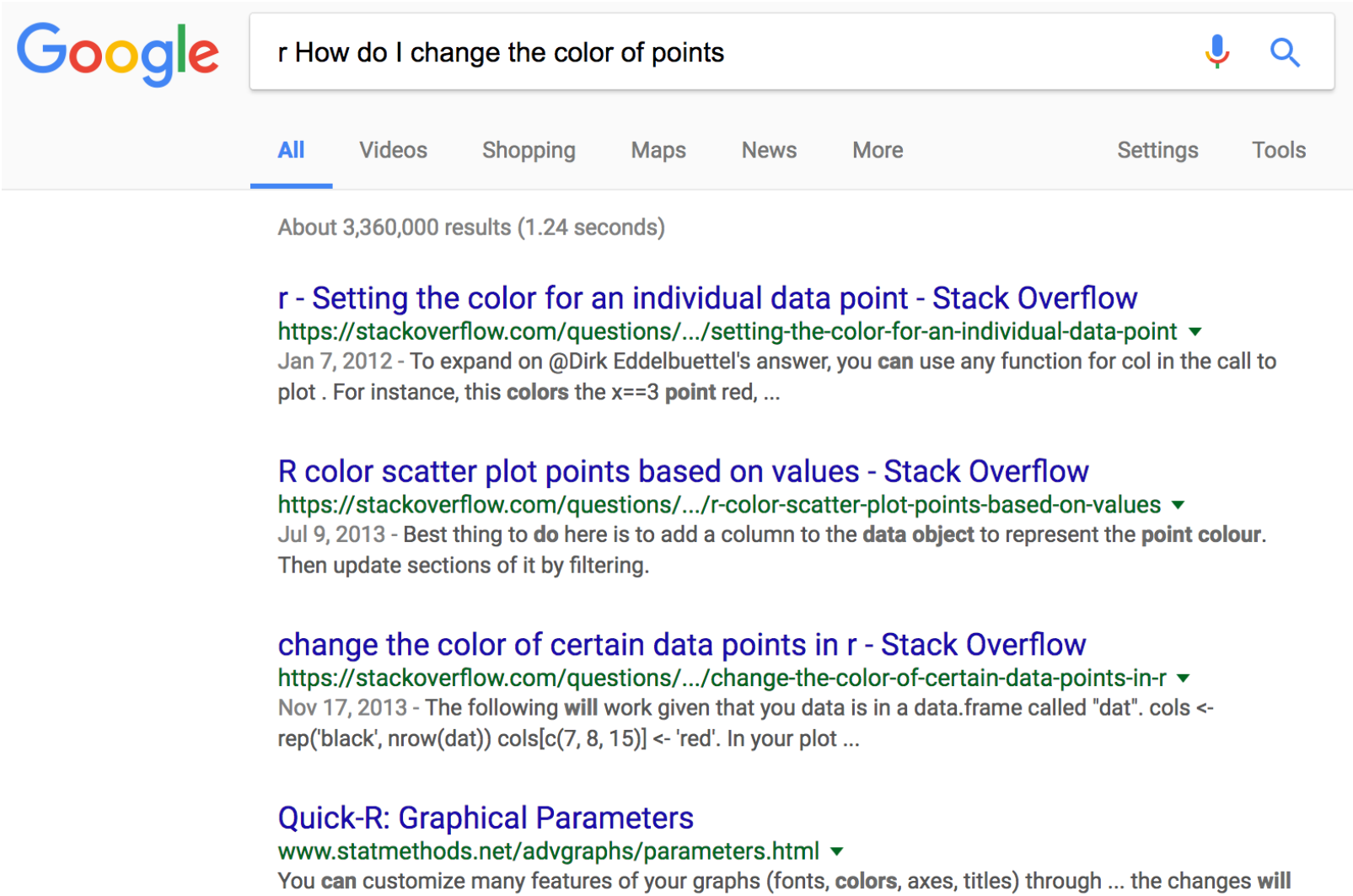
```
my.favourite.number <- exp(1)
my_favourite_number <- exp(1) # better
```

- Assignment using = is also discouraged: use <- instead. (Why?)
- `google.github.io/styleguide/Rguide.xml` is worth reading

Help

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

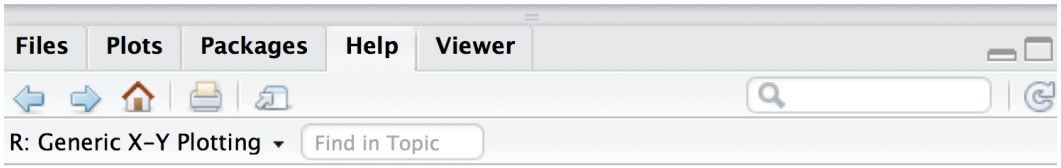
1. The internet



Help

2. `?fn` shows the documentation for `fn`...

```
> ?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 `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

Usage

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

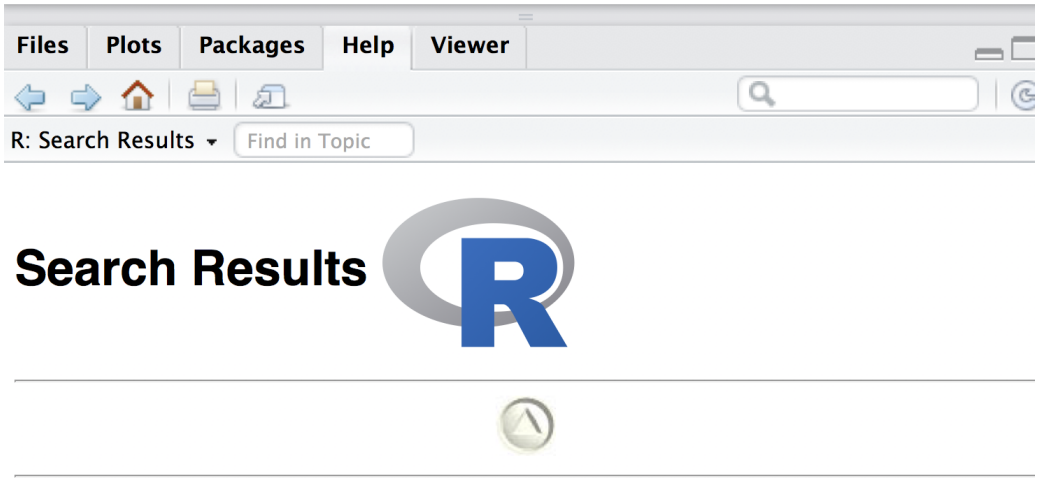
Arguments

`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

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

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



Code demonstrations:

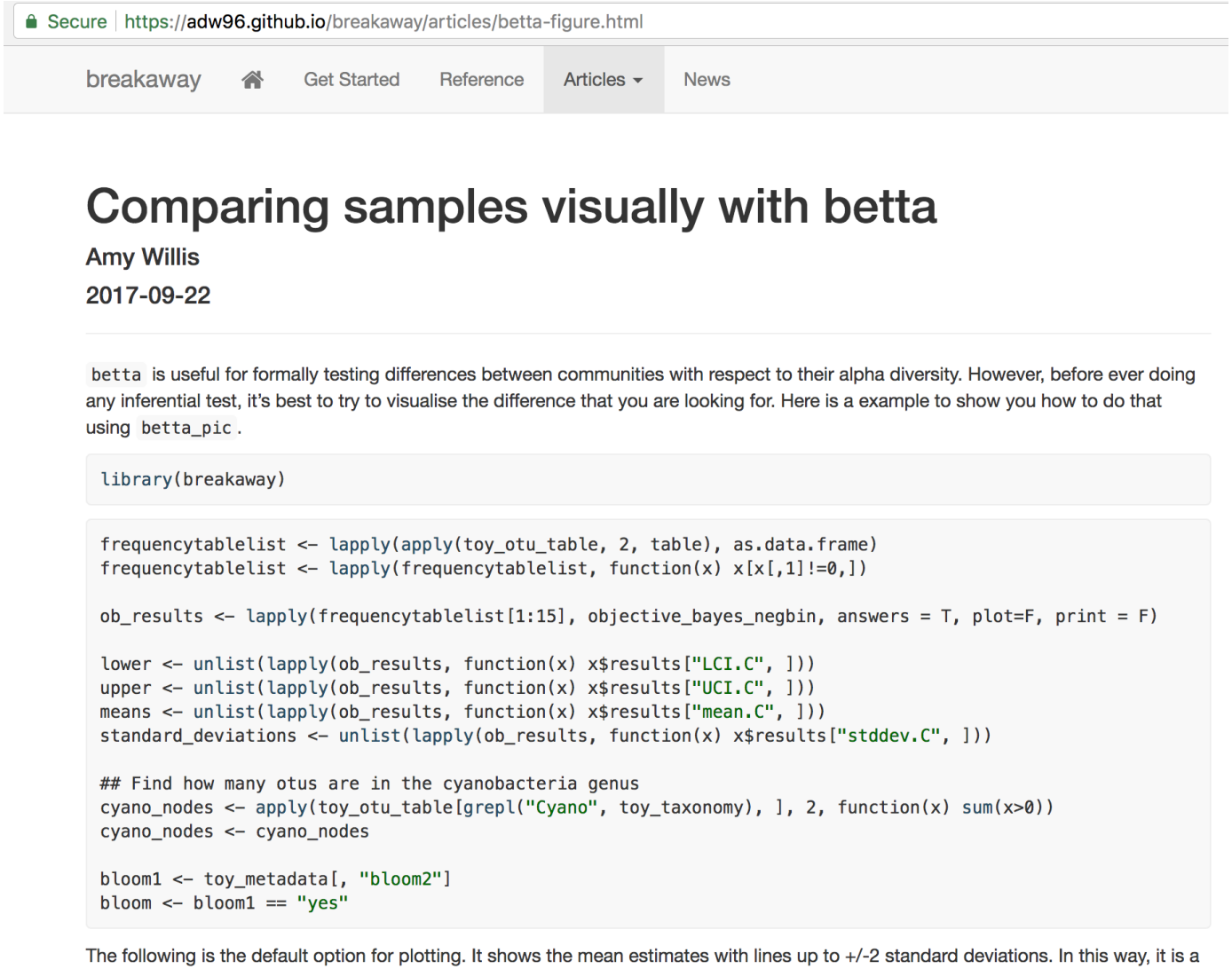
tcltk::tkcanvas	Creates a canvas widget showing a 2-D plot with data points that can be dragged with the mouse.	(Run demo in console)

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.



Keep in mind

- The user of a function assumes responsibility for giving arguments in the correct form
- arguments are ordered
  - Unnamed arguments are allocated as first arguments
  - Named arguments can be anywhere in ordering
- Not supplied arguments assume default value
  - Not supplying arguments without a default gives an error message

Don't get bogged down in reading *all* the documentation – experiment and learn from your mistakes instead!

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

RStudio

- You can use R in a terminal window by typing 'R' (once it is installed)
- Alternative: RStudio
  - Strongly recommended!
    - interface for scripts, console, figures
    - multiple R sessions in different directories
    - Projects!
    - RMarkdown (integrating `R` and Markdown/LaTex)

Workflow versus product

Your workflow:

- Your operating system
- Your code editor (e.g. RStudio)
- The name of your home directory (mine is `/Users/adwillis`)
- The code you ran yesterday
- The programs you have installed at this time

Workflow versus product

Your product:

- The raw data
- The code that needs to be run on the raw data to get results
  - including dependencies
- The report that you produce at the end of your analysis

Workflow versus product

"Any R script you write should be written assuming that it will be run from a fresh R process with working directory set to the project directory. It creates everything it needs, in its own workspace or folder, and it touches nothing it did not create..." – Jenny Bryan

Workflow versus product

You must avoid hardwiring your workflow into your product!

- Every distinct intellectual unit you work on should have its own folder
  - every data analysis, every method, every research project, every class
- You can type `install_packages` into your console (that's workflow!), but use `library` in a script (that's product!)
  - Even better, throw a `warning` if the version that is installed is not the one you thought

Workflow versus product

- Portable
  - other computers
  - robust to your own reorganisation
- Polite
  - Avoids overwriting or interfering with other projects

Workflow versus product

> *If the first line of your R script is*
>
> ```
> setwd("C:\Users\jenny\path\that\only\I\have")
> ```
>
> *I will come into your office and SET YOUR COMPUTER ON FIRE 🔥 .*

> *If the first line of your R script is*
>
> ```
> rm(list = ls())
> ```
>
> *I will come into your office and SET YOUR COMPUTER ON FIRE 🔥 .*

RStudio Projects

RStudio Projects are the implementation of this

- RStudio creates `myproj.Rproj`, which goes in your working directory for that project
- Double-click on your `Rproj` file to open a fresh RStudio instance in the correct working directory
  - Has the capacity to load your `RData`
  - Create a project with File -> New Project
- You can run multiple R sessions using multiple RStudio Projects at once

Modern statistical computing

If any of this seems like overkill

- Cut down 1000 lines to 80 lines with `tidyverse`
  - Amy 2017 >> Amy 2013
- "I had to change exactly one character to redo all of my simulations"
  - Bryan on the `simulator`
- "I didn't know you could run multiple R sessions at once"
  - Conversation with a (unnamed) fifth year
- "Could you rerun the entire analysis with this patient added?"
  - The ideal answer is "Yes!"
  - *This will happen to you*

Break

Pop quiz

What is the distribution of the median of 51 exponentially-distributed random variables with rate $= 1$?

How could we use computing power to help us?

Avoiding math with computers

To understand the distribution of the median of 51 exponentially-distributed random variables with rate $= 1$, we can

- Draw 51 Exp(1) random variables, calculate their median
- Do this again, and again, and again...

We can use the collection of medians to calculate summary statistics, draw histograms, do hypothesis testing...

Avoiding math with computers...

... and learning how to write loops in the process

```
simulations <- 10000
many_medians <- vector("numeric", simulations)
set.seed(171005)
for (i in 1:simulations) {
  my_sample <- rexp(n = 51, rate = 1)
  many_medians[i] <- median(my_sample)
}
```

What is the value of `many_medians` if `i = 5` triggers an error?

Avoiding math with computers

```
mean(many_medians) # actually: 0.70286
```

```
## [1] 0.7012355
```

```
var(many_medians) # actually: 0.01978
```

```
## [1] 0.01985761
```

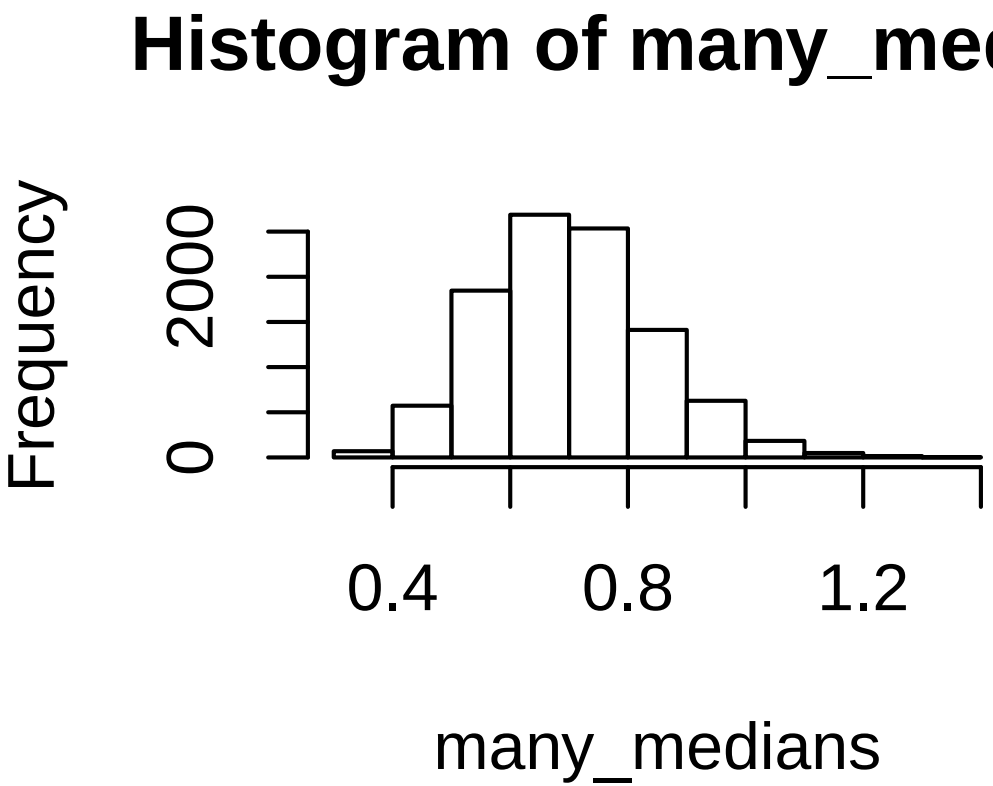
We just calculated the moments of an intractable distribution using computing!

Avoiding math with computers

We could work out almost anything about the sample median in this way!

The distribution of the median of 51 Exp(1) random variables:

```
hist(many_medians)
```



Reproducible simulations

```
set.seed(9)
rexp(4)
```

```
## [1] 1.403092 1.479229 1.255778 1.170410
```

```
rexp(4)
```

```
## [1] 0.337385913 0.005871764 0.897366012 0.971816242
```

```
set.seed(9)
rexp(4)
```

```
## [1] 1.403092 1.479229 1.255778 1.170410
```

A note on history

Why are permutation/resampling approaches to data analysis modern?

A note on history

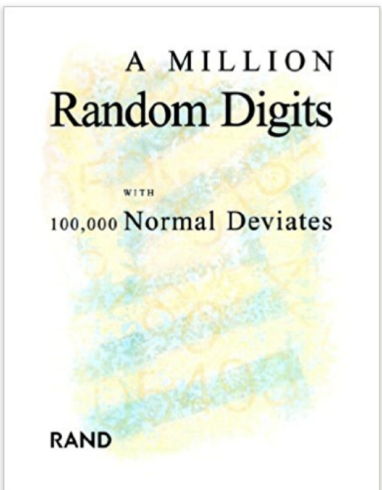
Books › Science & Math › Mathematics

A Million Random Digits with 100,000 Normal Deviates 0th Edition

by The RAND Corporation (Author)

680 customer reviews

Look inside



ISBN-13: 978-0833030474
ISBN-10: 0833030477
Why is ISBN important?

Sell yours for a Gift Card
We'll buy it for $17.57

Hardcover
$174.95

Paperback
$50.27 - $64.60

Other Sellers
See all 4 versions

Buy used $50.27

Buy new prime **$64.60**

In Stock.
Ships from and sold by Amazon.com. Gift-wrap available.
FREE Shipping for Prime members Details
Note: Available at a lower price from other sellers, potentially without free Prime shipping.

Want it Friday, Oct. 6? Order within **15 hrs 17 mins** and choose **One-Day Shipping** at checkout.
Details

List Price: $68.00 Save: $3.40 (5%)
8 New from $60.54
Qty: 1
Add to Cart
Turn on 1-Click ordering
Ship to:
Amy Willis- Seattle - 98115

More Buying Choices
8 New from $60.54 | 13 Used from $50.27
21 used & new from $50.27
See All Buying Options

Possibly containing errors?

Customer reviews

680

4.0 out of 5 stars

5 star	55%
4 star	16%
3 star	10%
2 star	5%
1 star	14%

See all verified purchase reviews

Share your thoughts with other customers

Write a customer review

Rated by customers interested in

What's this?

Math Books
4.5 out of 5 stars

Computer Books
4.1 out of 5 stars

Sports Books
3.7 out of 5 stars

Is this feature helpful? Yes No

Top customer reviews

Random? It lists almost 600 integers in numerical order!

By Obi Wan TOP 100 REVIEWER on January 27, 2015

Format: Paperback

I was duped by the title of this book. It is supposed to be about random digits. And at first glance you do see randomness.

But after reading the book a while I started seeing a pattern. I did extensive research to prove my theory. After hours of mathematical modeling I conclusively proved that there is a set of numbers in this book that it not only a pattern, but is outright sequential!

The top corner of each page (left corner on the left side pages, right corner of the right side pages) was a list of sequential numbers from 1 to 628, all in a row. No numbers are skipped. Even the prime numbers are included! At first you don't notice this because there is only 1 number on each page. But as you advance through the book you notice that the numbers keep advancing by 1 every time you turn the page.

3 comments | 67 people found this helpful. Was this review helpful to you? Yes No Report abuse

Difficult to follow

★★☆☆☆ **Too unpredictable**

By pontifex on January 24, 2011

Format: Paperback

The book is too hard to follow, the author randomly shifts from one number to another without any prior warning.

1 comment | 412 people found this helpful. Was this review helpful to you? Yes No Report abuse

Better just buy a sudoku book

★★★☆☆ **Weirdest sudoku book ever**
By John Peter O'connor on October 6, 2012
Format: Paperback

This has got to be the most useless set of sudoku puzzles ever.

In my copy of the book, all of the puzzles were already filled in which I find really annoying and what is worse, most of them have been filled in wrongly.

I have been through the whole book really carefully and only found seven puzzles that had been filled out correctly! Yes, just seven.

Well, making the best of a bad job, I am now going through the book trying to correct all of the faulty puzzles and I will then submit my corrections.

Perhaps a second edition will be more useful.

I did find last week's winning lottery numbers on page 18 though.

Comment | 139 people found this helpful. Was this review helpful to you? Yes No Report abuse

★☆☆☆☆ **Not really random**
By TDB on September 26, 2012
Format: Paperback

I bought two copies of this book. I find that the first copy perfectly predicts what the numbers will be in the second copy. I feel cheated.

Structure of a for loop

`for()` loops are not terrible, but watch out:

- First make an empty object of the correct dimension (e.g. vector, matrix, data frame) and *then* fill it in
  - Don't forget to store the output of each iteration!
- For large loops and objects, growing the output is a big slowdown
  - This is because of the way that memory is handled in R

A special set up

The only use of the index `i` was for storage.

```
simulations <- 10000
many_medians <- vector("numeric", simulations)
set.seed(171005)
for (i in 1:simulations) {
  my_sample <- rexp(n = 51, rate = 1)
  many_medians[i] <- median(my_sample)
}
```

A special set up

Since we are merely doing the same thing again and again, let's use a new function to take care of all of the admin

```
set.seed(171005)
many_medians <- replicate(simulations,
                          median(rexp(n = 51, rate = 1)))
```

The second argument to `replicate()` is the expression you want replicated

Loop indices

The index of our loop (`i`) does not need to be a vector

```
str(airquality) # a built-in dataset
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Loop indices

The index of our loop (`i`) does not need to be a vector

```
for (month in unique(airquality$Month)) {
  print(mean(airquality$Ozone[airquality$Month == month],
   na.rm = TRUE)) # prints but doesn't store
}
```

```
## [1] 23.61538
## [1] 29.44444
## [1] 59.11538
## [1] 59.96154
## [1] 31.44828
```

We'll see better ways to do this with `dplyr`

Looping over variables: apply()

```
apply(X=airquality, MARGIN=2, FUN=mean, na.rm=TRUE)
```

```
##       Ozone     Solar.R        Wind        Temp       Month
##   42.129310 185.931507    9.957516   77.882353    6.993464
```

- X: an array, usually a matrix or data frame
- MARGIN: the direction. MARGIN = 1 applies the function to each row, MARGIN = 2 applies the function to each column.
- FUN: the function to be applied
- Any other arguments to be passed to FUN

Defining your own functions

```
mean_and_sd <- function(x) {
  c(mean = mean(x), sd = sd(x))
}
apply(airquality, 2, mean_and_sd)
```

```
##       Ozone Solar.R     Wind     Temp    Month      Day
## mean     NA      NA 9.957516 77.88235 6.993464 15.80392
## sd       NA      NA 3.523001  9.46527 1.416522  8.86452
```

Defining your own functions as wrappers

```
mean_and_sd <- function(x, ...) { c(mean = mean(x, ...),
        sd = sd(x, ...)) }
apply(airquality, 2, mean_and_sd, na.rm = TRUE)
```

```
##          Ozone   Solar.R     Wind     Temp    Month
## mean 42.12931 185.93151 9.957516 77.88235 6.993464 15.8
## sd   32.98788  90.05842 3.523001  9.46527 1.416522  8.8
```

Debugging code with ellipses can be tricky! Be cautious...

Homework 1 and next week

- Slides and Homework 1:
  - `github.com/adw96/biostat561/lecture1`
- Homework 1 is due next Wednesday at 2:30 p.m.
  - `github.com/adw96/biostat561/lecture1/homework1.pdf`
- Complete Question 0 by next Monday (office hours!)
- Submission via github classroom (instructions included):
  - `classroom.github.com/classrooms/48962297-biostat561-2019`

My personal computing setup

- Mac user
- R via RStudio
  - R packages for all methods development
  - Rscript (if I have to)
- Atom as my primary text editor
  - Markdown, RMarkdown, shell scripts, data files...
  - works well with git
- Manage collaborations via GitHub wherever possible
  - Sometimes with overleaf
- Automate tasks using shell scripts (occasionally R)
- Avoid Dropbox and Microsoft Word as much is possible
- Avoid typing and mousing in favour of voicecoding and eyetracking
- Favourite programs include RSIGuard and vimium