

# An Introduction to

Yihui Xie, RStudio

2014/11/21 @ UPenn, The Warren Center

# Outline

- What is R
- Computing
  - *syntax*
  - *data manipulation*
  - *simulation*
  - *modelling*
- Graphics
  - *base graphics*
  - *ggplot2*
- Other stuff
  - *debugging*
  - *3D plots, animations, interactive graphics*

# Learning a new language



# not that scary



# What is R

R is the language for statistical computing and graphics (and more)

via

# Setting up

- Download: <http://cran.rstudio.com> (RStudio cloud mirror)
- Windows, Linux, Mac OS X
- Editor?
  - *RStudio for this workshop (alternatives: Notepad++/NppToR, Emacs/ESS, Eclipse/StatET, Vim, ...)*
  - *anything but Notepad under Windows*
- The RStudio IDE

# A note to the audience

To run the animation examples in the slides, you may need to open a native system graphics device instead of using the default RStudio device.

```
if (Sys.info()["sysname"] == "Darwin") quartz() else x11()
```

# Some quick examples

- a simple calculator
- a Mine sweeper

```
if (!require("fun")) install.packages("fun")
library(fun)
if (.Platform$OS.type == "windows") x11() else {
  x11(type = "Xlib")
}
mine_sweeper()
```

- Shiny “hello world”

```
if (!require("shiny")) install.packages("shiny")
shiny::runExample("01_hello")
```



# Comparison with other languages

- designed for statistics (`mean()`, `var()`, `lm()`, `glm()`, `rnorm()`, `boxplot()`, ...)
  - *but certainly not only for statistics*
- highly vectorized (avoid explicit loops in general)
- unbeatable number of packages (> 6000 now)
- computing on the language, lazy evaluation, ...

# Getting help

- use the question mark ? to read documentation
  - e.g. try `?lm` for help on linear models
- or `help.start()` for full HTML documentation
- Google, or your favorite searching engine
  - *the pain of the single letter R*
- StackOverflow: <http://stackoverflow.com/questions/tagged/r>
- mailing lists?

# Install add-on packages

```
## use install.packages(), e.g.  
install.packages("animation")
```

You may be asked to choose a mirror.

Too many packages?? Use the task view:

<http://cran.rstudio.com/web/views/> (e.g. Econometrics)

# Base R is still being actively updated

- Debian/Ubuntu: `sudo apt-get upgrade`
- Windows: may have to uninstall old version and install new version manually (or try the **installr** package)
- OS X: like Windows, or use Homebrew `brew update`
- easy to update packages: `update.packages()`

# Simple calculations

- assignment by `<-` or `=` (most people use the former but I'm an outlier)
- ?Arithmetic (e.g. `x + y`, `x %% y`)
- indexing by `[ ]` or `[[ ]]` or `$`

```
x <- 10
y <- 15:3

1 + 2
11 %% 2
11 %/% 2
3^4
log(10)

y[4]
y[-1] # what do negative indices mean?

z <- c(a = 1, b = 2, c = 3)
z[1]
z[-1]
z['a']
z[c('a', 'c')]
z[c(TRUE, TRUE, FALSE)]
```

# What to do with an unknown object

```
mtcars # just print it
str(mtcars) # the most useful function in R
plot(mtcars)
mtcars$mpg
mtcars[1, ]
mtcars[, 1]
mtcars[1:3, 5:7]
mtcars[, "mpg"]
mtcars$mpg
```

# Functions

## ■ `function_name(arguments)`

```
z = rnorm(100)
fivenum # what are the arguments of this function?

fivenum(z)

fivenum(x = z, na.rm = TRUE)

fivenum(z, TRUE)

f = function(a, b = 1) {
  a + b # or return(a + b)
}
f(1)
f(a = 1)
f(1, 2)
f()
```

# Conditions and loops

```
## the if-else statement
if (TRUE) {
  print(1)
} else {
  print(2)
}

## a for-loop (there is also while-loop and repeat)
s = 0
x = c(4, 2, 6)
for (i in 1:3) {
  s = s + x[i]
}
s
## the above loop is the most stupid thing to do in R
```



# Work with data

- a series of functions like `read.table()`, `read.csv()`, ...
- can work with databases too (need add-on packages like RODBC, RMySQL, ...)

```
## the tips data
tips = read.csv("http://dicook.public.iastate.edu/Army/tips.csv")

str(tips)

summary(tips)
mean(tips$bill) # index by $name
var(tips[, "tip"]) # index by character name
table(tips[, 4]) # index by column number

hist(tips$bill)
boxplot(tip ~ sex, data = tips)
plot(tip ~ bill, data = tips, col = as.integer(tips$sex))
```

# Other types of data objects

- vector
- matrix, array
- list
- ...

# Simulation

A family of functions for distributions: `dfun()`, `pfun()`,  
`qfun()` and `rfun()`

For example, `rnorm()`

# A simulation of fire flames

```
library(animation)
demo("fire", ask = FALSE) # an application of image()
```



# Another example with more scientific flavor

Q: How many times (on average) do we need to flip the coin until we get a sequence of HTH and HTT respectively? (For example, for the sequence *HHTH*, the number for HTH to appear is 4, and in *THTHTT*, the number for HTT is 6.)

A: You may consider a quick simulation.

# HTH vs HTT problem cont'd

```
coin_seq = function(v) {  
  x = rbinom(3, 1, 0.5) # sequence of coin results  
  n = 3 # a counter  
  while (!all(x == v)) {  
    x = c(x[-1], rbinom(1, 1, 0.5))  
    n = n + 1  
  }  
  n  
}  
  
set.seed(919)  
htt = replicate(10000, coin_seq(c(1, 0, 0)))  
mean(htt)  
hth = replicate(10000, coin_seq(c(1, 0, 1)))  
mean(hth)
```

# Modelling: linear regression

Take the tips data for example

```
fit1 = lm(tip ~ bill, data = tips)
summary(fit1)
```

You get

```
Call:
lm(formula = tip ~ bill, data = tips)

Residuals:
    Min       1Q   Median       3Q      Max
-3.1982 -0.5652 -0.0974  0.4863  3.7434

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.920270    0.159735   5.761 2.53e-08 ***
bill         0.105025    0.007365  14.260 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.022 on 242 degrees of freedom
Multiple R-squared:  0.4566, Adjusted R-squared:  0.4544
F-statistic: 203.4 on 1 and 242 DF, p-value: < 2.2e-16
```

# Modelling: the formula

The formula is an important component of many R functions for modelling.

```
fit2 = lm(tip ~ bill + sex, data = tips) # two variables  
  
fit3 = lm(tip ~ bill + 0, data = tips) # without intercept  
  
## you try summary() on them
```



# Two graphics systems in R

## ■ base graphics

- *the `graphics` package*
- *once drawn, no way to modify it again (have to redraw everything)*
- *functions to draw points, lines, polygons, ... like other languages*
- *many built-in types of plots (histogram, boxplot, bar chart, ...)*

## ■ grid graphics

- *the `grid` package*
- *more object-oriented: graphical elements are objects*
- *can be modified without explicitly redraw the whole plot*
- *more like an infrastructure package (no built-in plot types)*

# Add-on packages for graphics

There are many add-on packages based on the two systems; see the Graphics task view on CRAN for an overview:

<http://cran.rstudio.com/web/views/Graphics.html>

- `lattice`: Trellis plots
  - *sub-plots conditional on categorical variables*
  - *shipped with R (no need to install; just `library(lattice)`)*
  
- `ggplot2`: Grammar of Graphics
  - *nice abstraction of graphics layers*
  - <http://ggplot2.org>
  - *not a base R package, so `install.packages('ggplot2')`*

# Examples of base graphics

```
tips = read.csv("http://dicook.public.iastate.edu/Army/tips.csv")
str(tips)

## scatter plot: positive correlation with a 'constraint'
plot(tip ~ bill, data = tips)
## what is the problem with R's default choice of point shapes?

## plot() is a very 'tricky' function in R; details later

hist(tips$tip, main = "histogram of tips")
## you see nothing except a right-skewed distribution

hist(tips$tip, breaks = 30) # more bins

hist(tips$tip, breaks = 100) # what do you see now?
```

~30 years ago, the research on choosing the histogram binwidth was extremely hot in statistics, but... who cares?

# A trivial example of interactive graphics

We can change the binwidth interactively in R via many tools; one possibility is to build a GUI

```
library("shiny")
hist_ui = fluidPage(
  titlePanel("Hello Shiny!"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins", "Number of bins:",
                 min = 1, max = 50, value = 30)
    ),
    mainPanel(
      plotOutput("distPlot")
    )
  )
)
hist_server = function(input, output) {
  output$distPlot = renderPlot({
    x = faithful[, 2] # Old Faithful Geyser data
    bins = seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
}
shinyApp(ui = hist_ui, server = hist_server)
```

# Play with colors

There are many color models in R, like `rgb()`, `hsv()`, ...  
And there are built-in color names, e.g. `'red'`,  
`'purple'`. Here is an example of `rgb()`

```
rgb(1, 0, 0) # red (hexidecimal representation)
rgb(1, 1, 0) # yellow

shinyApp(
  ui = fluidPage(
    fluidRow(
      column(
        6,
        sliderInput('x1', 'Red', 0, 1, 0),
        sliderInput('x2', 'Green', 0, 1, 0),
        sliderInput('x3', 'Blue', 0, 1, 0)
      ),
      column(6, plotOutput('color'))
    )
  ),
  server = function(input, output) {
    output$color = renderPlot({
      par(bg = rgb(input$x1, input$x2, input$x3), mar = rep(0, 4))
      plot.new()
    })
  }
)

colors() # all names you can use
plot(rnorm(30), pch = 19, col = sample(colors(), 30), cex = 2)
```

# Other plots in base graphics system

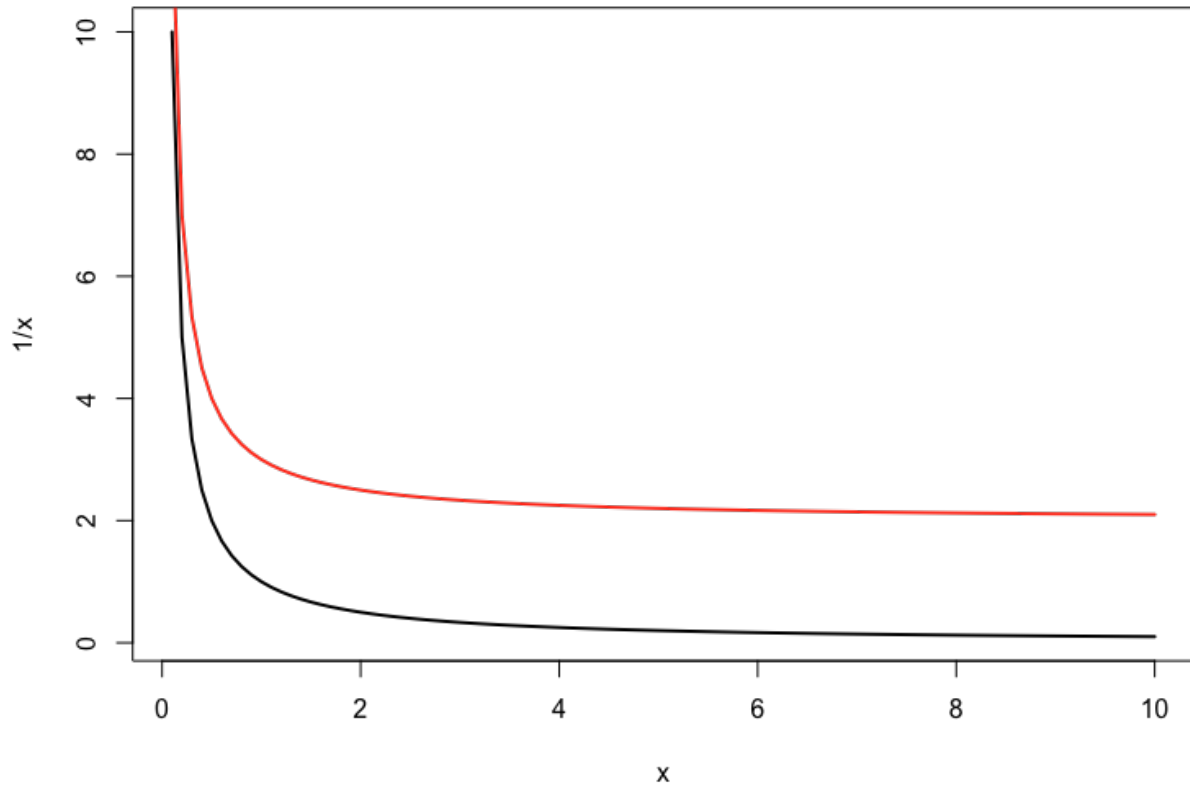
Old-fashioned but many goodies...

- open `help.start()` and take a look at the `graphics` package
- all you need to learn about base graphics is there
- many types of plots of interest: `contour()`, `filled.contour()`, `fourfoldplot()`, `mosaicplot()`, `pairs()`, `smoothScatter()`, `stripchart()`, `sunflowerplot()`, `symbols()`

# Some comments

- avoid pie charts (why?)
- avoid 3D plots (what?!)
  - *unless they are interactive (e.g. the `rgl` package)*
  - *an alternative is the contour plot*
- consider color-blind people
- The Elements of Graphing Data (William S Cleveland)
  - *order of precision (length good; angle bad; ...)*

# Trend of vertical difference between two curves





# Luke, use the source!

Only the source code is real.

```
x = seq(0.1, 10, 0.1)
plot(x, 1/x, type = "l", lwd = 2)
lines(x, 1/x + 2, col = "red", lwd = 2)
```

# Why ggplot2?

- you have to wrestle with gory details in base graphics
  - *yes, it is flexible*
  - *but you have to take care of everything*
  - *point symbols, colors, line types, line width, legend, ...*
- common tasks in graphics
  - *color the points according to the `sex` variable*
  - *different point symbols denote the `smoker` variable*
  - *darker points denote larger parties (the `size` variable)*
  - *add a smoothing/regression line on a scatter plot*
  - *...*

# Simple ggplot2 examples

We still use the `tips` data here.

```
library(ggplot2)
## different colors denote the sex variable
qplot(bill, tip, data = tips, color = sex)

## point symbols
qplot(bill, tip, data = tips, shape = smoker)

## you can manipulate ggplot2 objects
p = qplot(bill, tip, data = tips, color = size)
p

## do not like the color scheme? change it
p + scale_colour_gradient2(low = "white", high = "blue")

## faceting
qplot(tip, data = tips, facets = time ~ day)

p + geom_smooth() # smoothing line
```

# More examples in ggplot2 website

ggplot2 has its own website of documentation, which is a rich source of examples.

- boxplots: [http://docs.ggplot2.org/current/geom\\_boxplot.html](http://docs.ggplot2.org/current/geom_boxplot.html)
- contours: [http://docs.ggplot2.org/current/stat\\_contour.html](http://docs.ggplot2.org/current/stat_contour.html)
- hexagons: [http://docs.ggplot2.org/current/stat\\_binhex.html](http://docs.ggplot2.org/current/stat_binhex.html)
- Pac man chart: [http://docs.ggplot2.org/current/coord\\_polar.html](http://docs.ggplot2.org/current/coord_polar.html)
- ...

# Other packages on graphics

As mentioned before, there are many other packages based on the two graphics systems.

- `animation`: a gallery of statistical animations and tools to export animations
- `rgl`: interactive 3D plots
- `maps`: not surprisingly, for maps
- `iplots`: interactive statistical graphics
- `rggobi`: connect R with GGobi (a standalone software package for interactive stat graphics)
- `ggvis`: next generation of ggplot2, based on R + JavaScript

```
install.packages(c("animation", "rgl", "ggvis"))
```

# The animation package

The idea is simple:

```
## rotate the word 'Animation'  
for (i in 1:360) {  
  dev.hold()  
  plot(1, ann = FALSE, type = "n", axes = FALSE)  
  text(1, 1, "Animation", srt = i, col = rainbow(360)[i], cex = 7 * i/360)  
  dev.flush()  
  Sys.sleep(0.02)  
}
```

# animation examples

```
library(animation)
?brownian.motion
?quincunx
?grad.desc

## export to an HTML page
?saveHTML

demo('busybees')

demo('CLEvsLAL')
```

# rgl

Play with 3D graphics.

```
## use your mouse (drag or wheel up/down)
library(rgl)
demo("rgl")

## an artificial dataset
library(animation)
demo("pollen")
```



# RStudio's manipulate package

```
# this example only works in RStudio
library(manipulate)
y = rnorm(200)^2
manipulate({
  y1 = (y + c)^p
  qqnorm(y1)
  qqline(y1)
}, c = slider(0, 4, 0, step = 0.1), p = slider(-2, 2, 1, step = 0.1))
```

# Debugging

The function `debug` can be used to debug a function.

```
f = function(x) {  
  m = length(x)  
  x[is.na(x)] = mean(x, na.rm = TRUE) # impute by mean  
  s = sum(x^2) # sum of squares  
  s  
}  
f(c(1, NA, 2))  
  
## begin to debug the function now  
debug(f)  
  
f(c(1, NA, 2))  
  
undebug(f)
```

# Thanks!

- <http://yihui.name>
- @xieyihui
- <https://github.com/yihui>