How to Use R: An Introduction

Jeffrey R. Stevens

Department of Psychology
University of Nebraska-Lincoln

2 July 2015

What is R?

R is a free software environment for statistical computing and graphics

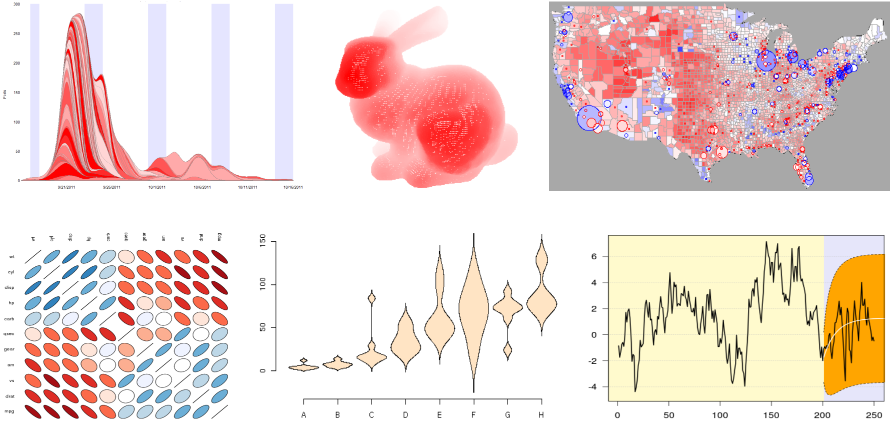
- The lingua franca for statistical computing and data analytics
- Free software version of $S/S+$ developed at Bell Labs
- Statistical package
- Graphics package
- Programming language

Why use R?

- It's free: as in speech (*libre*) and beer (*gratis*)
- Cross-platform (combined with above—give your code to anyone)

Why use R?

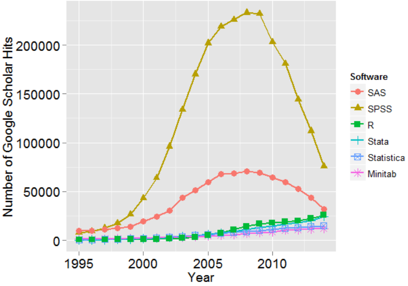
- The best graphics in the business

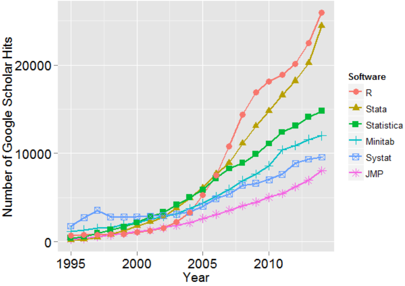


Source: R Graph Gallery.

Why use R?

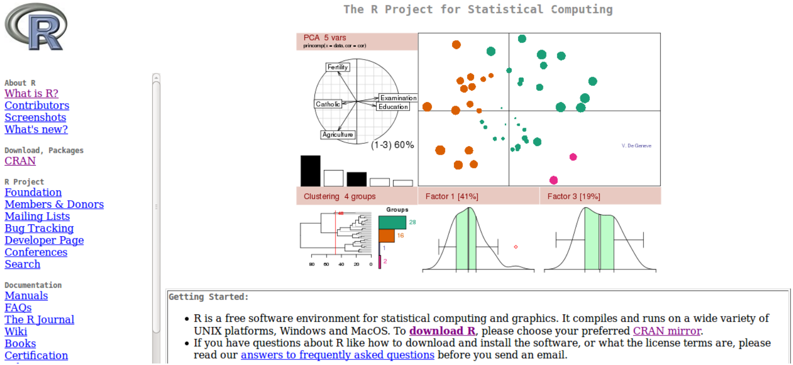
- Your colleagues are using it





Installing and maintaining R

- http://www.r-project.org



The R Project for Statistical Computing

About R
What is R?
Contributors
Screenshots
What's new?

Download, Packages
CRAN

R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search

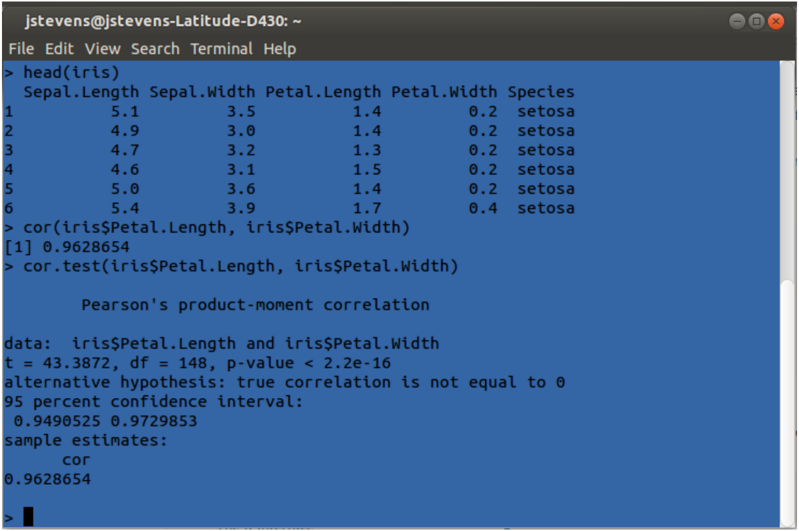
Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification

Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

How do I interact with R?

- Command line


jstevens@jstevens-Latitude-D430: ~
File Edit View Search Terminal Help

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> cor(iris$Petal.Length, iris$Petal.Width)
[1] 0.9628654
> cor.test(iris$Petal.Length, iris$Petal.Width)

	Pearson's product-moment correlation

data:  iris$Petal.Length and iris$Petal.Width
t = 43.3872, df = 148, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9490525 0.9729853
sample estimates:
      cor
0.9628654

> 
```

How do I interact with R?

Command line

- Start R core by typing `R`
- Check working directory with `getwd()`
- Set working directory with `setwd()`

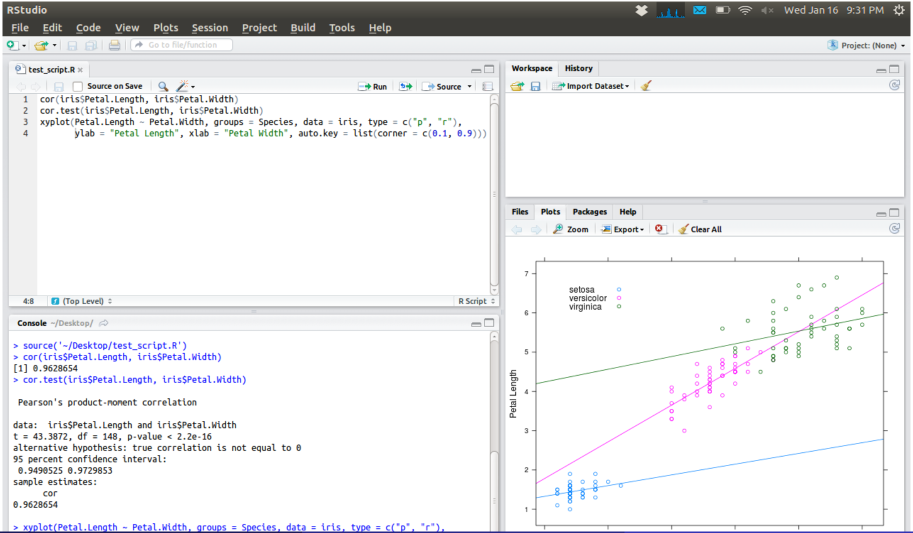
  ```
  > setwd("/home/jeff/Rwork/my_project")
  ```

- Stop R by typing

  `> q()` then `y`, `n`, or `c` to save the workspace
- Up arrow to move through history
- Tab to autocomplete

How do I interact with R?

- Rstudio (http://www.rstudio.org)



```
cor(iris$Petal.Length, iris$Petal.Width)
cor.test(iris$Petal.Length, iris$Petal.Width)
xyplot(Petal.Length ~ Petal.Width, groups = Species, data = iris, type = c("p", "r"),
       ylab = "Petal Length", xlab = "Petal Width", auto.key = list(corner = c(0.1, 0.9)))
```

```
> source('~/Desktop/test_script.R')
> cor(iris$Petal.Length, iris$Petal.Width)
[1] 0.9628654
> cor.test(iris$Petal.Length, iris$Petal.Width)

 Pearson's product-moment correlation

data:  iris$Petal.Length and iris$Petal.Width
t = 43.3872, df = 148, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9490525 0.9729853
sample estimates:
      cor
0.9628654

> xyplot(Petal.Length ~ Petal.Width, groups = Species, data = iris, type = c("p", "r"),
```

Installing and maintaining packages in R

- CRAN: Comprehensive R Archive Network
- http://cran.r-project.org/
- R plus packages
- Contains 6784 packages. Now, 6793...wait, 6794...

Installing and maintaining packages in R

- Install package with `install.packages()`

```
> install.packages("psych")
> install.packages(c("ez","Hmisc"))
```

- Install packages with RStudio
- But you must load packages each time you start a new R session

```
> library(ez)
```

or

```
> require(ez)
```

Installing and maintaining packages in R?

- Sometimes packages mask functions from R core or previously loaded packages
- To “turn off” a package, use `detach()`

```
> detach(package:ez)
```

- To uninstall a package, use `remove.packages()`

```
> remove.packages("ez")
```

- To update a package, use

```
> update.packages()
```

- All of this can be done in Rstudio

Quick quiz

- Install the `epicalc` package.
- Plot the following:

  ```
  > xyplot(1:10 ~ 2:11)
  ```
- Load the `lattice` package.
- Check that you loaded the `lattice` package.
- Plot the following:

  ```
  > xyplot(1:10 ~ 2:11)
  ```
- Detach the `lattice` package.
- Check that you detached the `lattice` package.

Getting help in R

- For the general R help page use `?`

  `> ?cor.test` (for R core functions)

  or

  `> ??layer` (for package-specific functions)
- Rstudio help

Getting help in R

- Rseek
  http://www.rseek.org
- Stack Overflow
  https://stackoverflow.com/questions/tagged/r
- Free documentation
  http://cran.r-project.org/manuals.html
  http://cran.r-project.org/other-docs.html
- Books
  Springer UseR series
  http://www.r-project.org/doc/bib/R-books.html

Quick quiz

- What does the `lm` function do?
- What is the default value of `model` in the `lm` function?
- What is the first argument for the `xyplot` function?
- What is the output of `xyplot`?
- What is listed under See Also for `codebook`?

Data types: how to store information

- Numeric (integer, double)

```
> a <- 7
> a
[1] 7
```

- Character

```
> b <- "hello world"
> b
[1] "hello world"
```

Data types: how to store information

- Logical (`TRUE` or `T` and `FALSE` or `F`)

```
> my_test <- a > 5
> my_test
[1] TRUE
> my_test2 <- b == "good-bye world"
> my_test2
[1] FALSE
```

- Logical operators `>`, `>=`, `<`, `<=`, `==`, `!=`, `%in%`

Data types

- How can you tell? `class()`

```
> x <- 10
> class(x)
[1] "numeric"
> y <- "hello world"
> class(y)
[1] "character"
> z <- x > 7
> class(z)
[1] "logical"
```

Assignment

- Assignment operators `<-` or `=`, but many prefer `<-`
- Multiple assignments:

```
>    a <- b <- 6 # assign 6 to a and b
>    a
[1] 6
>    b
[1] 6
```

Assignment

- Names are case sensitive

```
> my_variable <- 8
> My_variable <- "yes"
> my_variable == My_variable
[1] FALSE
```

- Clear individual variables with `rm()`

```
> rm(my_variable)
> my_variable
Error:  object ‘my_variable’ not found
```

- Clear all variables with (use at top of scripts)

```
> rm(list = ls(all = TRUE))
```

Quick quiz

- Assign your name to a variable called `my_name`.
- What type of data is `my_name`?
- Assign the contents of `my_name` to `name`.
- Delete `my_name`.
- Find out of `6` is in the vector `1:8`.

Data structures

- Vector
- Matrix
- Array
- List
- Data frame

Vectors

Single dimension of values

- Create by adding terms

```
> a
[1] 6
> a[2] <- 3
> a[3] <- 6
> a
[1] 6 3 6
> a[2] <- NA
> a
[1]  6 NA  6
```

Vectors

Concatenating

- Create by concatenating with `c()`

```
> my_vector <- c(1, 5, 3, 6)
> my_vector
[1] 1 5 3 6
> my_vector2 <- c(11, 14, 18, 12)
> my_vector2
[1] 11 14 18 12
> c(my_vector, my_vector2)
[1]  1  5  3  6 11 14 18 12
```

Vectors

Sequences

- Create sequences with seq()

```
> seq(from = 0, to = 20, by = 5)
[1]  0  5 10 15 20
> seq(20, 0, -5)
[1] 20 15 10  5  0
> seq(0, 1, 0.2)
[1] 0.0 0.2 0.4 0.6 0.8 1.0
```

- Use : for sequence incrementing by one

```
> 4:9
[1] 4 5 6 7 8 9
> 9:4
[1] 9 8 7 6 5 4
```

Vectors

Repetitions

- Create repetitions with `rep()`

```
> rep(0, times = 10)
 [1] 0 0 0 0 0 0 0 0 0 0
```

- Including repeating sequences

```
> rep(1:4, times = 3)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4
> rep(1:4, each = 3)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

Vectors

- Find vector length with `length()`

```
> my_vector
[1] 1 5 3 6
> length(my_vector)
[1] 4
```

Matrices

Two dimensions of values

- Matrix is built column-wise by default

```
> my_vector <- 1:8
> my_vector
[1] 1 2 3 4 5 6 7 8
> my_matrix <- matrix(my_vector, nrow = 2)
> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

Matrices

Basics

- If you want it by rows, you must tell it so

```
> my_vector
[1] 1 2 3 4 5 6 7 8
> matrix(my_vector, nrow = 2)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> matrix(my_vector, nrow = 2,
+   byrow = TRUE)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Matrices

Basics

- Can give row and column names

```
> rownames(my_matrix) <- c("top row",
+  "bottom row")
> my_matrix
           [,1] [,2] [,3] [,4]
top row       1    3    5    7
bottom row    2    4    6    8
```

Matrices

Basics

- Test whether variable is matrix with `is.matrix()`

```
> is.matrix(my_matrix)
[1] TRUE
> a
[1]  6 NA  6
> is.matrix(a)
[1] FALSE
```

- Check dimensions with `dim()`

```
> dim(my_matrix)
[1] 2 4
```

Matrices

Basics

- Indexing is by [row, column], starting with 1 (not 0)

```
> rownames(my_matrix) <- NULL
> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> my_matrix[2, 3]
[1] 6
> my_matrix[2, 3] <- 10
> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4   10    8
```

Matrices

Basics

- You can index sequences

```
> my_matrix[1, 2:4]
[1] 3 5 7
> my_matrix[1:2, 1:3]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4   10
> my_matrix[1, 2:3] <- c(1, 1)
> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    7
[2,]    2    4   10    8
```

Matrices

Basics

- Index an entire row or column by leaving it blank

```
> my_matrix
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    7
[2,]    2    4   10    8
> my_matrix[, 3]
[1]  1 10
> my_matrix[1, ]
[1] 1 1 1 7
```

Data frames

- List of named vectors of same length
- Probably most common data structure
- Create with `data.frame()`

```
> my_df <- data.frame(1:3, 8:6)
> my_df
  X1.3 X8.6
1    1    8
2    2    7
3    3    6
> names(my_df) <- c("subject", "response")
> my_df
  subject response
1       1        8
2       2        7
3       3        6
```

Data frames

- Also, with existing vectors

```
> var1 <- c(1:6)
> var2 <- c(6:1)
> var3 <- c(21:26)
> my_df2 <- data.frame(var1, var2, response = var3)
> my_df2
  var1 var2 response
1    1    6       21
2    2    5       22
3    3    4       23
4    4    3       24
5    5    2       25
6    6    1       26
```

Data frames

- Index like matrices with [row, column]

```
> my_df
  subject response
1       1        8
2       2        7
3       3        6
> my_df[2, 2]
[1] 7
```

- But also can call columns by name with $

```
> my_df$response
[1] 8 7 6
> my_df$response[2]
[1] 7
```

Data frames

- Filter out specific rows using `subset()`

```
> subset(my_df, subject == 2)
  subject response
2       2        7
```

Data frames

- Add columns with `cbind()`

```
> cbind(my_df, sex = c("f", "m", "f"))
  subject response sex
1       1        8   f
2       2        7   m
3       3        6   f
```

- Add rows with `rbind()`

```
> rbind(my_df, c(4, 5))
  subject response
1       1        8
2       2        7
3       3        6
4       4        5
```

Viewing data structurests

- Type the variable name
- Use head() to view first 6 rows

```
> head(iris)
  Sepal.Length Sepal.Width
1          5.1         3.5
2          4.9         3.0
3          4.7         3.2
4          4.6         3.1
5          5.0         3.6
6          5.4         3.9
  Petal.Length Petal.Width Species
1          1.4         0.2  setosa
2          1.4         0.2  setosa
3          1.3         0.2  setosa
4          1.5         0.2  setosa
5          1.4         0.2  setosa
```

Viewing data structures

- Use `tail()` to view last 6 rows

```
> tail(iris)
    Sepal.Length Sepal.Width
145          6.7         3.3
146          6.7         3.0
147          6.3         2.5
148          6.5         3.0
149          6.2         3.4
150          5.9         3.0
    Petal.Length Petal.Width
145          5.7         2.5
146          5.2         2.3
147          5.0         1.9
148          5.2         2.0
149          5.4         2.3
150          5.1         1.8
```

Viewing data structures

- How can you tell what kind of structure? `str()`

```
> x <- 10
> str(x)
 num 10
> y <- 1:10
> str(y)
 int [1:10] 1 2 3 4 5 6 7 8 9 10
```

Viewing data structures

- How can you tell what kind of structure? str()

```
> str(my_matrix)
 num [1:2, 1:4] 1 2 1 4 1 10 7 8
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : NULL
> str(iris)
'data.frame': 150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0
 $ Species     : Factor w/ 3 levels "setosa","versi
```

Data structures

- But some structures have complicated structures.

```
> iris_aov <- aov(Sepal.Length ~ Species, data = ir
> str(iris_aov)
List of 13
 $ coefficients : Named num [1:3] 5.01 0.93 1.58
  ..- attr(*, "names")= chr [1:3] "(Intercept)" "Spe
 $ residuals    : Named num [1:150] 0.094 -0.106 -0
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4"
 $ effects      : Named num [1:150] -71.5659 0.8025
  ..- attr(*, "names")= chr [1:150] "(Intercept)" "
 $ rank         : int 3
 $ fitted.values: Named num [1:150] 5.01 5.01 5.01
  ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4"
 $ assign       : int [1:3] 0 1 1
 $ qr           :List of 5
  ..$ qr    : num [1:150, 1:3] -12.2474 0.0816 0.081
```

Quick quiz

- Create a sequence of numbers from 0 to 100 in steps of 10.
- Create a repetition of "male" and "female" with 10 instance of each, alternating between the two.
- Create a 3 x 4 matrix. Extract the value of the second row and fourth column in the matrix.
- What is the sepal length for the last data point in the `iris` data frame?
- Assign the petal length column of the `iris` data frame to the variable `petal_length`

Functions

- Create functions for repeated use

```
> std_dev <- function(vec) {
+   sqrt((1 / (length(vec) - 1)) *
+   sum((vec - mean(vec)) ^ 2))
+ }
> std_dev(0:10)
[1] 3.316625
> my_vec <- c(5, 2, 4)
> std_dev(my_vec)
[1] 1.527525
> sd(my_vec) # R  has a standard deviation function
[1] 1.527525
```

Writing and running scripts

- Comments

```
> # this is a comment--use me often!
```

- Run script with `source()`

```
> source(my_script.R)
> source(backup/my_script.R)
```

- In RStudio, Ctrl + Shift ⇑ + s sources
- In RStudio, Ctrl + Enter runs current line in editor

Quick quiz

- Create a script called mean.R.
- In the script, create a function called my_mean that calculates the mean of a vector, using the sum and length functions.
- Use the function to calculate the mean of this vector: 1180, 270, 122.
- Source your script to find the mean.
- In the command line, apply your my_mean function to this vector: 7.3, 2.3, 5.6.

Good R practices

- Use short but descriptive names
- Separate words with _ or . (e.g., my_vector)
- Don't use functions as names (e.g., c, mean)
- Use blank space between all objects, operators (*, =, ==), and after all commas

```
> var1 <- (my_matrix[1, 3] * 6) /
+ round(exp(14), digits = 2)
```

- Write out TRUE and FALSE
- Use indents to separate nested components

```
if(length(my_vector) > 3) {
  my_vector_length <- "too long"
}
```

Where do I start?

- http://www.r-project.org
- http://www.rstudio.org
- http://tryr.codeschool.com/
- R-Uni (A List of Free R Tutorials and Resources in University webpages)
- http://www.r-bloggers.com/
- jeffrey.r.stevens@gmail.com

These slides are available at https://imnotamember.github.io/Programming-Workshop/