Introduction to **R**

Jason Huff, QB3 CGRL—UC Berkeley

April 15, 2016

Installing R

- **R** is constantly updated and you should download a recent version; the version when this workshop was written was 3.2.4
- I also highly recommend the free open source edition of RStudio integrated development environment
- Finally, for this workshop install the `ggplot2` package using the following code:

```
install.packages("ggplot2")
```

What is R?

R is a free and powerful statistical programming language, and also a project, and also an environment. It's an interpreted language, as opposed to a compiled language. That simply means you run code directly in the **R** program; there's no need to compile the code you write. That makes it a bit like **Perl** or **Python**, but with completely different syntax. **R** syntax originally developed from **S** as a **GNU** project, and is probably most similar to the proprietary programming languages **Stata** and **MATLAB**.

What will you learn about R in this workshop?

- Data structures
- Basic functions
- Plotting
- File handling
- Packages
- User-defined functions and loops
- Statistical tests
- Advanced use examples: integration into pipelines, interactive tools, publishing and reproducible research

---

Getting Started in R

The most useful function for a new user:

```
help.start()
```

Under "An Introduction to R", there is a great "sample session" in Appendix A.

Note: if you just type `help.start`, you'll see the function code, which will be pretty meaningless to you; the general syntax to *execute* a function in R is `function()`

Data structures

R reads in data and operates on it as objects. This makes **R** an object-oriented language, which is one reason that it's easier for beginners to learn. It also makes it easily extensible.

Let's make an object using the `<-`, which is equivalent to the `=` in most languages for defining objects:

```
new_obj <- 1
```

What did we make?

```
new_obj
```

The naming of objects can be fairly arbitrary in **R**, and you can name them with underscores, dots and different cases:

```
new_obj <- 1
new.obj <- 1
newObj <- 1
```

R is case sensitive. Try to avoid naming objects the same as existing functions and objects in R.

To save memory and/or clean up the workspace, you can remove objects with the `rm` remove function:

```
rm(new_obj)
rm(new.obj)
rm(newObj)
```

Aside—the most important RStudio shortcuts:

- The **tab** key can autocomplete
- **cmd** + **the up arrow** (**control** + **the up arrow** on Windows) provides suggestions from only what you have previously typed
- **cmd** + **return** (**control** + **enter** on Windows) runs the code selected in the editor
- **esc** aborts and erases the current line in the console (**control** + **c** from within **R** outside of RStudio)

The general scheme of R data structures

Dimension	Homogeneous elements	Heterogeneous elements
1	"Atomic" vector	List
2	Matrix	Data Frame
N	Array	

Use the combine function `c` to create "atomic" vectors, of which there a 4 most common types in **R**:

```
# R assumes here that you want a double vector
dbl_var <- c(1, 2.5, 4.5)

# With the L suffix, you can force an integer vector rather than a double
int_var <- c(1L, 6L, 10L)

# Use TRUE and FALSE (or T and F) to create logical vectors
log_var <- c(TRUE, FALSE, T, F)

# A character vector
char_var <- c("these are", "some strings")
```

Let's see the built-in example data using the `data` sets function:

```
data()
```

Ok, let's see what is in the object `pressure`

```
pressure
```

```
##    temperature pressure
## 1            0   0.0002
## 2           20   0.0012
## 3           40   0.0060
## 4           60   0.0300
## 5           80   0.0900
## 6          100   0.2700
## 7          120   0.7500
## 8          140   1.8500
## 9          160   4.2000
## 10         180   8.8000
## 11         200  17.3000
## 12         220  32.1000
## 13         240  57.0000
## 14         260  96.0000
## 15         280 157.0000
## 16         300 247.0000
## 17         320 376.0000
## 18         340 558.0000
## 19         360 806.0000
```

What kind of **R** data structure is this object?

We can interrogate an object using the structure display `str` function:

```
str(pressure)
```

How do we access or subset parts of an object?

```
pressure$temperature
```

We can subset an individual element from a vector:

```
pressure$temperature[5]
```

We can coerce one data type into another:

```
pressure.matrix <- as.matrix(pressure)
```

We can subset individual columns and rows:

```
pressure.matrix[1,]
pressure.matrix[,1]
pressure.matrix[4,2]
```

true/false subsetting

Basic functions

First, we use a function `rnorm` to randomly generate a normally distributed sample of 10 data points:

```
# Make 10 data points with mean 3.4 and standard deviation of 1
a <- rnorm(10, mean=3.4, sd=1)
a
```

We can calculate the sample average:

```
mean(a)
median(a)
```

How about measures of dispersion?

```
# sample variance
var(a)

# sample standard deviation
sd(a)

# interquartile range
IQR(a)

# median absolute deviation
mad(a)
```

The quicker method for some basic stats is to use the `summary` function in **R**:

```
summary(a)
```

Finally, quitting the **R** program

```
q()
```

Each function has different input and output and arguments (parameters), which is all very confusing. How do I use a particular function?

Use the code `?<topic>` to learn more about a particular function. For example, let's learn about the `plot` generic X-Y plotting function using the `help` function, which can be used by `?`:

```
?plot
```

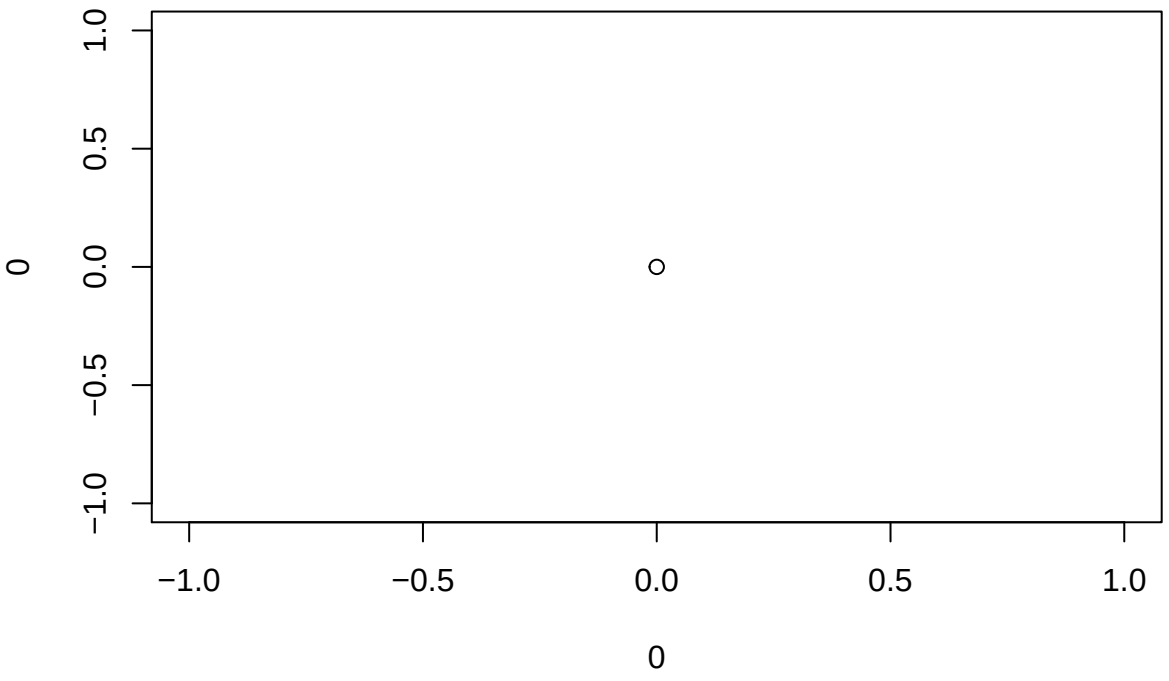
and the `par` graphical parameters function:

```
?par
```

Plotting

The most basic X-Y plot using `plot`:

```
plot(0,0)
```



Ok, what does the pressure data look like?

```r
# X-Y plot of the pressure data set
plot(pressure)

# Fit a locally weighted scatter plot smoothing  (LOWESS) by polynomial regression
fit <- lowess(pressure)

# Display the LOWESS fit
lines(fit)

# We can make the LOWESS less smooth by tweaking the argument f
fit.lesssmooth <- lowess(pressure, f=1/3)

# Display the LOWESS fit
lines(fit.lesssmooth, lty=2)
```

Let's look at real car engine data instead:

```r
plot(mtcars$disp, mtcars$mpg)

# We'll change some graphical arguments to make it look better
plot(mtcars$disp, mtcars$mpg, xlab="Engine displacement", ylab="Miles per gallon", main="MPG of engines

# We can add a LOWESS fit
fit.engines <- lowess(mtcars$disp, mtcars$mpg)
lines(fit.engines)
```

File handling

Find out what directory you're in and change directory:

```r
getwd()
setwd("")
```

```r
# Read in a table
tab <- read.table("temp-pressure.txt")

# Write out a table
write.table(t(tab), "temp-pressure_transposed.txt")

# Write out a table without the row and column names
write.table(t(tab), "temp-pressure_transposed.txt", col.names=FALSE, row.names=FALSE)
```

Let's write a plot to a file:

```r
png("EngineMPG.png",width=1000,height=700)
plot(mtcars$disp, mtcars$mpg, xlab="Engine displacement", ylab="Miles per gallon", main="MPG of engines
fit.engines <- lowess(mtcars$disp, mtcars$mpg)
lines(fit.engines)
dev.off()
```

Packages

We'll introduce now the **ggplot2** package. A package is a set of functions, data and documentation authored to add functionality to **R**.

ggplot2 is developed by Hadley Wickham, Chief Scientist for RStudio. **ggplot2** is a slightly more user-friendly plotting environment within **R** that lets you make accurate plots that incorporate features of the grammar of graphics.

If you'd lke to become a **ggplot2** expert or see what it's capable of, check out Wickham's YouTube video A Backstage Tour of ggplot2

Let's load the **ggplot2** package we installed already:

```
library(ggplot2)
```

Now you can learn about the package, using the search version of `help`:

```
??ggplot2
```

The most general up-to-date documentation is on the ggplot2 website

```
# We can use a couple lines of code to break things down in a straightforward way
p <- ggplot(mpg, aes(class, hwy))
p + geom_boxplot()

# We can also change the presentation of the data in many informative ways
p + geom_boxplot() + geom_jitter(width = 0.2)
```

User-defined functions and loops

```
# Bootstrap
for (i in 1:10) {
  print(sample(a, 10, replace=TRUE))
}

bootstrap.custom <- function(data, x) {
for (i in 1:x) {
  print(sample(data , 10, replace=TRUE))
}
}

bootstrap.custom(a, 10)
```

Statistical tests

Next, we'll randomly generate a normally distributed samples of 10 data points with a different mean:

```
# Make 10 data points with mean 4.8 and standard deviation of 1
b <- rnorm(10, mean = 4.8, sd = 1)
b
```

We can easily see their distributions in a boxplot:

```
boxplot(a, b, names = c("a", "b"))
```

Are they correlated using Pearson's correlation coefficient?

```
cor(a, b, method = "pearson")
```

We can easily perform Student's t-Test:

```
t.test(a, b, var.equal=TRUE)
```

Advanced use: integration into pipelines

The main thing that is useful is the ability to run your **R** code non-interactively by not invoking the **R** window.

In Mac OS X terminal or Linux command line:

> R CMD BATCH [options] my_script.R [outfile]

Windows command:

> "C:\Program Files\R\R-3.2.4\bin\R.exe" CMD BATCH --vanilla --slave "C:\my projects\my_script.R"

Advanced use: interactive tools

Choose one of a select few built-in data sets:

Does not properly render in PDF – see the live Shiny link for the workshop instead

---

Advanced use: publishing results

Let's write a postscript:

```
postscript("EngineMPG.ps",width=7,height=5)
plot(mtcars$disp, mtcars$mpg, xlab="Engine displacement", ylab="Miles per gallon", main="MPG of engines
fit.engines <- lowess(mtcars$disp, mtcars$mpg)
lines(fit.engines)
dev.off()
```

Advanced use: reproducible research

This whole file is an **R** Markdown document.

> Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.
> For more details on using **R** Markdown see http://rmarkdown.rstudio.com.

I've found the **R** Markdown cheatsheet really useful for getting up and running.

What did we cover in this workshop?

- How to use the basic data structures and functions in **R**
- How to plot (graph) to explore data and make figures
- How to read in and write out data and plots
- How to use package functions and define your own functions and loops
- How to perform statistical tests in **R**
- Brief introduction to some advanced examples: integration into pipelines, interactive tools, publishing and reproducible research

---

How do I keep learning about R?

```
help.start()
```

The Manual: An Introduction to R by W. Venables, D. Smith and the R Core Team
In Appendix A is a quick sample session to get started in **R**, some of which is adapted in this workshop. Also, Chapter 12 has a lot of good info about graphics.

If you want a super-dense companion, it can be very useful when you become more familiar with **R** or if you're familiar with other languages. Some of the companion material was adapted in this workshop.

Roger Peng's YouTube videos on **R**:

- Reading/Writing Data: Part 1
- Reading/Writing Data: Part 2
- Lecture 2b: Data Types

- Lecture 2b: Subsetting
- Lecture 2c: Vectorized Operations

The 3-/4-day SCF D-Lab R bootcamp at Berkeley may happen this year around August or perhaps in 2017. The Department of Statistics has online Tutorials on advanced **R** topics.

What if I want to do something really specific, and I have no idea what to do and/or the code I wrote doesn't work?

Example: I want to make a kernel density plot, and I have no idea what the function is in **R**

Google it
