R Basics

Fundamental Techniques in Data Science

Kyle M. Lang

Department of Methodology & Statistics
Utrecht University

Outline

Attribution

This course was originally developed by Gerko Vink. You can access the original version of these materials on Dr. Vink's GitHub page: `https://github.com/gerkovink/fundamentals`.
Some of the materials in this repository have been modified. Any errors or inaccuracies introduced via these modifications are fully my own responsibility and shall not be taken as representing the views and/or beliefs of Dr. Vink.
www.gerkovink.com/fundamentals

OPEN–SOURCE SOFTWARE

What is “Open-Source”?

R is an open-source software project, but what does that mean?

- Source code is freely available to anyone who wants it.
    - Free Speech, not necessarily Free Beer
- Anyone can edit the original source code to suit their needs.
    - Ego-less programming
- Many open source programs are also “freeware” that are available free of charge.
    - R is both open-source and freeware

Strengths of Open-Source Software

FREEDOM

- If the software you are using is broken (or just limited in capability), you can modify it in any way you like.
- If you are unsure of what the software you are using is doing, you can dig into the source code and confirm its procedures.
- If you create some software, you can easily, and independently, distribute it to the world.
    - There is a global community of potential users that are all linked via a common infrastructure that facilitates open-source software development and distribution.

Strengths of Open-Source Software

Peer Review

- Every user of open-source software is a reviewer of that software.
- What “bedroom programmers” lack in term of quality control procedures is overcome by the scrutiny of a large and empowered user-base.
    - When we use closed source software, we are forced to trust the honesty of the developing company.
    - We have no way of checking the actual implementation.

Strengths of Open-Source Software

ACCESSIBILITY

- Many open-source programs (like R) can be downloaded, for free, from the internet.
  - You can have R installed on all of you computers (and your mobile phone, your car's info-tainment system, your microwave, your clock-radio, ...).
  - No need to beg, borrow, or steal funds to get yourself up-and-running with a cutting-edge data analysis suite.
- Licensing legality is very simple—no worries about being sued for installing open-source software on "too many" computers.
- Open-source software tends to run on more platforms than closed-source software will.

A Note on Licensing

Some popular open-source licenses:

- The GNU General Public License (GPL)
  - `http://www.gnu.org/licenses/gpl-3.0.en.html`
- The GNU Lesser General Public License (L-GPL)
  - `http://www.gnu.org/licenses/lgpl-3.0.en.html`
- The Apache License
  - `http://www.apache.org/licenses/`
- The BSD 2-Clause License (FreeBSD License)
  - `http://opensource.org/licenses/BSD-2-Clause`
- The MIT License
  - `https://opensource.org/licenses/MIT`

A Note on Licensing

Many open-source licenses (e.g., GPL, L-GPL) “copyleft” their products.

- Copyleft is designed to ensure that open-source software cannot be closed.
  - I can’t take your copylefted software, repackage it, and sell it in violation of your original licensing terms.

Other open-source licenses (e.g., BSD-Types, Apache, MIT) are non-copyleft, “permissive” licenses.

- Many of these licenses are designed to promote commercialization of open-source products.
  - E.g., allowing a student to develop a company selling a product they developed for their dissertation

THE R STATISTICAL PROGRAMMING LANGUAGE

What is R?

R is a holistic (open-source) software system for data analysis and statistical programming.

- R is an implementation of the S language.
  - Developed by John Chambers and colleagues
    - **?**
    - **?**
    - **?**
    - **?**
- Introduced by **?**.
  - Currently maintained by the *R Core Team*.
- Support by thousands of world-wide contributors.
  - Anyone can contribute an R package to the *Comprehensive R Archive Network* (CRAN)
  - Must conform to the licensing and packaging requirements.

What is R?

I prefer to think about R as a *statistical programming language*, rather than as a data analysis program.

- R **IS NOT** its GUI (no matter which GUI you use).
- You can write R code in whatever program you like (e.g., RStudio, EMACS, VIM, Notepad, directly in the console/shell/command line).
- R can be used for basic (or advanced) data analysis, but its real strength is its flexible programming framework.
    - Tedious tasks can be automated.
    - Computationally demanding jobs can be run in parallel.
    - R-based research *wants* to be reproducible.
    - Analyses are automatically documented via their scripts.

What is RStudio?

RStudio is an integrated development environment (IDE) for R.

- Adds a bunch of window dressing to R
- Also open-source
- Both free and paid versions

R and RStudio are independent entities.

- You do not need RStudio to work with R.
- You are analyzing your data with R, not RStudio
    - RStudio is just the interface through which you interact with R.

Getting R

You can download R, for free, from the following web page:

- `https://www.r-project.org/`

Likewise, you can freely download RStudio via the following page:

- `https://www.rstudio.com/`

What to Expect when Opening R

As noted above, we have many ways of interacting with R:

- Base R
- EMACS
- RStudio
- Text-only console (i.e., even more base R)

How R Works

R is an interpreted programming language.

- The commands you enter into the R *Console* are executed immediately.
- You don’t need to compile your code before running it.
- In this sense, interacting with R is similar to interacting with other syntax-based statistical packages (e.g., SAS, STATA, Mplus).

How R Works

R mixes the *functional* and *object-oriented* programming paradigms.

FUNCTIONAL

- R is designed to break down problems into functions.
- Every R function is a first-class object.
- R uses pass-by-value semantics.

OBJECT-ORIENTED

- Everything in R is an object.
- R functions work by creating and modifying R objects.
- The R workflow is organized by assigning objects to names.

Interacting with R

When working with R, you will write *scripts* that contain all of the commands you want to execute.

- There is no “clicky-box” Tom-foolery in R.
- Your script can be run interactively or in “batch-mode”, as a self-contained program.

The primary purpose of the commands in your script will be to create and modify various objects (e.g., datasets, variables, function calls, graphical devices).

Getting Help

Everything published on the Comprehensive R Archive Network (CRAN), and intended for R users, must be accompanied by a help file.

- If you know the name of the function (e.g., `anova()`), then execute `?anova` or `help(anova)`.
- If you do not know the name of the function, type `??` followed by your search criterion.
  - For example, `??anova` returns a list of all help pages that contain the word "anova".

Alternatively, the internet will tell you almost everything you'd like to know

- Sites such as `http://www.stackoverflow.com` and `http://www.stackexchange.com` can be very helpful.
- If you google R-related issues, include "R" somewhere in your search string.

Packages

Packages give R additional functionality.

- By default, some packages are included when you install R.
- These packages allow you to do common statistical analyses and data manipulation.
- Installing additional packages allows you to perform state-of-the-art statistical analyses.

These packages are all developed by R users, so the throughput process is very timely.

- Newly developed functions and software are readily available
- Software implementations of new methods can be quickly dissemintated
- This efficiency differs from other mainstream software (e.g., SPSS, SAS, MPlus) where new methodology may take years to be implemented.

A list of available packages can be found on CRAN.

Installing & Loading Packages

Install a package (e.g., ‘mice‘):

```r
install.packages("mice")
```

There are two ways to load a package into R

```r
library(stats) require(stats)
```

Working Directory

Directory Structure

RStudio Projects

RStudio projects provide a convenient way to organize all of the code and supporting resources for a given research project.

- Every project has its own history
- Every research project can have its own RStudio project
- Every project can have its own directory
- Every project can have its own version control system
- R-studio projects can relate to GitHub (or other online) repositories

Data I/O

Built-In R Data & Workspaces

We have many ways to read data into R

```
## Load the built-in 'bfi' data from the 'psychTools' package
data(bfi, package = "psychTools")

Error in find.package(package, lib.loc, verbose = verbose):  there is no
package called 'psychTools'

## Access the documentation for the 'bfi' data
?psychTools::bfi

Error in find.package(if (is.null(package)) loadedNamespaces() else
package, :  there is no package called 'psychTools'

##################################################################################
## PRACTICE PROBLEM 3.1
##
## (a) Use the data() function to load the 'Cars93' dataset from the 'MASS'
##     package.
## (b) Use the dim() function to check the dimensoins of the 'Cars93' data.
##     - How many rows?
##     - How many columns?
##
##################################################################################
```

Delimited Data Types

```r
## Load the 'diabetes' data stored in tab-delimited file '../data/diabetes.txt'
diabetes <- read.table(paste0(dataDir, "diabetes.txt"),
                       header = TRUE,
                       sep = "\t")

## Load the 2017 UTMB data from the comma-seperated file '../data/utmb_2017.csv'
utmb1 <- read.csv(paste0(dataDir, "utmb_2017.csv"))

### NOTE: For EU-formatted CSV files, use read.csv2()

## Load the 'titanic' data stored in R data set '../data/titanic.rds'
titanic <- readRDS(paste0(dataDir, "titanic.rds"))

################################################################################
## PRACTICE PROBLEM 3.2
##
## (a) Load the dataset saved as '../data/diabetes.rds'.
## (b) Use the str() function to compare the structure of the data you loaded in
##     (a) to the diabetes data loaded above using the read.table() function.
##     - Are there any differences between these two objects? If so, what are
##       the differences?
##
###########
```

SPSS Data

Reading data in from other stats packages can be a bit tricky. If we want to read SAV files, the two most popular options are foreign::read.spss() and haven::read$_{s}pss()$.

```
## Load the foreign package:
library(foreign)

## Use foreign::read.spss() to read '../data/mtcars.sav' into a list
(mtcars1 <- read.spss(paste0(dataDir, "mtcars.sav")))

$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8
[12] 16.4 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5
[23] 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4

$cyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4
[29] 8 6 8 4

$disp
 [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8
[10] 167.6 167.6 275.8 275.8 275.8 472.0 460.0 440.0  78.7
[19]  75.7  71.1 120.1 318.0 304.0 350.0 400.0  79.0 120.3
```

SPSS Data

```
## Load the packages:
library(haven)
library(labelled)

Error in library(labelled):  there is no package called 'labelled'

## Use haven::read_spss() to read '../data/mtcars.sav' into a tibble
(mtcars4 <- read_spss(paste0(dataDir, "mtcars.sav")))

# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec       vs       am
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl+l>  <dbl+l>
 1  21       6  160    110  3.9   2.62  16.5 0 [V-S~ 1 [Man~
 2  21       6  160    110  3.9   2.88  17.0 0 [V-S~ 1 [Man~
 3  22.8     4  108     93  3.85  2.32  18.6 1 [Str~ 1 [Man~
 4  21.4     6  258    110  3.08  3.22  19.4 1 [Str~ 0 [Aut~
 5  18.7     8  360    175  3.15  3.44  17.0 0 [V-S~ 0 [Aut~
 6  18.1     6  225    105  2.76  3.46  20.2 1 [Str~ 0 [Aut~
 7  14.3     8  360    245  3.21  3.57  15.8 0 [V-S~ 0 [Aut~
 8  24.4     4  147.    62  3.69  3.19  20   1 [Str~ 0 [Aut~
 9  22.8     4  141.    95  3.92  3.15  22.9 1 [Str~ 0 [Aut~
10  19.2     6  168.   123  3.92  3.44  18.3 1 [Str~ 0 [Aut~
# ... with 22 more rows, and 2 more variables: gear <dbl>
```

Excel Data

```
## Load the packages:
library(readxl)
library(openxlsx)

## Use the readxl::read_excel() function to read the data from the 'titanic'
## sheet of the Excel workbook stored at '../data/example_data.xlsx'
titanic2 <- read_excel(paste0(dataDir, "example_data.xlsx"), sheet = "titanic")

## Use the openxlsx::read.xlsx() function to read the data from the 'titanic'
## sheet of the Excel workbook stored at '../data/example_data.xlsx'
titanic3 <- read.xlsx(paste0(dataDir, "example_data.xlsx"), sheet = "titanic")

str(titanic2)

tibble [887 x 8] (S3: tbl_df/tbl/data.frame)
 $ survived          : chr [1:887] "no" "yes" "yes" "yes" ...
 $ class             : chr [1:887] "3rd" "1st" "3rd" "1st" ...
 $ name              : chr [1:887] "Mr. Owen Harris Braund" "Mrs. John Bradley (Flo
 $ sex               : chr [1:887] "male" "female" "female" "female" ...
 $ age               : num [1:887] 22 38 26 35 35 27 54 2 27 14 ...
 $ siblings_spouses: num [1:887] 1 1 0 1 0 0 0 3 0 1 ...
 $ parents_children: num [1:887] 0 0 0 0 0 0 0 1 2 0 ...
 $ fare
```

Workspaces & Delimited Data

All of the data reading functions we saw earlier have complementary data writing versions.

```r
## The save() function writes an R workspace to disk
save(boys, file = paste0(dataDir, "tmp.RData"))

## For delimited text files and RDS data, the write.table(), write.csv(), and
## saveRDS() function do what you'd expect
write.table(boys,
            paste0(dataDir, "boys.txt"),
            row.names = FALSE,
            sep = "\t",
            na = "-999")
write.csv2(boys, paste0(dataDir, "boys.csv"), row.names = FALSE, na = "")
saveRDS(boys, paste0(dataDir, "boys.rds"))
```

SPSS Data

To write SPSS data, the best option is the haven::write$_s av() function$.

```
write_sav(mtcars2, paste0(dataDir, "mctars2.sav"))

## write_sav() will preserve label information provided by factor variables and
## the 'haven_labelled' class, but not by attributes
write_sav(mtcars4, paste0(dataDir, "mctars4.sav"))
write_sav(mtcars5, paste0(dataDir, "mctars5.sav"))

Error in is.data.frame(data):  object 'mtcars5' not found
```

Excel Data

The ’openxlsx’ package provides a powerful toolkit for programmatically building Excel workbooks in R and saving the results. Of course, it also works for simple data writing tasks.

```r
## Use the openxlsx::write.xlsx() function to write the 'diabetes' data to an
## XLSX workbook
write.xlsx(diabetes, paste0(dataDir, "diabetes.xlsx"), overwrite = TRUE)

## Use the openxlsx::write.xlsx() function to write each data frame in a list to
## a seperate sheet of an XLSX workbook
write.xlsx(list(titanic = titanic, diabetes = diabetes, mtcars = mtcars),
           paste0(dataDir, "example_data.xlsx"),
           overwrite = TRUE)
```

Some Programming Tips

- Keep your code tidy.
- Use comments to clarify what you are doing.
- When working with functions in RStudio, use the TAB key to quickly access the documentation of the function's arguments.
- Give your R scripts and objects meaningful names.
- Use a consistent directory structure and RStudio projects.

General Style Advice

Use common sense and BE CONSISTENT.

- Browse through the tidyverse style guide.
  - The point of style guidelines is to enforce a common vocabulary.
  - You want people to concentrate on *what* you're saying, not *how* you're saying it.
- If the code you add to a project/codebase looks drastically different from the extant code, the incongruity will confuse readers and collaborators.

Spacing and whitespace are your friends.

- 'a¡-c(1,2,3,4,5)'
- 'a ¡- c(1, 2, 3, 4, 5)'
- At least put spaces around assignment operators and after every comma!

References