Working with R

A University of Queensland Advanced Workshop

Session 1: Introduction and Outline

Bill Venables, CSIRO/Data61, Dutton Park

Rhetta Chappell, Griffith University

2–5 February, 2021

Contents

1 Preliminaries

- **R** is a *programming language* that grew out of a suite of *interactive* tools for data analysis, graphics and modelling.
- The emphasis has changed from an interactive focus to working in a *reproducible* way.
- We also recommend working in a *platform neutral* and *portable* way.
  - **RStudio** *projects* are key.
- We assume at least some superficial familiarity with **R**
  - our aim in this workshop is to deepen and strengthen your understanding of **R** and how to work productively *with* **R**.
- This is *not* a Statistics course, but statistics and modelling will play a large part in what we do.
- To work with **R** is, at some level at least, to *program*.

1.1 The RStudio IDE: gratuitous advice

- **RStudio** is an IDE ('integrated development environment') which facilitates using **R**, in a reproducible way.
- It comes in a free version and also with several levels of commercial support. The free version is (so far!) entirely adequate.
- *You do not have to use it*, but if you collaborate with colleagues using **R**, it is practically certain they will be.
- Learn to use it *properly* rather than just "well enough" for your immediate goals.
  Use **R** to *learn how to learn*: you will need the knack often in future.
- The key to using **RStudio** properly is to work with *projects*, keeping things compact and portable.
- The materials for this course are distributed as an **RStudio** project, developed on **Linux**. The should nevertheless work seamlessly across platforms.

1.2 The concept of an **RStudio** *project*

- The idea is to set aside a dedicated master folder, or directory, in which to locate all the materials for a discrete piece of work.
  - Sub-folders will normally be used for various purposes, including data sets, scripts, reports, graphics, &c.
  - In addition, **RStudio** may also create specific sub-folders to hold the output materials for document rendering.
  - In this course we will also have *hidden* sub-folders to hold cached objects. These have names such as `.R_Cache` and `.Robjects`.
- A new **RStudio** project is created from within **RStudio** itself using a simple menu-driven process.
  This will also put inside your project folder several house-keeping sub-folders and files, one of which is a tag.
  If the project folder is called, for example, `MyProject`, the tag file within it will be called `MyProject.Rproj`

1.2.1 Using **RStudio** projects: The tag file

To start **RStudio** *within a project*

- Do not click any **RStudio** button to start the program.
- Begin by using your *file locator* (or other means) to find the project on which your wish to work.
- Double-click on the tag file, e.g. `MyProject.Rproj`.

  This will start **RStudio** with the project context saved from the last time you worked on it.

Alternatively, for those people unable *not* to begin by clicking on the **RStudio** button:

- Start **RStudio** by clicking on the button, if you *must*.
- Click on the `File -> Open Project` menu.
- Navigate to the project directory you wish to work upon and select the tag file as above.

1.2.2 Setting global and project options

To make your working environment more comfortable, you may set **RStudio** options.

- The *Global* options will apply every time you use **RStudio** on that machine.

  Edit these using `Tools -> Global Options`.

- In addition, there are a few *Project* options, which may be set for the current project only, but will apply when next you work on that project.

  Edit these using `Tools -> Project Options`.

- One hidden feature of **RStudio** worth checking out are code snippets. These can greatly help when coding or drafting documents.

  See `Tools -> Globap Options -> Code -> Snippets`

1.3 Working protocols

- Establish your primary data sources early.
- Establish a clear path from your primary data sources to **R**, and be prepared for changes. Hence:
- Work reproducably. Two approaches:
  - Use **R** scripts, with a clearly defined sequence to reproduce all the steps of the analysis. Use comments to explain what is going on. **RStudio** codebooks can easily turn scripts into (rudimentary) documents.
  - Better still, use **RStudio** notebooks or Rmarkdown files for *both* documentation and code *together*.

- *Work portably*. Make sure you can send the *entire* **RStudio** *project directory* to another system and have it work *seamlessly and without change*.
- *Do not use absolute file names in scripts!* Your file names should be relative to the working directory.
- *NEVER* use `setwd()`, especially not in scripts!
- Keep your global environment clean and uncluttered.
- Do not rely on saved items *in the workspace* from one session to the next.
- Use the `SOAR` package judiciously for temporary transfers between sessions.

1.4 Use the file system

In working with **R**, *use the file system*. A good protocol is

- For each new project, set up a *working directory* which will contain all the files needed for that project in one place.
- The working directory may contain sub-directories for natural entities such as `data`, `fig`, `archive`, `scripts`, `results`, &c
  You can do a lot of file management from within **R** itself, usually vie the console rather than in scripts. E.g.

```r
if(!dir.exists("data")) dir.create("data")
file.copy("myData.csv", "data/")
file.remove("myData.csv")  ## now that a saved copy is available in data/
```

- Start **R** *in* the working directory via the **RStudio** tag file rather than start elsewhere use the GUI or *`setwd()`* to go there. (The latter is non-portable, of course!)

2 The search path and how to use it

- The R session works by finding *objects* and *evaluating* them.
- *Objects* are found by looking down the *search path*, set up initially by R and modified during the session by functions such as `library`, `require`, `detach`, `SOAR::Attach`, &c.
- Two ways to keep an eye on the search path:
  - Using the button in **RStudio**.
  - From the command line. E.g.

```
search() %>% noquote()                ## no "quotation" marks, please!
 [1] .GlobalEnv           .R_Cache            .Robjects
 [4] package:WWRCourse    package:WWRUtilities package:WWRGraphics
 [7] package:WWRData      package:forcats     package:stringr
....
[25] package:stats        package:graphics    package:grDevices
[28] package:utils        package:datasets    package:methods
[31] Autoloads            package:base
```

– You can see what is in any position of the search path using *ls*:

```
ls("package:WWRUtilities") %>% noquote()
 [1] bc                 bc_inv             box_cox
 [4] bs                 default_test       dropterm
 [7] eigen2             fractions          GIC
....
[43] var_c              vcovx              which_tri
[46] width.SJ           zq                 zr
[49] zs                 zu
```

– You can also find where things are on the search path using *find*:

```
pi <- 22/7                              ## good enough for government purposes
find("pi", numeric = TRUE)              ## gives position in search path
  .GlobalEnv package:base
           1           32
print(c(pi, base::pi), digits = 20)
[1] 3.1428571428571427937 3.1415926535897931160
```

It is the *first* occurrence that is selected by the **R** interpreter. For safety use a package qualifier: `base::pi`, which works *whether or not* the package itself is currently on the search path.

2.1 Modifying the search path (and ways to avoid it)

An example

```
search()[1:3]

[1] ".GlobalEnv" ".R_Cache"   ".Robjects"

myList <- list(x = 3, y = rnorm(10000), z = TRUE)
attach(myList)
ls()

[1] "myList" "pi"

search()[1:4]

[1] ".GlobalEnv" "myList"      ".R_Cache"   ".Robjects"

x <- 4
find("x")

[1] ".GlobalEnv" "myList"

rm(x)
x

[1] 3
```

An example, continued.

```
detach("myList")
search()[1:3]

[1] ".GlobalEnv" ".R_Cache"   ".Robjects"

x

Error in eval(expr, envir, enclos):  object 'x' not found

ls()

[1] "myList" "pi"

myList$x

[1] 3
```

- *NEVER* use `attach()`/`detach()` in this way!
- The tools `with()` and `within()` provide simpler, clearer and safer ways of achieving the same result.

2.1.1 `with` and `within`

- Both take two arguments:
  - a `list` (or `data.frame`) and
  - an *`expression` to be evaluated.*
- The *`expression` is evaluated* with the components of the list used as variables ahead of the search path, including the global environment.
- *In the case of* `with(list, expr)` *the result is the* value of the expression.
- *In the case of* `within(list, expr)` *the result is* a modified list, (usually a data frame) *with the expression defining the changes.*
- *In both cases the* `expr` *is usually a sequence enclosed in braces,* `{ ... }`

In addition to safety, both provide elegant ways of avoiding “dollar clutter” in your code and hence making it much easier to read and to maintain.

Examples:

```
find("quine"); head(quine, 2)

[1] "package:WWRData"
  Eth Sex Age Lrn Days
1   A   M  F0  SL    2
2   A   M  F0  SL   11

##
## compare:
tapply(quine$Days, list(quine$Sex, quine$Age), mean)

        F0       F1       F2       F3
F 18.70000 12.96875 18.42105 14.00000
M 12.58824  7.00000 23.42857 27.21429

##
## with using with()..
with(quine, {
  tapply(Days, list(Sex, Age), mean)
})

        F0       F1       F2       F3
F 18.70000 12.96875 18.42105 14.00000
M 12.58824  7.00000 23.42857 27.21429
```

Examples (continued)

```
quine_ext <- within(quine, {
  LSE          <- Lrn:Sex:Eth
  levels(Age) <- paste0("Form", 0:3)
  Means        <- ave(Days, LSE, Age, FUN = mean)
  Residuals    <- Days - Means
}) %>% select(Eth:Lrn, LSE, Days, Means, Residuals)  ## pipes will occur a lot!
head(quine_ext, 3)
```

```
  Eth Sex   Age Lrn    LSE Days Means Residuals
1   A   M Form0  SL SL:M:A    2     9        -7
2   A   M Form0  SL SL:M:A   11     9         2
3   A   M Form0  SL SL:M:A   14     9         5
```

```
with(quine_ext, {
  tapply(Days, list(Age, LSE), median)
})
```

```
      AL:F:A AL:F:N AL:M:A AL:M:N SL:F:A SL:F:N SL:M:A SL:M:N
Form0   17.5   15.5   13.0    4.5    3.0     25   11.0     17
Form1   11.0    6.5   10.5    3.5   18.0      5    6.0      5
Form2    2.0    1.0   17.0    7.0   33.5      5   42.5     30
Form3   10.0   12.0   28.0   27.0     NA     NA     NA     NA
```

2.2 The `SOAR` package, (now included within `WWRCourse`)

- Use for keeping objects from one session to the next. Especially useful for *large* objects, or objects requiring a lot of time to generate.
- Keeps `.RData` files of stores objects in a sub-directory of the working directory, `./.R_Cache`.
- Four key functions:
  - *`Store(...)`*: place objects in cache, removing from memory, but still visible as *promises*,
  - *`Objects()`* (or *`Ls()`*): list cache contents,
  - *`Attach()`*: place the cache on the search path as promises
  - *`Remove(...)`*: delete objects from the cache, permanently.
- An additions function, *`Search()`*, gives and enhanced view of the current search path.

- In this workshop we work with two cache folders,
  - `.R_Cache` for temporaries and
  - `.Robjects` for initial supply and more permanent carryover.
- From the session initialisation script:

```r
Attach(lib = .Robjects)
                     ## check for duplicate dataset copies
._objects <- intersect(ls(".Robjects"), ## could have used Ls(lib=.Robjects)
                       ls("package:WWRCourse"))
if(length(._objects) > 0) {
  Remove(list = ._objects, lib = .Robjects)  ## remove duplicates
}
rm(._objects)        ## a hidden object, seen only by ls(all = TRUE)

Attach()             ## the .R_Cache folder, same as Attach(lib = .R_Cache)
Remove(list = Ls())  ## clean up working .R_Cache,
rm(list = ls())      ## clean up workspace ... for now.
```

3 Objects and working with them

"*Everything* in **R** is an object,
Everything you do in **R** is a function call.
John M. Chambers, c. 2005

- Two broad kinds of object:

  Atomic Five main types *numeric*, *character*, *logical*, *complex* and *raw*. (A sixth type, *integer* is now commonly recognized, but for most purposes can be regarded as *numeric*.)

  Recursive Main ones *list* (including *data.frame*s), *language* (such as *formula*s, *expression*s and *function*s) and *environment*s

- Objects may carry additional information as *attribute*s. E.g. *length*, *names*, *dim*, *dimnames*, *levels* and, most importantly *class*.

- In this sense, objects are *self-describing*, (or the *should be!*)

- All objects have *components*, which may be accessed by operators (functions) such as `[`, `[[`, `$`.
  - Atomic objects are characterised by having all their components of the one (atomic) kind.
  - Recursive objects can have components of different kinds; in the case of lists and environments these can be of any kind whatever, (including lists and environments - hence the name "recursive").
- Atomic objects are the "building blocks" of the objects we usually work with, such as *data objects*, *fitted model objects*, *graphical objects* and even *language objects*.
- The `class` attribute of an object, determines how *generic functions* behave when presented with them as an argument.

  These include many familiar actions such as printing, plotting, summarizing, &c, as well as the generic functions you write yourself for new actions of a generic kind in your work.

Some simpler examples:

```
(x <- structure(1:20, names = letters[1:20]))

 a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

## compare (names attribute retained)
x[5]

e
5

## with (names attribute discarded)
x[[5]]

[1] 5

## give it a class
class(x) <- "shuffle"; x

 a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
attr(,"class")
[1] "shuffle"

## now provide a pring method for such objects
```

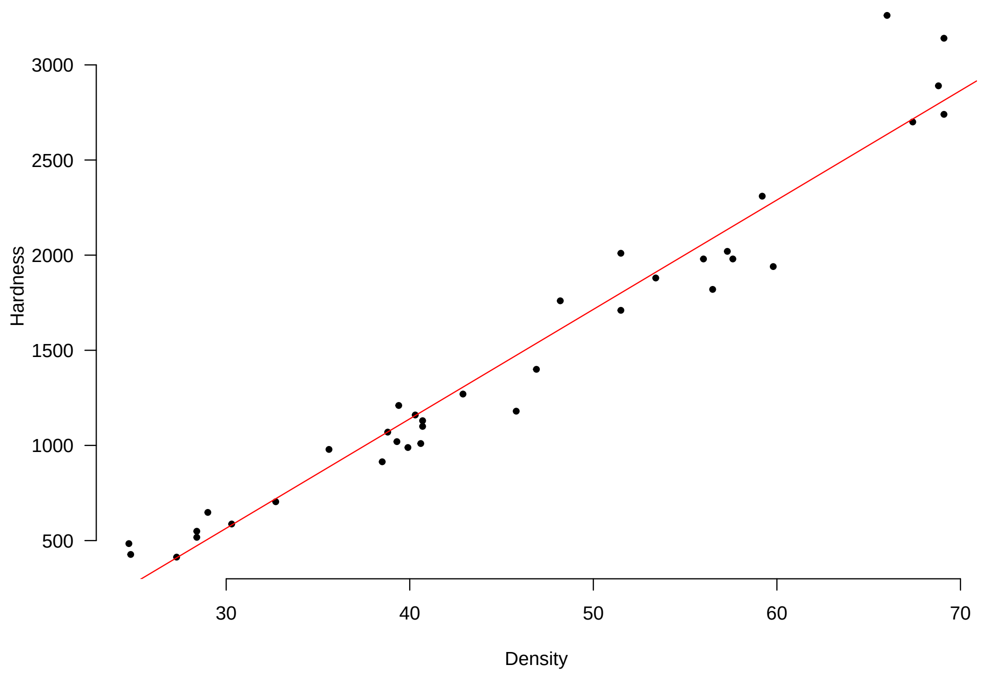
```
print  ## the generic function in the base package

function (x, ...)
UseMethod("print")
<bytecode: 0x556f8c29d330>
<environment: namespace:base>

print.shuffle <- function(x, ...) {
  cat("This shuffle is:\n")
  print.default(sample(x))
  invisible(x)
}
x            ## implicit call to print(x)

This shuffle is:
 l  b  k  h  j  d  n  t  a  o  q  e  s  r  i  f  g  p  m  c
12  2 11  8 10  4 14 20  1 15 17  5 19 18  9  6  7 16 13  3

x            ## implicit call to print(x) again

This shuffle is:
 l  c  a  e  h  o  j  m  q  n  r  p  k  s  g  t  b  d  f  i
12  3  1  5  8 15 10 13 17 14 18 16 11 19  7 20  2  4  6  9
```

More realistic example. Some bad practices, and why

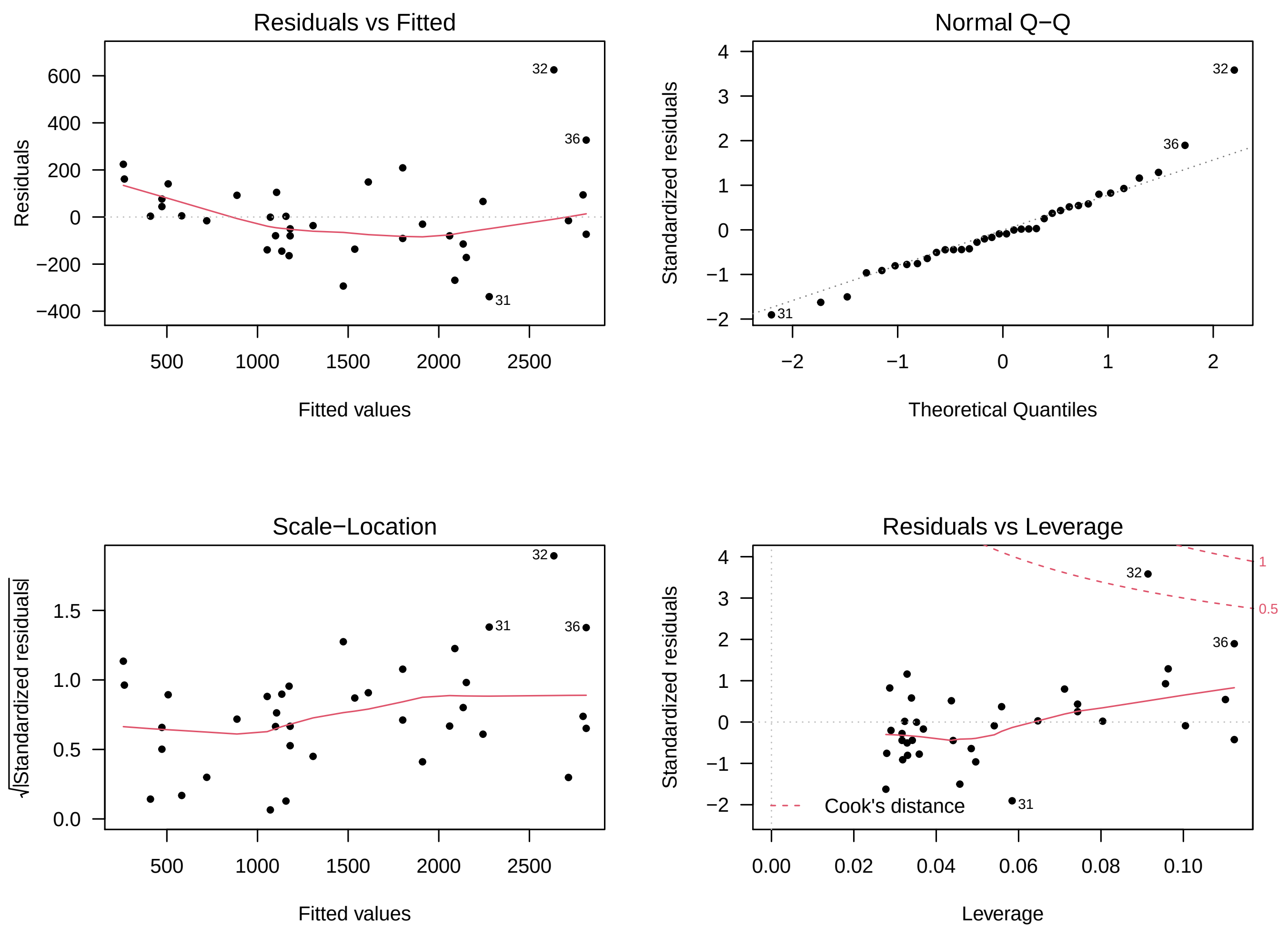
```
find("janka")

[1] "package:WWRData"

# head(janka)
plot(Hardness ~ Density, janka, pch=20, bty="n") ## plot with formula
fm <- lm(janka$Hardness ~ janka$Density)  ## poor. Cannot predict from fm
fm <- with(janka, lm(Hardness ~ Density)) ## bad. Object not fully self-describing
fm <- lm(Hardness ~ Density, data = janka)## good. Self-describing and prediction O
class(fm)

[1] "lm"

methods(class = class(fm))

 [1] add1           alias          anova          box_cox
 [5] case.names     coerce         confint        cooks.distance
 [9] default_test   deviance       dfbeta         dfbetas
....
[41] show           simulate       slotsFromS3    summary
[45] variable.names vcov           xtable
see '?methods' for accessing help and source code

summary(fm)$coefficients
```

```
               Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -1160.49970 108.579605 -10.68801 2.065919e-12
Density        57.50667   2.278534  25.23845 1.332735e-23
```

```r
abline(coef(fm), col = "red")
```



```r
par(mfrow = c(2,2))
plot(fm)    ## call to the "lm" method function of plot()
```



Residuals vs Fitted
Residuals
Fitted values
600
400
200
0
−200
−400
500
1000
1500
2000
2500
32
36
31
Normal Q−Q
Standardized residuals
Theoretical Quantiles
4
3
2
1
0
−1
−2
32
36
31
Scale−Location
√|Standardized residuals|
Fitted values
1.5
1.0
0.5
0.0
32
31
36
Residuals vs Leverage
Standardized residuals
Leverage
0.00
0.02
0.04
0.06
0.08
0.10
Cook's distance
1
0.5
32
36
31

4 Course packages

The course materials includes four *interdependent* source packages.

WWRData Is a pure data package containing the data sets used, and more.

WWRGraphics Is a collection of functions to provide various enhancements to traditional (base) graphics.

WWRUtilities Is a collection of functions providing some extension to, and enhancement of, the modelling tools in the `MASS` package. It may be useful in its own right after the workshop.

WWRCours Is a collection of functions and other materials to be used in the workshop and unlikely to be useful afterwards.

Installation These packages are to be installed *from source*. This can be done from **RStudio** from `Packages -> Install` or from the console.

It is important they be installed in order: First *WWRData*, then *WWRGraphics*, then *WWRUtilities* and finally *WWRCours*.

5 Package structure and installation

Construction Packages are constructed as `folders` *(aka* `directories`*) and on construction have a particular structure:*

- *The top level has two plain files called* `DESCRIPTION` *and* `NAMESPACE`*,*
- *There are several subdirectories present initially, including* `R`*,* `man`*,*
- *Other subdirectories with special roles include* `src`*,* `data`*,* `vignettes` *and* `inst`*.*

Building *When a package is* built, *(by* **R***):*

- *a series of basic checks are made, vignettes, if any, are built the result is placed in a folder* `inst/doc`*, created on the fly, if necessary.*
- *The result is not a changed folder, but rather a compressed* `.tar` *file, ("tape archive"!) with a name encoding the package name the version numbers. E.g.* `WWRData_0.1.0.tar.gz`*.*

- *To recover the (slightly modified) construction folder from such a file use*

  `tar zxvf WWRData_0.1.0.tar.gz`

  on the command line in a suitable shell (provided you have the tools installed, of course!).

Installation *The process of installing a built package decompresses and expands the* `.tar.gz` *file as a package folder in your package tree (i.e. your library), but*

- *The process causes further changes that are irreversible:*
  - *Code files are byte-compiled and stored as binaries, as well as with all the data, help, &c.* `html` *help versions are created.*
  - *The* `inst` *folder is removed, but anything inside it, usually* `doc` *and* `extdata` *subfolders, are raised to the top level of the installed package folder,* verbatim. *(I.e. the checking and modification processes are bypassed.)*
  - *Any unneeded files or folders, including* `data-raw`*, are discarded.*

- *To install a built package from the command line use*

  ```
  R CMD INSTALL WWRData_0.1.0.tar.gz
  ```

  To install a built package from within **R** *itself use*

  ```
  install.packages("WWRData_0.1.0.tar.gz", repos = NULL)
  ```

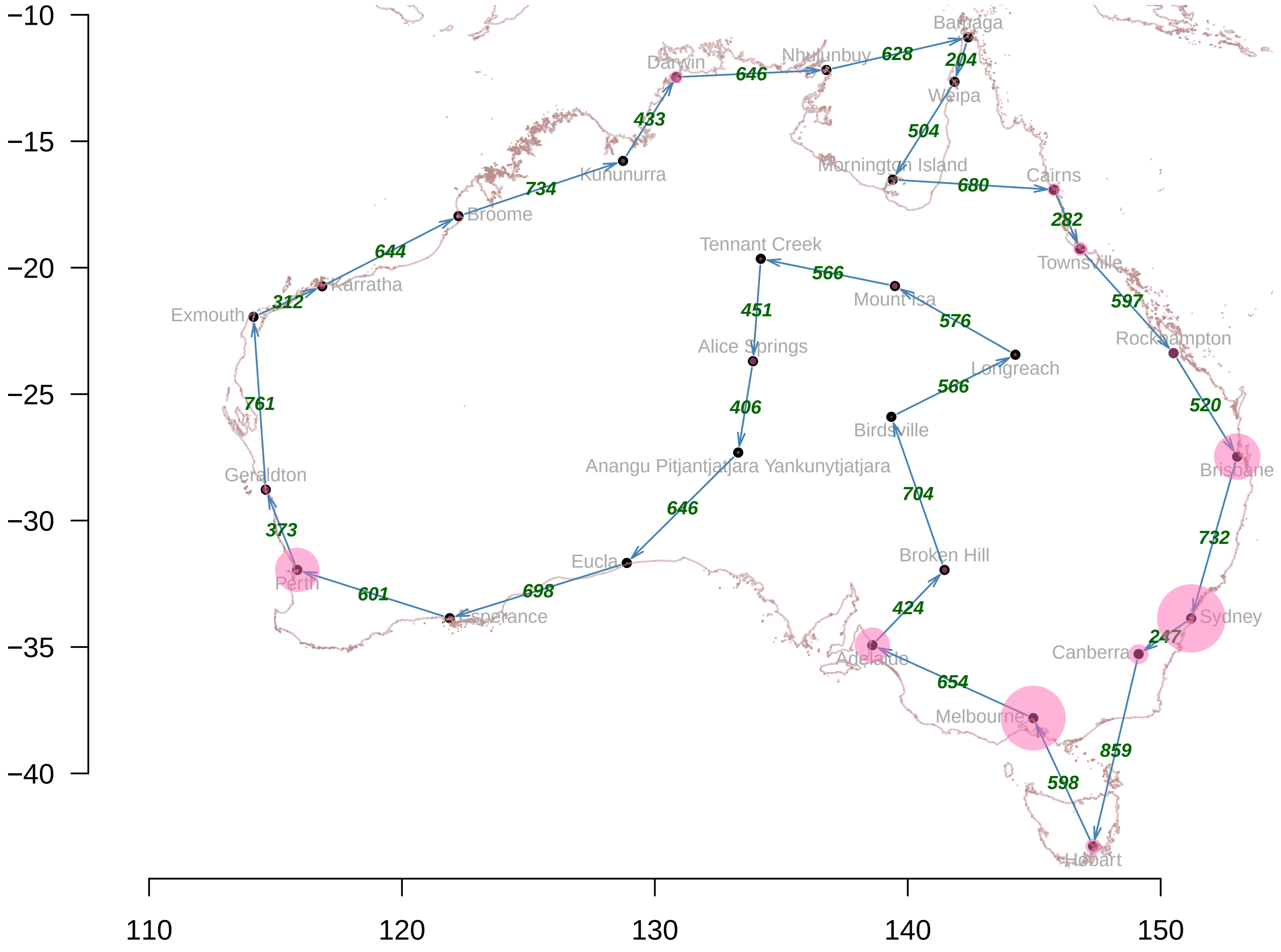
  In both cases you need to have the `.tar.gz` *file in your working directory. (Alternatively you could provide the file path to it.)*
- *If packages are installed manually like this, it is important to have any dependencies installed beforehand.*

  E.g. `WWRCourse` *depends on* `WWRUtilities`*, which itself depends on* `WWRGraphics`*, so they must be installed in this reverse order.*

  Note that there are further CRAN dependencies for all of them, too, so these need to be installed before you start.

Example: An Australian round trip

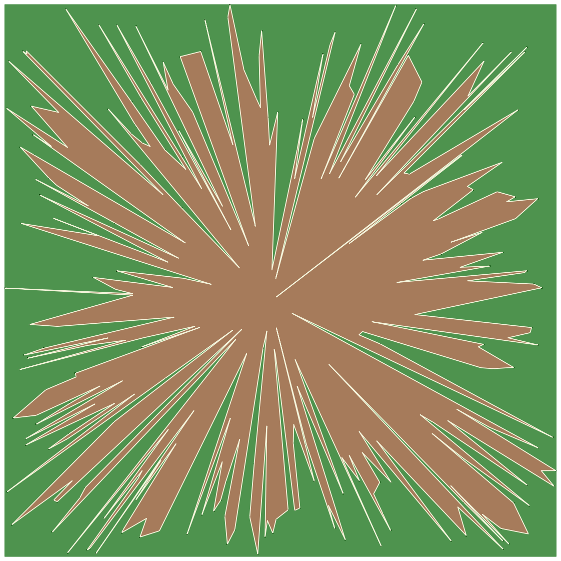
```r
library(WWRGraphics)
brownish <- alpha("rosy brown", 0.5)
pinkish <- alpha("hot pink", 0.5)
z <- with(roundTrip,
          complex(real = Longitude, imaginary = Latitude) %>%
            setNames(Locality))
plot(z, asp = 1, ann = FALSE, bty = "n", pch = 20)
lines(Oz, col = brownish)
text(z, names(z), cex = 0.7, pos = avoid(z), offset = 0.25, col="dark grey")
arrows(z, col = "steel blue", gap = 1, circular = TRUE, length = 2)
dists <- gcd_km(z, cyc(z)) %>% round()
text((z + cyc(z))/2, dists, col = "dark green", cex = 0.7, font = 4)
##
## add city sizes
##
circles(Latitude ~ Longitude, roundTrip, radii = sqrt(Population),
       fill = pinkish, colour = pinkish, maxradius = 0.5)
```



−10
−15
−20
−25
−30
−35
−40
110
120
130
140
150
Bamaga
Nhulunbuy
Darwin
Weipa
Mornington Island
Cairns
Kununurra
Broome
Tennant Creek
Townsville
Mount Isa
Karratha
Exmouth
Alice Springs
Rockhampton
Longreach
Birdsville
Anangu Pitjantjatjara Yankunytjatjara
Brisbane
Geraldton
Broken Hill
Eucla
Perth
Esperance
Sydney
Adelaide
Canberra
Melbourne
Hobart
628
204
646
433
504
680
734
282
644
566
451
576
312
597
566
761
406
520
704
646
373
732
698
601
424
247
654
859
598

Example: A random exploding polygon.

```r
par(mar = rep(0, 4))
set.seed(20200211)
z <- complex(real = runif(300), imaginary = runif(300))
z <- z[order(Arg(z - mean(z)))]
plot(z, asp = 1, ann = FALSE, axes = FALSE, pch = ".",
     xlim = 0:1, ylim = 0:1)
greenish <- alpha("dark green", 0.7)
rect(0, 0, 1, 1, fill = greenish, colour = greenish)
polygon(z, fill = getColors("French beige"), colour = "beige")
```



Some useful books:

Chang, W. (2019). *R Graphics Cookbook*. O'Reilly. https://r-graphics.org/.

Jones, O., R. Maillardet, and A. Robinson (2020). *Introduction to Scientific Programming and Simulation Using R* (2nd Edition ed.). CRC Press.

Matloff, N. (2011). *The Art of R Programming*. San Francisco: No Starch Press.

Murrell, P. (2011). *R Graphics* (2nd ed.). Boca Raton: CRC Press.

Wichham, H. (2019). *Advanced R* (Second ed.). CRC Press. https://adv-r.hadley.nz/.

Wickham, H. (2016). `ggplot2`*: Elegant Graphics for Data Analysis* (Second ed.). UseR! New York: Springer-Verlag. https://ggplot2-book.org/.

Session information

Date: 2021-01-29

- R version 4.0.3 (2020-10-10), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 20.04.1 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: dplyr 1.0.3, english 1.2-5, forcats 0.5.1, ggplot2 3.3.3, ggthemes 4.2.4, gridExtra 2.3, knitr 1.31, lattice 0.20-41, patchwork 1.1.1, purrr 0.3.4, readr 1.4.0, scales 1.1.1, stringr 1.4.0, tibble 3.0.5, tidyr 1.1.2, tidyverse 1.3.0, WWRCourse 0.2.3, WWRData 0.1.0, WWRGraphics 0.1.2, WWRUtilities 0.1.2, xtable 1.8-4
- Loaded via a namespace (and not attached): assertthat 0.2.1, backports 1.2.1, broom 0.7.3, cellranger 1.1.0, cli 2.2.0, colorspace 2.0-0, compiler 4.0.3, crayon 1.3.4, DBI 1.1.1, dbplyr 2.0.0, ellipsis 0.3.1, evaluate 0.14, fansi 0.4.2, farver 2.0.3, fractional 0.1.3, fs 1.5.0, generics 0.1.0, glue 1.4.2, grid 4.0.3, gtable 0.3.0, haven 2.3.1, highr 0.8, hms 1.0.0, httr 1.4.2, iterators 1.0.13, jsonlite 1.7.2, lazyData 1.1.0, lifecycle 0.2.0, lubridate 1.7.9.2, magrittr 2.0.1, MASS 7.3-53, modelr 0.1.8, munsell 0.5.0, parallel 4.0.3, PBSmapping 2.73.0, pillar 1.4.7, pkgconfig 2.0.3, R6 2.5.0, randomForest 4.6-14, Rcpp 1.0.6, readxl 1.3.1, reprex 1.0.0, rlang 0.4.10, rpart 4.1-15, rstudioapi 0.13, rvest 0.3.6, SOAR 0.99-11, splines 4.0.3, stringi 1.5.3, tidyselect 1.1.0, tools 4.0.3, vctrs 0.3.6, withr 2.4.1, xfun 0.20, xml2 1.3.2