Learning *R*

Carl James Schwarz

StatMathComp Consulting by Schwarz
cschwarz.stat.sfu.ca @ gmail.com

Final Summary

R Summary

R is powerful and versatile.

R is free, but not cheap.

- *R* has a steep learning curve.
- *R* is not consistent in usage and syntax.
- *R* creates nice graphics, but poor at textual output (e.g. nicely formatted tables are tedious to construct)
- Packages have NO quality control.
- Requires a fair degree of statistical sophistication.
  - E.g. *anova()* function gives Type I rather than Type III SS, F-tests without warning!

R Summary

Work flow using *Rstudio*.

- Launch *Rstudio*.
- Navigate to directory with scripts and data.
- SET WORKING DIRECTORY!
  Check the console pane to see if done properly.
- Open the Script
- Highlight and Run.
- Create HTML notebook at end to ensure that script works properly.

R Summary - Data Structures and Objects

R has a rich set of data structures (special case of objects):

- **vectors** - an ordered collection of the same data type (e.g. numbers, characters, logicals, etc.)
- matrix - a two-dimensional collection of the same data type.
- array - a 2+ dimensional collection of the same data type.
- **dataframe** - collection of vectors (of same length) but vectors can be different data types.
- **list** - an arbitrary collection of objects (including lists).

Other objects:

- **function** - contains a list of instructions
- expressions - fragments of *R* code or formulae

R Summary - Object Types

R has several object types

- numbers (integer, real, or complex)
- characters ("abc")
- logical (TRUE or FALSE)
- Date, DateTime
- factor (CAUTION) of any type - index to a set of values
  - use *stringsAsFactors=FALSE* on all *read.csv()* operations.
  - use *stringsAsFactors=FALSE* on all *data.frame()* operations.
  - explicitly create factors for all categorical variables using *df$varF = factor(df$var)*.
  - distinguish between *size=var* or *size=varF* in *ggplot()*.
  - may need to order factors to sort levels on a graph.

Missing values *NA* are different from *Inf*, " ", 0, *NaN* etc.

The *str()* function is YOUR FRIEND!

R Summary - Accessing values

R has several ways to access values (see reference card)

- vectors
  - v[k] - selects the k^{th} item
  - v[-k] - all but the k^{th} item
  - v[1:4], v[c(1,3,5)], v[-(1:k)] - sets of values
  - v["name"], v[c("name1","name2")] - select by named component
  - v[c(TRUE, FALSE, TRUE)], v[v>3], v[v == 24] - use LOGICAL vector to select items
- data frames
  - dafr[,k], dafr[k,] - all of k^{th} column/row
  - dafr$name1, dafr[,"name1"], dafr[,c("name1","name2")] - select columns by names
  - dafr[dafr$v>10, c("v2","v3")] - subsets of the data frame
- lists
  - mylist$name - select a named component of the list
  - mylist[["name"]] - element with certain name
  - mylist[1] vs. mylist[[1]] - CAREFUL

R Summary - Input

R is a bit clumsy.

- *read.table()*, *read.csv()*, *read.delim()*, *readxl::read_excel()*
  - *header=TRUE* - assumes variable names in first row
  - *as.is=TRUE* or *stringsAsFactors=FALSE* - don't convert character string to factors - RECOMMENDED
  - *strip.white=TRUE* - remove extra white space when ever possible - CAUTION of leading blanks in character strings
- *scan()* - a more general way to read files with odd organizations
- Several packages for access to database systems - see reference card.
- Define categorical variables as factor by making new variable (*df$varF <- factor(df$var)* or replacing variable (*df$var <- factor(df$var)*) after creating data frame.

Try and vectorize.
Most of time you can avoid *if()* and *for()* control structures.
R will often cycle through shorter arguments

```
1  v <- 15:18
2  v + 3  # adds 3 to every value
3  v + c(2,3) # cycles through the (2,3) pair
4
5  v[ v==10]  <- 15  # replaces for/if structure
6
7  Fatal <- c("no","yes")[1+ (Severity==1)]
```

recode() in *car* package is useful

R Summary - Output

R is a bit clumsy.

- *sink("filename", split=TRUE)* ... *sink()* - text output.
- *write.table()*, *write.csv()*, *write.xlsx()*- create output files.
- *ggsave* - graphical output. Don't forget *h=*, *w=*, *units=*, and *dpi=* arguments.
- *Rstudio* with the html notebook.
- *Rmarkdown* - combined documents with text, programming, and output..
- *Sweave* - LaTeXand *R* integrated together to create complete document that is publication quality.

R Summary - Packages

Many user-written extensions to *R*; but no quality control.

- Create a personal library for packages on your computer as this makes it easier to update on a regular basis.
- Load library prior to first use using the *library()* function.
- Very difficult to detach a package once it is loaded
- Beware of name conflicts, i.e. several packages with the same name for different objects. Use *package::function()* to ensure that correct function is used.

R Summary - Functions

Make a collection of analyses for reuse.
Scripts and *source()* can serve a similar functionality.

```
1  myfunction <- function(arg1, arg2, arg3=defaultvalue) {
2  #  Comment describing the arguments and purpose of funct
3       arg1[3] <- new value # ok see below
4       ...
5       myresults <- ....
6       return(myresults)  # don't forget
7     } # don't forget
```

- Arguments can be any data structure or type
- Return can be any data structure or type (most commonly a list or a vector)
- Arguments are call-by-name, i.e copies passed.
- New variables are local only
- AVOID SIDE EFFECTS IN A FUNCTION
- *browser()*, *trace()* - useful for debugging

Scripts and *source()* can serve a similar functionality.

```
1  source("MyDocuments/MyStuff/myRfunction.r")
2
3  # now you can use the functions you defined
4  # in myRfunction.r
```

R Summary - Useful Builtin Functions - I

Base *R* has many useful functions; packages provide more

- *c(...)* - combine arguments into a vector
- *seq()*, a:b - generate sequeces
- *is.na()* - tests for missing values – see other is.xx functions
- *str()* - show structure of an object
- *nrow()*, *ncol()* - number of rows and columns
- *match(x,y)*, x %in% y - which values of *x* are in *y*
- *unique()* - return unique values of object
- *xtabs()* - cross tabulations - useful for check recodes, etc – include the NAs
- *reshape()* - interchange between wide and long formats - documentation sucks

R Summary - Useful Builtin Functions - II

- *cbind()*, *rbind()* - paste together columns and rows
- *split()*, *stack()* - split and stack dataframes
- Split-apply-combine paradigm - RECOMMEND *plyr* package rather than Base *R* functions, esp. the *summarize* usage.

```
ddply(cereal, "shelf", summarize,
      mean.cal = mean(calories))

ddply(cereal, "shelf",  function(x){
      ncereals <- nrow(x)
      fit <- lm( Calories ~ Fat, data=x)
      mycoef <- coef(fit)
      res <- data.frame(ncereals, mycoef, stringsAsFactor
      return(res)
    })
```

R Summary - Useful Builtin Functions - III

- Usual math functions.
  - CAUTION between *min()* and *pmin()*
- Usual statistical functions
  - NAs propagate, so many functions have na.rm=TRUE
  - *lm()* - basic linear models (e.g. simple ANOVA and regression)
  - *glm()* - generalized linear models (e.g. logistic regression)
  - *lmer()* - linear models with mixed effects (e.g. split-plot designs)
  - Use methods (specialized functions) to extract information from output
  - See `http://www.stat.sfu.ca/~cschwarz/CourseNotes`

R Summary - Useful Builtin Functions - IV

Base *R* is a bit clumsy with dates and times.

- *as.Date()* to convert to internal format (# of days since origin)
- *as.POSIXct()* to convert to constant date-time (avoid POSIXlt() unless really needed)
- format='%m/%d/%Y' to convert from external to internal and out to external formats
  - CAUTION - when converting from dates/datetimes, the "%xx" gives CHARACTERS, not numbers

The *lubridate* package will make your life much easier.
Other packages useful for duration data (*hms*) or clock data (*psych*)

R Summary - Useful Builtin Functions - V

Dealing with character strings

- *paste()* - combines strings, numbers, etc into a single string
- *substr()* - extracts substrings - CAUTION of syntax
- *grep()* - matching of patterns - CAUTION - complex syntax
- *stringer* package is easier to use in many cases

R Summary - Plotting

R has extensive facilities for plotting

- Base *R* - pen-on-paper paradigm AVOID
- *Lattice* graphics - plot objects - AVOID
- *ggplot2* package - grammar of graphs - RECOMMENDED
- *Shiny* package - visualization (interactive) plots - RECOMMENDED
  - Build a graph using various layers
  - Adjust final graph when done with axes etc
  - CAUTION: Don't forget to print final object created

R Summary - Shiny

A quick way to bring interactivity to your applications.

- Start small and build up.
- For large datasets/applications, it may be difficult to debug
- Lots of addons, e.g. *leaflet*
- Why are you making a *Shiny* app?

R Summary - Spatial I

Whew!

- Use the *sf*, *sp* and *raster* packages.
- Are you trying to use *R* as a GIS? It may be very slow with large databases and many layers.
- Start small and work your way up.

Becoming an Rexpert.

- Never assume that the data is in a particular order.
  - Never use *cbind()*; use *merge()*
  - Never select particular rows. Use a selection vector to select rows of interest.
- Never assume columns are in a particular order.
  - Never refer to columns by number, i.e. do not use *df[, 2:3]*
  - Refer to columns by names or by selection vector
- Seldom need to use series of *ifelse()*.
  - Use *car::recode* or do table lookups using *merge()*
  - Check your recodes using *xtabs(Old+new, data=...* or a *ggplot*

R language elements - Summary - Advanced II

- NO FOR LOOPS!
  - *for()* implies that results of one iteration depend on results of previous iterations. This is seldom true except in MCMC situations.
  - Use *plyr* or *dplyr* packages or equivalents
  - Use these packages in simulation studies as they parallelize naturally
- Be careful of time zone.
  - Do you really want the instant (time-stamp) or do you just want dates+time (use UTC as timezone)
- Don't hard code *setwd()* in code. Use relative file names.
  - Rely on person setting the working directlory
  - Use *file.path()* to avoid the different file system naming conventions (slashes vs colons, etc.)

R language elements - Summary - Advanced III

- Worry about the impacts of missing values.
  - Compare *df[df$x==7,]* vs. *df[df$x==7 & !is.na(df$x),]*
  - Do your functions deal nicely with missing values (e.g. use *na.rm=TRUE*
- All data in data frames or tibbles.
  - Do not store data in individual vectors unless they are temporary selection. vectors
- Data.frame vs. tibble differences.
  - *xf[, "x", drop=FALSE]* vs. *xf[,"x"]* varies depending if df is a data frame or a tibble (groan()
  - If selecting a variable number of columns, what do you want to happen if you select only one column?
- Functions should be self contained and have no side effects.
  - All data should be passed to functions.
  - Do not rely on global variables.

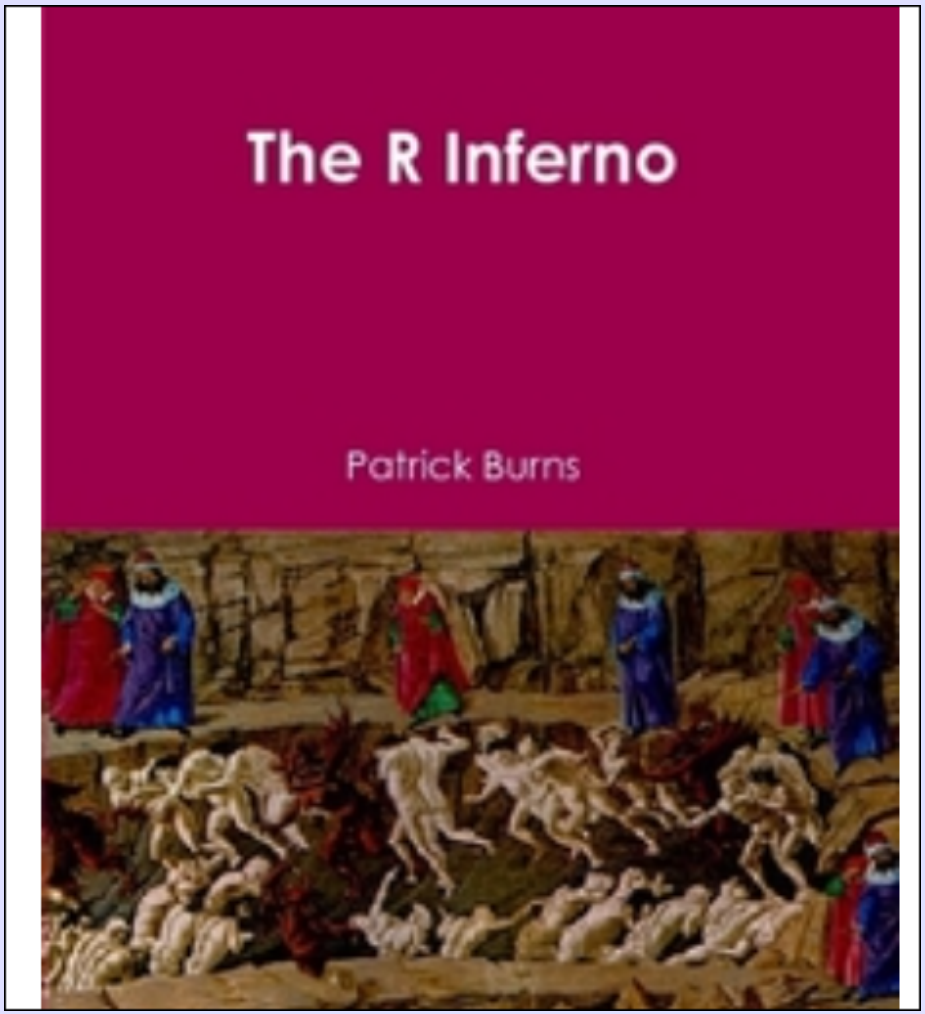
R language elements - Summary - Advanced IV

- Qualify functions from packages, i.e. *package::function()(*
  - Particularly true if using the *dply* and *dplyr* packages and summarize
- Do not hard code stuff.
  - Use functions (e.g. *med()* rather than hardcoding the actual value of the median
  - Create a variable at top of script that is used, e.g. *year.to.analyze <- 2018*, *alpha<- 0.05*
- *grep()*, *regexpr()* are your friends!
  - *select.rows <- grepl("abcd', df$name)* followed by *df[selec.rowst,]*
  - *select.col <- names(df)[grep("abc", names(df))]* selects certain columns followed by *x[,select.col]*

- If using MSWord, build tables to as close as possible and then cut and paste

```
report <- ...
temp <- report
temp[, 3:3] <- round( temp[,2:3],2)
write.csv(temp, file.path=(...))
```

R Summary - The *R* Inferno



If you are using *R* and you think you're in hell, this is a map for you. A book about trouble spots, oddities, traps, glitches in *R*. Even if it doesn't help you with your problem, it might amuse you (and hence distract you from your sorrow).

`http://www.burns-stat.com/documents/books/the-r-inferno/`

To err is human,
but it really takes a computer to screw things up!

R is free, but not cheap.

cschwarz.stat.sfu.ca@gmail.com