Introduction to R

Intensive Statistics Course

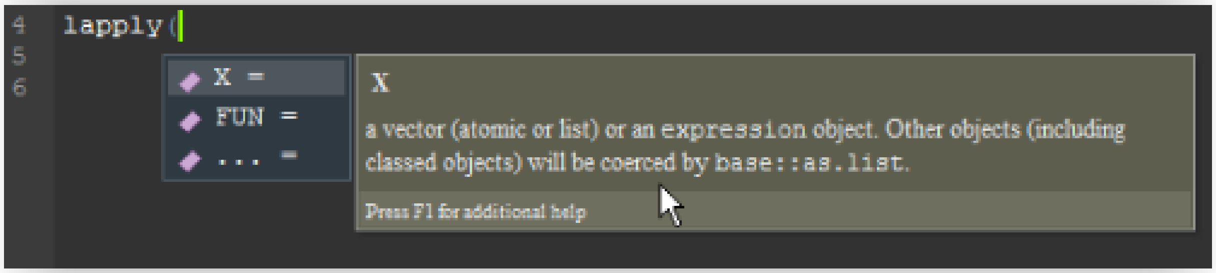
Wihan Marais

wihanmarais@sun.ac.za | github.com/WihanZA

25 January 2022

Why R?

1. R is free and open-source.
    - At the time of writing, a new Stata 17 annual license is priced between R11,670 and R21,280 excluding VAT.
    - Free upgrades, updates and dissemination.
    - Availability of helpful resources like **stackoverflow**.
2. R uses **packages**
    - R consists of Base-R coupled with third-party libraries of pre-written code, or packages.
    - **CRAN**, or The Comprehensive R Archive Network, is a network of ftp (file transfer protocol) and web servers around the world that store identical, up-to-date, versions of code and documentation for R.
    - More on this later.
3. R uses predictive coding (Ctrl/Cmd + Space is very useful).



```
lapply(
```

X

a vector (atomic or list) or an expression object. Other objects (including classed objects) will be coerced by base::as.list.

Press F1 for additional help

4. R is compatible with Markdown.
    - These lecture notes were created as a '.Rmd' file using **R Markdown**, RStudio's native authoring framework for data science.
    - See this 1-minute video summary of what R Markdown entails.

Before we start

You need the following installed on your machine:

- **R** or Base-R.

  ”R is a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.”

- **RStudio**

  ”RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.”

- **Rtools**

  Select the Rtools download link for the relevant version of R installed on your machine. To determine the version currently installed, run the following code in your console. First, highlight the line of code you would like to run and Ctrl/Cmd + Enter to run.

```
sessionInfo()[1]$R.version$version.string

# IMPORTANT:
# Take care to check the box to have the installer 'edit your path'
```

To verify that we have installed Rtools properly, we need to make use of the `devtools` package.

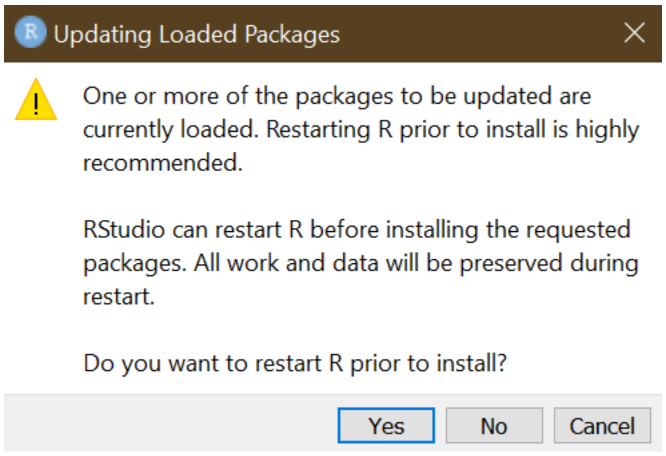
```
install.packages("devtools") # Install the package from CRAN.
library(devtools) # Load package into your current library.

find_rtools() # Run this command from the devtools package
# or
devtools::find_rtools()
# should return TRUE in your console
```

Packages

Install and load a few packages that you would likely use often. Let’s use the `installr` package as an example.

```
# Packages need only be installed once
install.packages("installr")
# and can be loaded into your library with
library(installr)
```

However, if I have already installed `installr` before, `install.packages()` produces the following error:


Updating Loaded Packages

One or more of the packages to be updated are currently loaded. Restarting R prior to install is highly recommended.

RStudio can restart R before installing the requested packages. All work and data will be preserved during restart.

Do you want to restart R prior to install?

It can be hard to keep track of all the packages that you have or have not installed on your machine. How, then, should we install packages?

```r
# In short: if package has not yet been installed, run code to install
if (!require(installr)) {
    install.packages("installr")
    require(installr)
}
```

Instead, I propose using the **pacman** package. It enables us to easily install new- and load old packages from curated lists such as CRAN, or any open-source package from **GitHub** using `p_load()` and `p_load_gh()`, respectively. These commands install packages if they have not yet been installed, and subsequently load them into our library.

```r
# Installs pacman from CRAN.
if (!require(pacman)) {
    install.packages("pacman")
    require(pacman)
}
# Load pacman into our library.
library(pacman)

# And finally...
pacman::p_load(installr)
```

Why did we load `installr` in the first place?

```r
# Are you using the latest version of R?
check.for.updates.R()
# Download and run the latest R Version.
install.R()
# Copy your packages to the newest R installation
copy.packages.between.libraries()
```

What are the packages we need to install and load?

```r
# From CRAN
pacman::p_load(fixest, tidyverse, huxtable, hrbrthemes, modelsummary, glue)

# From GitHub
pacman::p_load_gh("karthik/wesanderson", "BlakeRMills/MetBrewer")
                # "profile/repository name"
```

Basics

Projects

To improve your workflow in R, it is essential to work from a R project or directory.

`File > New project > New directory > New project > Choose directory location and name`

If you are planning on applying version control to your new project, it is useful to check `Create git repository`. Once you have created a project, its location will serve as your 'root folder' or reference directory for any future operations performed within this project. It is advisable to always reopen a R session by clicking the relevant `.Rproj` file, as it will keep track of your most current workspace and variable environment.

Workspace

When operating from a new project, you ought to observe a workspace divided into quadrants as summarised in Table 1. As with Stata, code can be typed and executed directly from your console (previously "Command"). Alternatively, this code can be stored in—and later relied upon—a script or `.R` file (like Stata do-file).

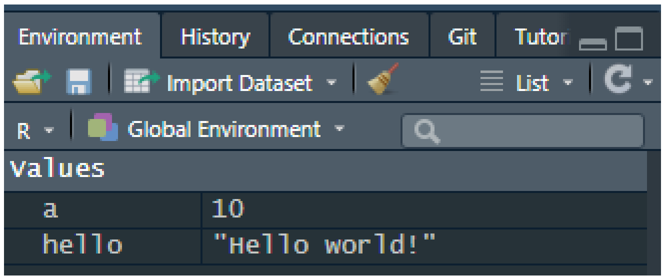
Table 1: Workspace

Location	**Function**
Top left	Script/R Markdown
Top right	Global environment
Bottom left	Console
Bottom right	Files/Plots/Packages/Help

R is an object-orientated language. Objects of various types (scalars, matrices, data frames, vectors, etc.) can be stored in memory for later use. Once named and saved, these objects will appear in your global environment. In R, we use the assignment operator `<-` to name and save objects (Tip: **Alt/Option** + **-**). For example:

```r
# object name <- value(s)
a <- 10
hello <- "Hello world!"
```

By highlighting and executing the relevant code (Ctrl/Cmd + Enter), `a` and `hello` should appear in your global environment like this:



```
Values
  a        10
  hello    "Hello world!"
```

We are able to report these variables in our console (or R Markdown output) by running the following:

```r
a
```

```
## [1] 10
```

```r
# or
print(hello)
```

```
## [1] "Hello world!"
```

```r
# or by using the glue package for something more fancy
glue::glue("I saved a variable which contains {hello} and I stored the number {a}.")
```

```
## I saved a variable which contains Hello world! and I stored the number 10.
```

Finally, your directories can be viewed to the bottom right, in addition to plot outputs, currently loaded packages, and help files. Should you ever require help or additional information regarding a specific command, add a `?` before that command and run the code. For example:

```r
?glue::glue()
```

Arrays

In R, an "array" object is basically a vector of values in the form of integers, doubles, string, etc. An array can be created by using the **concatenate** function or `c()`.

```r
x <- c(1, 2, 3)
y <- c(4, 5, 7)
z <- c(7, 8, 9)

# Useful functions to perform on arrays/vectors
sum(x)
min(x)
median(x)
summary(x)
# the latter provides a summary of the functions above
```

Data frames

Data frames are the workhorse of statistical analysis in R. Data frames are essentially tables of data consisting of rows and columns. Both rows and columns can also be assigned headings. They can be constituted in a variety of ways.

```
# data.frame() can create columns from arrays and assign column names
df_1 <- data.frame(A = x, B = y, C = z)
# arrays must be of the same length!
```

Specific rows, columns, and cells can be referenced as follows:

```
# Return column "A" as a vector
df_1$A # "$" preceding the name of column "A"
df_1[ , 1] # [all rows, column = 1] - empty reference implies all
df_1 %>% pull(A) # Ctrl/Cmd + Shift + M for %>%

# Return row 2 as single row data frame
df_1[2, ]

# Return row 2-3 as two row data frame
df_1[2:3, ]

# Return cell in row 2 column 1
df_1[2, 1]

# Create a new column "D" that is the sum of A and B
df_1$D <- df_1$A + df_1$B # notice the assignment operator
# or
df_1 <- df_1 %>% mutate(D = A + B)
```

As you can see, there's more than one way to skin a cat in R.

Reading and writing data

In practice, you will likely be loading data from external files, such as a `.csv` file. For instance, create a data frame from the external file `Ireland_energy.csv` which contains Ireland's energy consumption data for 1980-2018. Do the same for the corresponding file for Ireland's population, `Ireland_population.csv`.

```
ire_energy <- read.csv(file = "data/Ireland_energy.csv", header = TRUE)
# "file" refers to the file path originating from your root directory
# "header" is set to true because the .csv file contains column headings

ire_pop <- read.csv(file = "data/Ireland_population.csv", header = TRUE)
```

Merge the two data frames to create a single data frame. Subsequently, create a new variable (or column) for the natural logarithm of Ireland's per capita energy consumption.

```
ireland_df <- merge(x = ire_energy, y = ire_pop, by.x = "Year", by.y = "Year")
# Create new data frame ireland_df
# merge() allows specification of two constituent data frames (x & Y)
# as well as their common column on the basis of which matching occurs (by.x & by.y)
```

```r
ireland_df <- ireland_df %>% mutate(ln_per_capita = log(GJ/Population))
# Assign 'new' data frame with additional variable
# tidyverse piping ( %>% )
# mutate(new_variable_name = transformation with existing columns)
# log()'s defualt setting implies natural logarithm
```

Table 2: Ireland's energy consumption

Year	**GJ**	**Population**	**ln__per__capita**
2018	695195813.70	4876547	4.96
2017	674283626.40	4813138	4.94
2016	665078014.40	4762595	4.94
2015	632644273.00	4708040	4.9
2014	603311823.70	4662713	4.86
2013	600839123.10	4627322	4.87
2012	601369213.40	4602095	4.87
2011	613012821.90	4582385	4.9
2010	669066288.50	4562835	4.99
2009	650362841.40	4538890	4.96
2008	720078003.90	4497485	5.08
2007	717332966.10	4413647	5.09
2006	727737237.30	4290635	5.13
2005	696672524.60	4171975	5.12
2004	669279007.60	4080615	5.1
2003	641655257.90	4004605	5.08
2002	647311914.00	3940007	5.1
2001	650201488.70	3874605	5.12
2000	625321139.70	3812075	5.1
1999	586079128.70	3760480	5.05
1998	543043222.10	3717525	4.98
1997	501619669.10	3678975	4.92
1996	473312049.10	3641872	4.87
1995	441490199.60	3611645	4.81
1994	431078491.50	3592195	4.79
1993	411899903.90	3577982	4.75
1992	394357343.90	3561165	4.71
1991	394415250.40	3537550	4.71
1990	379028868.40	3515725	4.68
1989	356521559.70	3510000	4.62
1988	338259545.20	3522475	4.56
1987	346035098.80	3538755	4.58

Regression tables

I have fairly strong preferences about how regression tables should look (threeparttable FTW). Luckily, the fantastic **modelsummary** package has us covered for nice looking regression tables, particularly since it automatically supports different Rmd output formats and backends. (For example, via the equally excellent **kableExtra** package.) This makges it easy to produce regression tables that look good in both HTML and PDF… although the latter requires that the corresponding LaTeX packages be loaded first. This template loads those LaTeX packages automatically, so tables like the below Just Work^{TM}.

```
library(fixest) ## For quick multi-model regression object

mods = feols(c(mpg, hp) ~ disp + csw(wt, drat) | cyl + vs, data = mtcars)

library(modelsummary)
library(kableExtra)

msummary(
  mods,
  title = "fixest: multi-model estimation",
  stars = TRUE,
  gof_omit = "Adj|Pseudo|Log|AIC|BIC"
  ) %>%
  add_footnote(
    c(paste("This footnote is pretty long. In fact, it runs over several lines",
          "of standard PDF output. Luckily that's no problem thanks to",
          "modelsummary, kableExtra, and threeparttable. As an aside, the",
          "fixest package is also amazing and you should use it."),
      "A shorter note."),
    threeparttable = TRUE
    ) %>%
  kable_styling(latex_options = "hold_position") ## (Optional) Print table directly below code
```

PDF support for non-standard fonts

This is an easy one; simply a matter of adding `dev: cairo_pdf` to the YAML. But it's nice not having to remember that every time, no?

Note: As the figure caption suggests, to run this next chunk you'll need to add Arial Narrow to your font book if it's not installed on your system already.

```
library(ggplot2)
library(hrbrthemes)

ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  labs(x = "Fuel efficiency (mpg)", y = "Weight (tons)",
       title = "This plot uses Arial Narrow fonts",
       caption = "Note: Fonts must be installed separately on your system.") +
  theme_ipsum()
```

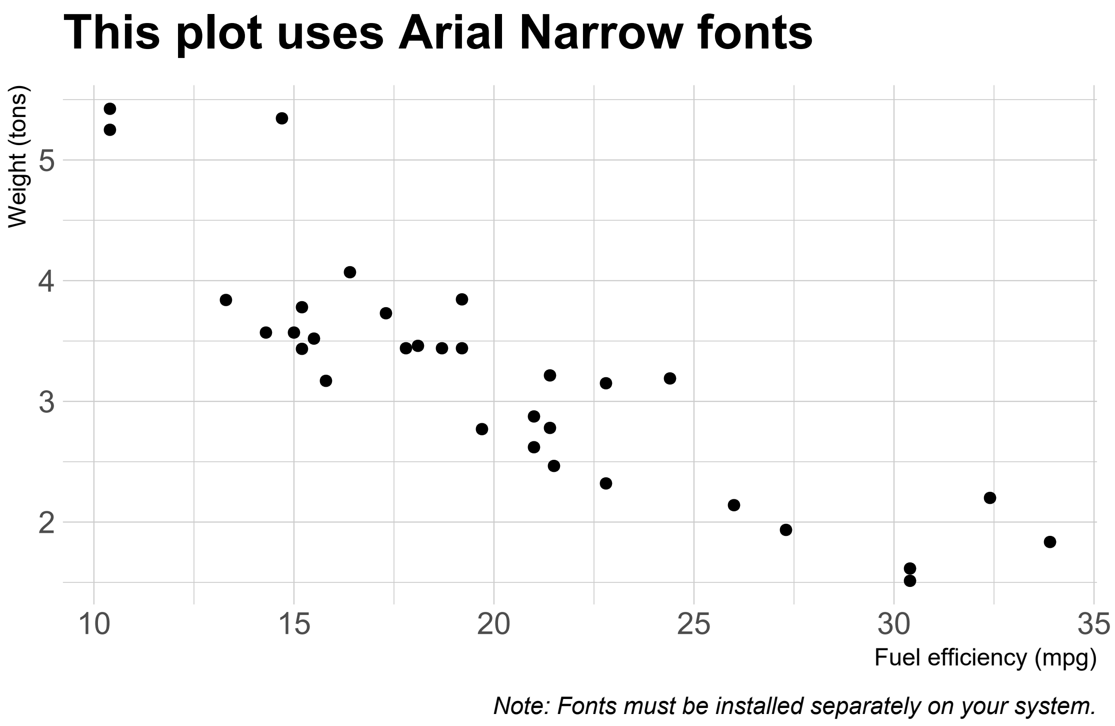
Table 3: fixest: multi-model estimation

	mpg	mpg	hp	hp
disp	0.002	0.002	0.104	0.126+
	(0.005)	(0.005)	(0.117)	(0.031)
wt	−3.403	−3.397	−3.502	2.863
	(1.331)	(1.168)	(8.289)	(6.026)
drat		0.038		37.781
		(1.223)		(44.566)
Num.Obs.	32	32	32	32
R2	0.839	0.839	0.721	0.756
R2 Within	0.396	0.396	0.012	0.138
Std.Errors	by: cyl	by: cyl	by: cyl	by: cyl
FE: cyl	X	X	X	X
FE: vs	X	X	X	X

+ $p < 0.1$, * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

[a] This footnote is pretty long. In fact, it runs over several lines of standard PDF output. Luckily that's no problem thanks to modelsummary, kableExtra, and threeparttable. As an aside, the fixest package is also amazing and you should use it.

[b] A shorter note.



Acknowledgements and further reading

Lecture notes are compiled from the following resources:

- *R intro* (2018) by Grant R. McDermott and Ed Rubin.

- *Data Science for Economics and Finance: Getting you staRted* (2021) by N.F. Katzke.

Should you need additional resources to get started, try the following:

- Quick-R
- Stata2R
- Data Science Programming Methods (STAT 447) by Dirk Eddelbuettel (University of Illinois)
- RStudio Cheatsheets
- Data Science for Economists (EC 607) by Grant McDermott (University of Oregon)
- Use your student credentials to sign up for a GitHub Pro account.
- Download GitHub Desktop for free and use version control for all your projects.