Reproducible Projects and Version Control

Itamar Caspi

November 25, 2019 (updated: 2019-11-26)

Replicating this Presentation

R packages used to produce this presentation

```
library(knitr) # for presenting tables
library(xaringan) # for rendering xaringan presentations
library(tidyverse) # for data wrangling and plotting
library(tidymodels) # for modelling the tidy way
```

If you are missing a package, run the following command

```
install.packages("package_name")
```

Alternatively, you can just use the pacman package that loads and installs packages:

```
if (!require("pacman")) install.packages("pacman")

pacman::p_load(tidyvers, tidymodels, knitr, xaringan)
```

From Best Practices to Methodology

Methodology	**Best Practice**
Machine learning	High dimensional statistics
Notebooks (R Markdown, Jupyter)	`# code annotation`
Version control	`mydoc_1_3_new_final_23.docx`
Git branches	`W:\`
Generate tables (SQL, dplyr, pandas)	Ask tables (xlsx)
Reproducibility	??
R, Python	Stata, SAS, EViews
Interdisciplinary teams	Solo

Outline

1. Reproducibility
2. The Tidyverse
3. Version Control
4. GitHub

RStudio Projects

Reproducibility

- Reproducible research allows anyone to generate your exact same results.
- To make your project reproducible you'll need to:
  - document what you did (code + explanations).
  - name the packages you used (including version numbers).
  - describe your R environment (R version number, operating system, etc.)
- Being in a "reproducible" state-of-mind means putting yourself in the shoes of the consumers, rather than producers, of your code.

(In "consumers" I also include the future you!)

An Aside: Docker

- Docker is a virtual computer inside your computer.
- Docker makes sure that anyone running your code will be able to perfectly reproduce your results.
- Docker solves a major predictability barrier: replicating your entire development environment (operating system, R versions, dependencies, etc.).
- For further details, see rOpenSci's tutorial.

RStudio Project Environment

- If your R script starts with `setwd()` or `rm(list=ls())` then are doing something wrong!
- Instead:
  1. Use RStudio's project environment.
  2. Go to `Tools -> Global Options -> General` and set the "Save workspace to .RData on exit" to **NEVER**.

R Markdown

- R Markdown notebooks, by RStudio, are perhaps THE go-to tool for conducting reproducible research in R.

- The process of "knitting" an Rmd file starts with a clean slate.

- An R Markdown file integrates text, code, links, figures, tables, and all that is related to your research project.

- R Markdown is perfect for communicating research. One if its main advantages is that an *.Rmd file is a "meta-document" that can be exported as a:

  - document (word, PDF, html, markdown).
  - presentation (html, beamer, xaringan, power point)
  - website (`blogdown`).
  - book (`bookdown`).
  - journal article (`pagedown`)
  - dashboard (`flexdashboards`).

The Tidyverse

This is Not a Pipe

Prerequisite: %>% is a pipe

- The "pipe" operator `%>%` introduced in the `magrittr` package, is deeply rooted in the `tidyverse`.

- To understand what `%>%` does, try associating it with the word "then".

- Instead of `y <- f(x)`, we type `y <- x %>% f()`. This might seen cumbersome at first, but consider the following two lines of code:

```
> y <- h(g(f(x), z))

> y <- x %>% f() %>% g(z) %>% h()
```

The second line of code should be read as: "take `x`, *then* put it through `f()`, *then* put the result through `g(. , z)`, *then* put the result through `h()`, and finally, keep the result in y.

Base R vs. the Tidyverse

- Consider the following data frame:

```
df <- data.frame(x = rnorm(10),
                 y = rnorm(10),
                 z = rnorm(10))
```

- Can you guess what the following code chunk does?

```
df_new <- df[df$x > 0, c("x", "y")]
df_new$xx <- df_new$x^2
```

- How about this one?

```
df_new <- df %>%
  select(x, y) %>%
  filter(x > 0) %>%
  mutate(xx = x^2)
```

How to read "piped" code?

```
df_new <- df %>%
  select(x, y) %>%
  filter(x > 0) %>%
  mutate(xx = x^2)
```

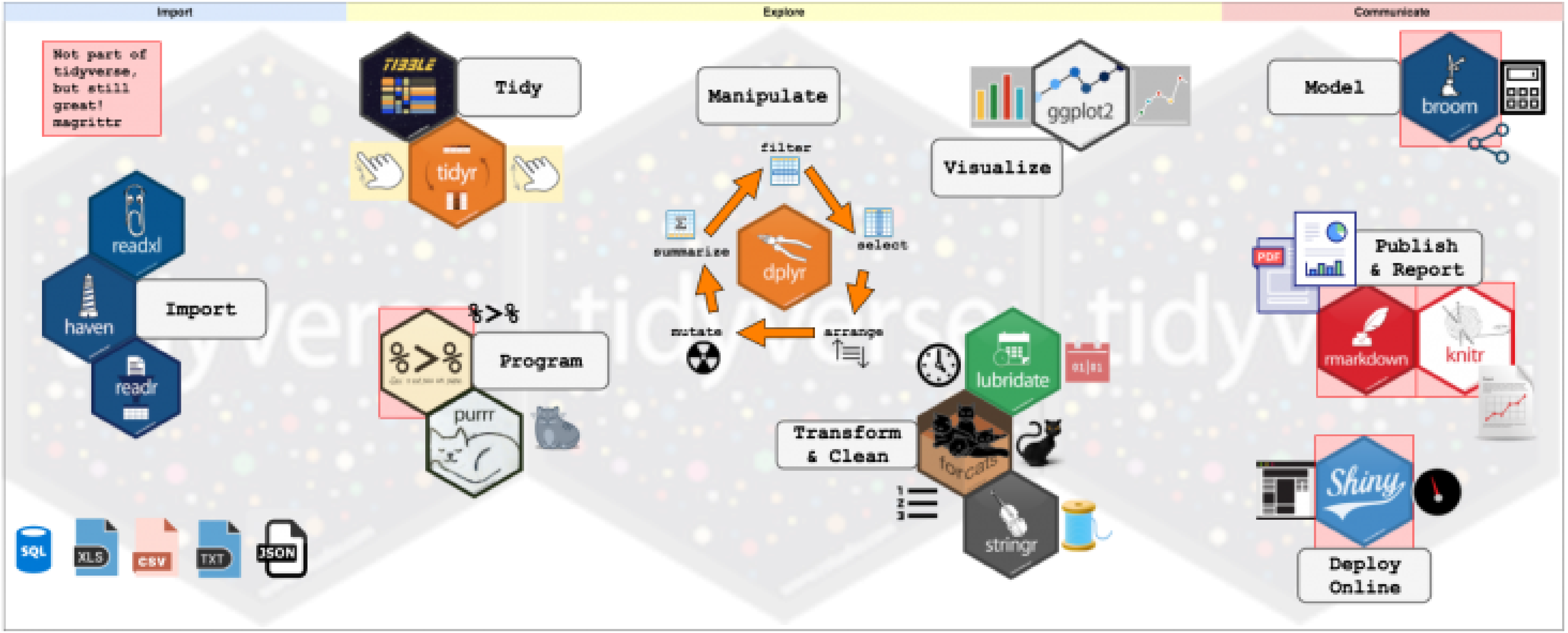
The above code chunk should be read as:

> "generate a new dataframe `df_new` by taking `df`, *then* select `x` and `y`, *then* filter rows where `x` is positive, *then* mutate a new variable `xx = x^2`"

Pros

- Following a "tidy" approach makes your code more readable $\Rightarrow$ more reproducible.
- I believe that there is a growing consensus in the #rstats community that we should learn the tidyverse first.
- Nevertheless, note that the tidyverse is "Utopian" in the sense that it strives toward *perfection*, and thus keeps changing. By contrast, base R was built to last.
- As usual, being proficient in both (base R and the tidyverse) will get you far...

The Tidyverse



Tidyverse Packages

Which packages come with `tidyverse`?

```
tidyverse_packages()
```

```
##  [1] "broom"        "cli"          "crayon"       "dplyr"        "dbplyr"
##  [6] "forcats"      "ggplot2"      "haven"        "hms"          "httr"
## [11] "jsonlite"     "lubridate"    "magrittr"     "modelr"       "purrr"
## [16] "readr"        "readxl\n(>=" "reprex"       "rlang"        "rstudioapi"
## [21] "rvest"        "stringr"      "tibble"       "tidyr"        "xml2"
## [26] "tidyverse"
```

Note that not all these packages are loaded by default (e.g., `lubrudate`.)

We now briefly introduce one the tidyvers flagships: `dplyr`.

`dplyr`: The grammar of data manipulation

`dplyr` is THE go-to tool for data manipulation

- Key "verbs":
  - `filter()` - selects observations (rows).
  - `select()` - selects variables (columns).
  - `mutate()` - generate new variables (columns).
  - `arrange()` - sort observations (rows).
  - `summarise()` - summary statistics (by groups).
- Other useful verbs:
  - `group_by()` - groups observations by variables.
  - `sample_n()` - sample rows from a table.
- And much more (see `dplyr` documentation)

The `tidymodels` package

- Tidymodels extends the tidyverse "grammar" philosophy to modelling tasks.

```
tidymodels::tidymodels_packages()
```

```
##  [1] "broom"         "cli"           "crayon"        "dials"        
##  [5] "dplyr"         "ggplot2"       "infer"         "magrittr"     
##  [9] "parsnip"       "pillar"        "purrr"         "recipes"      
## [13] "rlang"         "rsample"       "rstudioapi"    "tibble"       
## [17] "tidytext"      "tidypredict"   "tidyposterior" "yardstick"    
## [21] "tidymodels"
```

For further details, visit the tidymodels GitHub repo.

Resources

1. R for Data Science (r4ds) by Garrett Grolemund and Hadley Wickham.
2. Data wrangling and tidying with the "Tidyverse" by Grant McDerrmot.
3. Getting used to R, RStudio, and R Markdown by Chester Ismay and Patrick C. Kennedy.
4. Data Visualiztion: A practical introduction by Kieran Healy.

Version Control

Version Control

Git

- Git is a distributed version control system.
- Huh?!
- Sorry. Think of MS Word "track changes" for code projects.
- Git has established itself as the de-facto standard for version control and software collaboration.

GitHub

- GitHub is a web-based hosting service for version control using Git.
- OK, OK! Think of "Dropbox" for git projects. On steroids. And then some.
- GitHub is where and how a large share of open-source projects (e.g., R packages) are being developed.

Resources

1. Happy Git and GitHub for the useR by Jenny Bryan.
2. Version Control with Git(Hub) by Grant McDerrmot.
3. Pro Git.

Let's Practice!

Suggested workflow for starting a new (desktop) R project

RStudio:

1. Open RStudio.
2. File -> New Project -> New Directory -> New Project.
3. Name your project under "Directory name:". Make sure to check "Create git repository".

GitHub Desktop:

1. Open GitHub Desktop.
2. File -> Add local repository.
3. Set "Local path" to your RStudio project's folder.
4. Publish local git repo on GitHub (choose private or public repo).

Git Workflow

The **pull -> stage -> commit -> push** workflow:

1. Open GitHub Desktop.
2. Change "Current repository" to the cloned repo.
3. Click "Fetch origin" and **pull** any changes made to the GitHub repo.
4. Open your project.
5. Make changes to one or more of your files.
6. Save.
7. **stage** or unstage changed files.
8. write a summary (and description) of your changes.
9. Click "**Commit** to master".
10. Update remote: Click "**Push** origin" (Ctrl + P).

Clone and Sync a Remote GitHub Repository

Cloning:

1. Open GitHub Desktop.
2. Open the remote repository.
3. Click on "Clone or download".
4. Set the local path of your cloned repo (e.g., "C:/Documents/CLONED_REPO".

Syncing:

1. Open GitHub Desktop.
2. Change "Current repository" to the cloned repo.
3. Click the "Fetch origin" button.
4. **Pull** any changes made on the remote repo.

Your Mission

1. Login to RStudio Cloud.
2. Create your first R project.
3. Initiate Git.[1]
4. Create a new RMarkdown file.
5. Commit.

[1] RStudio automatically generates a `.gitignore` file that tells git which files to ignore (duh!). Click here for further details on how to configure what to ignore.

```
slides %>% end()
```

Source code