03 - Reproducibility and Version Control

ml4econ, HUJI 2023

Itamar Caspi
March 26, 2023 (updated: 2023-03-26)

Replicating this Presentation

R packages used to produce this presentation

```r
library(tidyverse)  # for data wrangling and plotting
library(tidymodels) # for modeling the tidy way
```

```
## Warning: package 'tidymodels' was built under R version 4.2.2

## Warning: package 'dials' was built under R version 4.2.2

## Warning: package 'infer' was built under R version 4.2.2

## Warning: package 'modeldata' was built under R version 4.2.2

## Warning: package 'parsnip' was built under R version 4.2.2

## Warning: package 'recipes' was built under R version 4.2.2

## Warning: package 'rsample' was built under R version 4.2.2

## Warning: package 'tune' was built under R version 4.2.2
```

From Best Practices to Methodology

Best Practice	**Methodology**
High dimensional statistics	Machine learning
`# code annotation`	Notebooks (R Markdown, Jupyter)
`mydoc_1_3_new_final_23.docx`	Version control
Ready to use tables (xlsx)	Generate tables (SQL, dplyr, pandas)
??	Reproducibility
Stata, SAS, EViews	R, Python, Julia
work solo	Interdisciplinary teams

Outline

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

RStudio Projects

Reproducibility

- **Reproducible research:** Enables others to replicate your results
- **Project requirements:**
  - Document your work (code and explanations)
  - List used packages (including version numbers)
  - Detail your R environment (R version, OS, etc.)
- **Reproducible mindset:** Focus on code consumers, including your future self

An Aside: `renv`

- `renv` **package:** Create reproducible environments for R projects

- **Key benefits:**

  - *Isolated:* Private package library for each project
  - *Portable:* Easily transfer projects across computers and platforms
  - *Reproducible:* Records exact package versions for consistent installations

- **Learn more:** Introduction to `renv`

RStudio Project Oriented Workflow

- **Avoid `setwd()` and `rm(list=ls())`:** Improper R script practices

- **Recommended alternatives:**

  1. Utilize RStudio's project environment

  2. Modify settings:

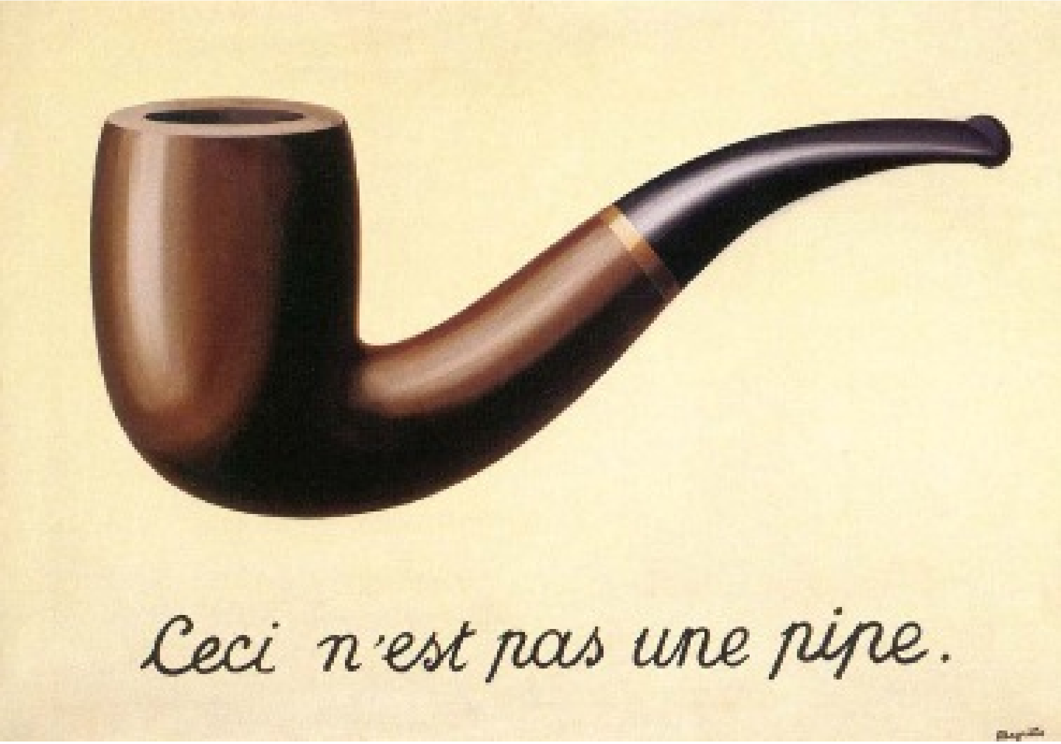
     - Go to `Tools -> Global Options -> General`
     - Set "Save workspace to .RData on exit" to **NEVER**

R Markdown

- **R Markdown notebooks:** Premier tool for reproducible research in R
- **Knitting process:** Starts with a clean slate
- **R Markdown file:** Integrates text, code, links, figures, tables, etc.
- **Ideal for communication:** Export .Rmd file as:
    - Document (Word, PDF, HTML, Markdown)
    - Presentation (HTML, Beamer, Xaringan, PowerPoint)
    - Website (`blogdown`)
    - Book (`bookdown`)
    - Journal article (`pagedown`)
    - Dashboard (`flexdashboards`)

The Tidyverse

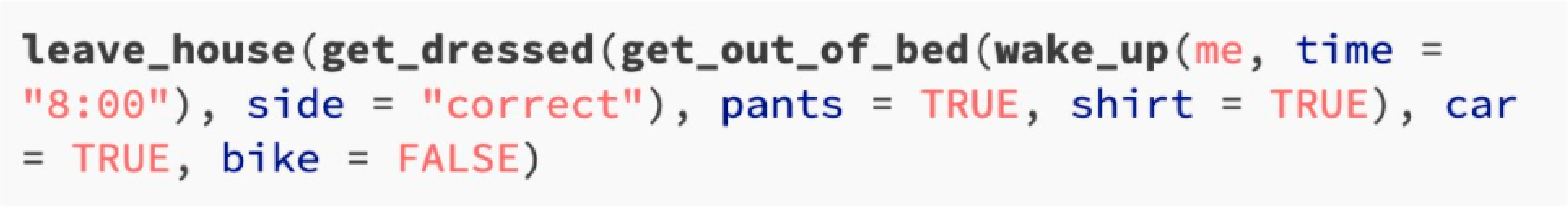
This is Not a Pipe



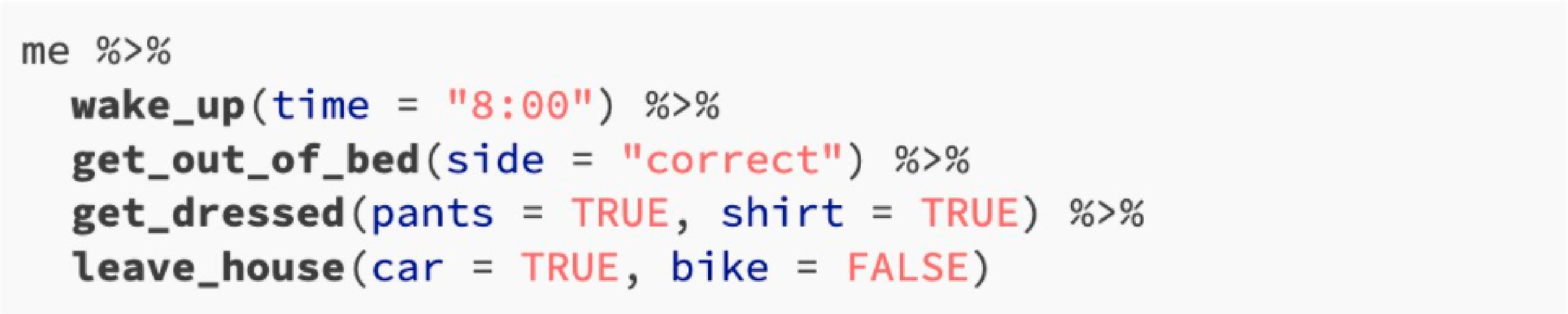
Prerequisite: %>% is a pipe

- **Pipe operator %>%:** From `magrittr` package, integral to `tidyverse`
- **Understanding %>%:** Associate it with "then"
- **Example:**
  - Traditional: `y <- h(g(f(x), z))`
  - With pipe: `y <- x %>% f() %>% g(z) %>% h()`
  - Read as: Take `x`, *then* apply `f()`, *then* apply `g(. , z)`, *then* apply `h()`, and store result in `y`

Morning Routine



```
leave_house(get_dressed(get_out_of_bed(wake_up(me, time =
"8:00"), side = "correct"), pants = TRUE, shirt = TRUE), car
= TRUE, bike = FALSE)
```



```
me %>%
  wake_up(time = "8:00") %>%
  get_out_of_bed(side = "correct") %>%
  get_dressed(pants = TRUE, shirt = TRUE) %>%
  leave_house(car = TRUE, bike = FALSE)
```

Source: https://twitter.com/andrewheiss/status/1359583543509348356?s=20

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
```

"Piped" code?

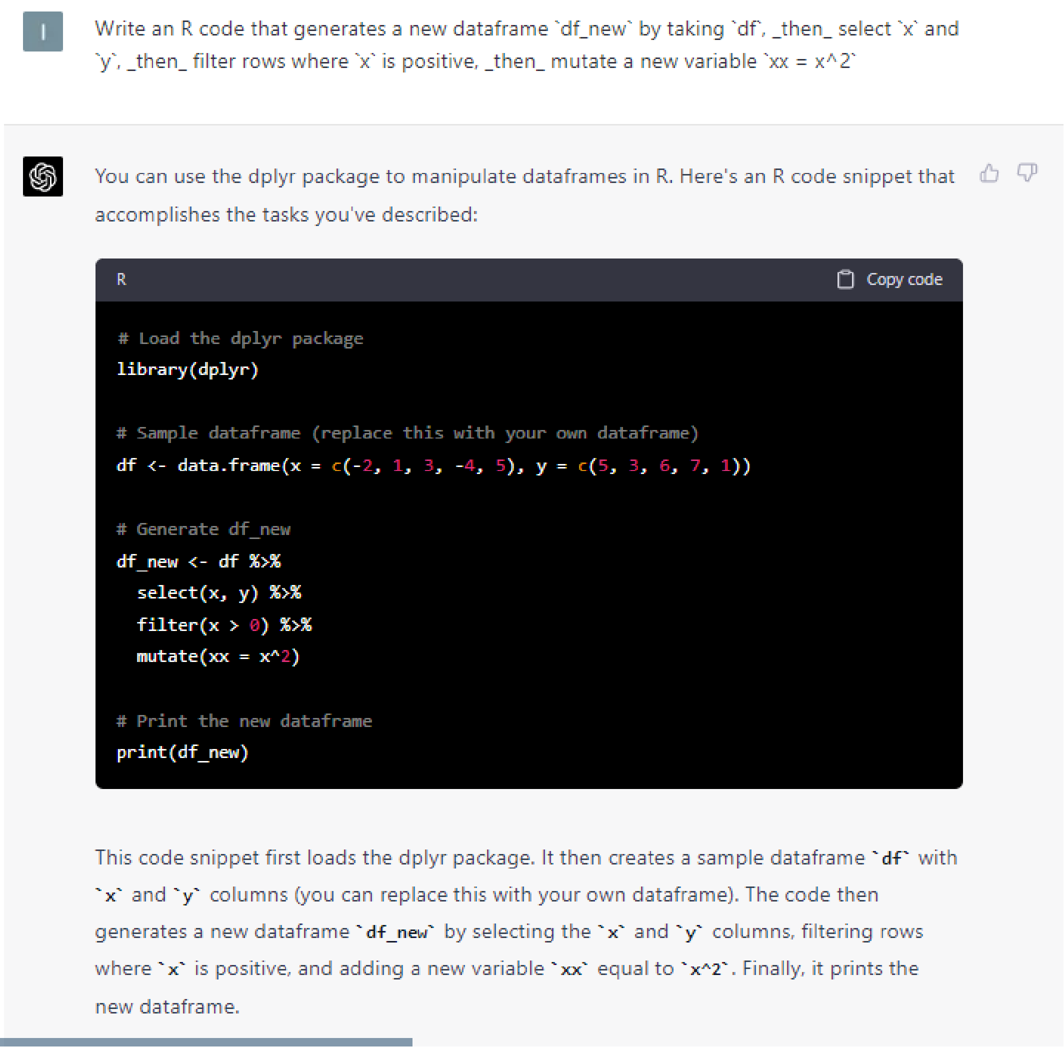
- How about this one?

```r
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`"

Enter ChatGPT...



Write an R code that generates 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`

You can use the dplyr package to manipulate dataframes in R. Here's an R code snippet that accomplishes the tasks you've described:

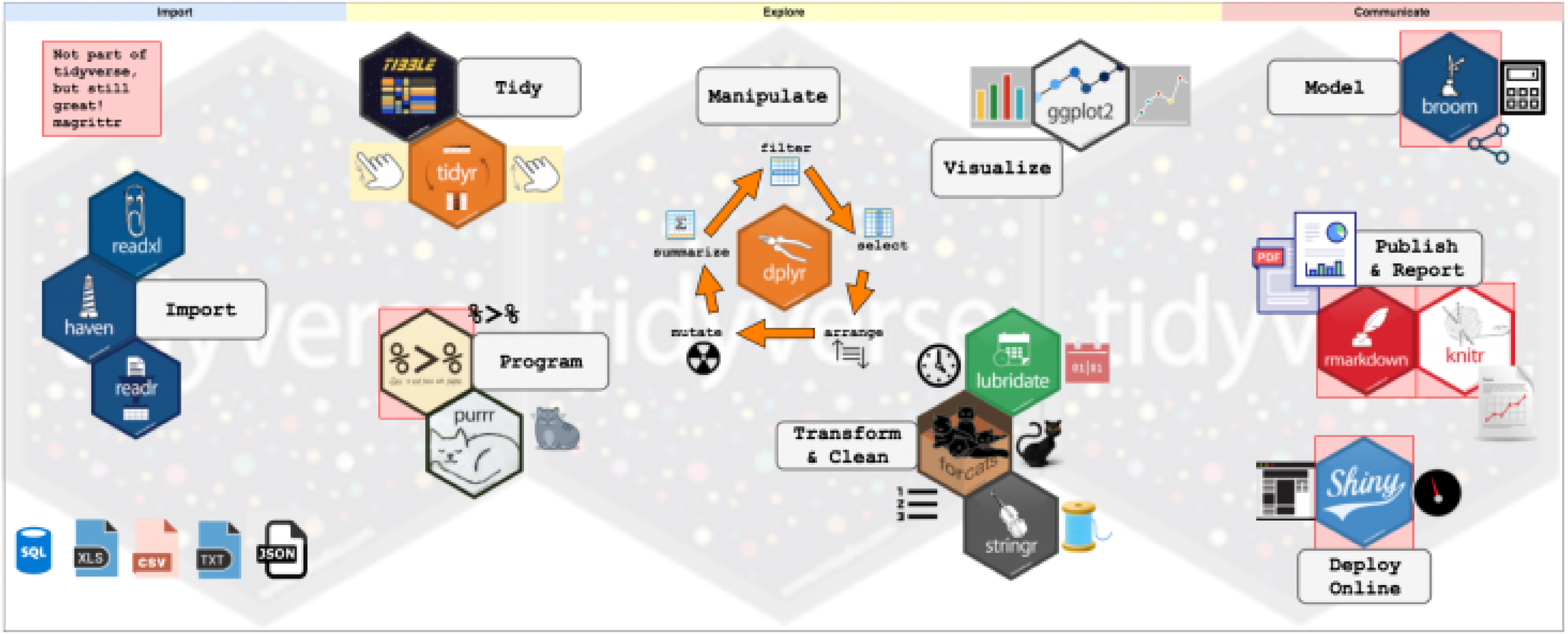
```R
# Load the dplyr package
library(dplyr)

# Sample dataframe (replace this with your own dataframe)
df <- data.frame(x = c(-2, 1, 3, -4, 5), y = c(5, 3, 6, 7, 1))

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

# Print the new dataframe
print(df_new)
```

This code snippet first loads the dplyr package. It then creates a sample dataframe `df` with `x` and `y` columns (you can replace this with your own dataframe). The code then generates a new dataframe `df_new` by selecting the `x` and `y` columns, filtering rows where `x` is positive, and adding a new variable `xx` equal to `x^2`. Finally, it prints the new dataframe.

Pros & cons

- Following a "tidy" approach makes your code more readable ⇒ 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"        "dbplyr"        "dplyr"         "dtply
##  [7] "forcats"       "ggplot2"       "googledrive"   "googlesheets4" "haven"         "hms"
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"      "modelr"        "pilla
## [19] "purrr"         "readr"         "readxl"        "reprex"        "rlang"         "rstud
## [25] "rvest"         "stringr"       "tibble"        "tidyr"         "xml2"          "tidyv
```

Note that not all these packages are loaded by default.

We now briefly introduce the tidyverse's flagship: `dplyr`.

`dplyr`: The grammar of data manipulation

- **`dplyr`:** Essential tool for data manipulation
- **Key verbs:**
  - `filter()` - Select observations (rows)
  - `select()` - Select variables (columns)
  - `mutate()` - Generate new variables (columns)
  - `arrange()` - Sort observations (rows)
  - `summarise()` - Summary statistics (by groups)
- **Some additional verbs:**
  - `group_by()` - Group observations by variables
  - `sample_n()` - Sample rows from a table
- **Learn more:** `dplyr` documentation

The `tidymodels` package

- Tidymodels extends the tidyverse's "grammar" philosophy to modeling tasks.

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

```
##  [1] "broom"        "cli"          "conflicted"   "dials"        "dplyr"        "ggplot2"
##  [8] "infer"        "modeldata"    "parsnip"      "purrr"        "recipes"      "rlang"
## [15] "rstudioapi"   "tibble"       "tidyr"        "tune"         "workflows"    "workflowse
## [22] "tidymodels"
```

For more information, visit the tidymodels GitHub repo.

Resources

1. R for Data Science (r4ds) by Garrett Grolemund and Hadley Wickham.

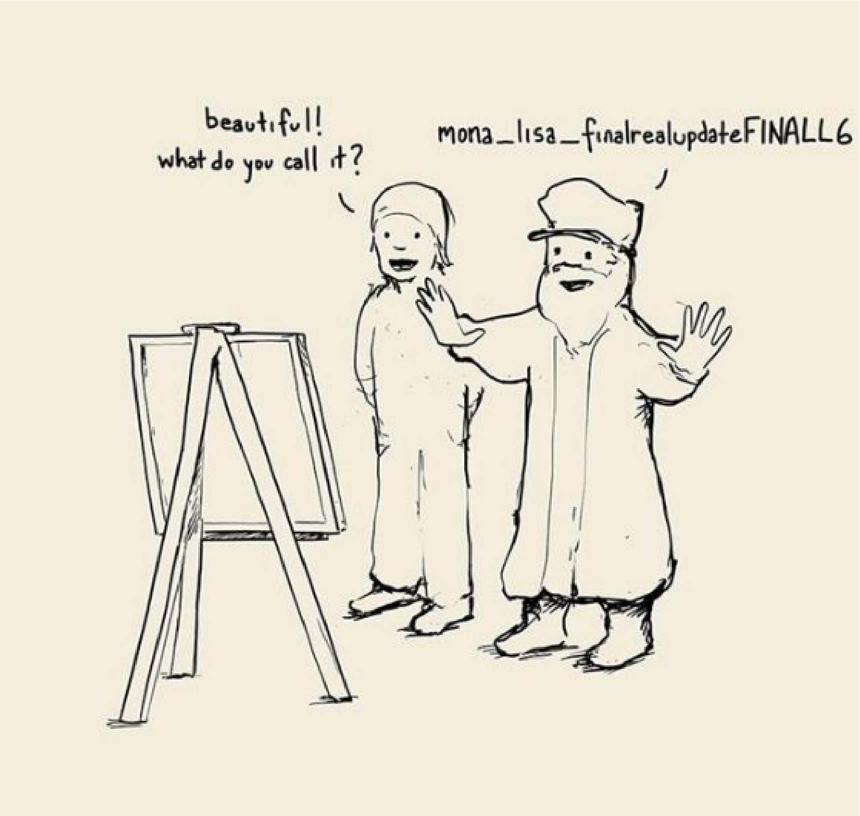
2. Data wrangling and tidying with the “Tidyverse” by Grant McDerrmot.

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



What's wrong with the "*.X_FINAL_FINAL" method?

- What changed?
- Where??
- When???
- By who????

You get the picture...

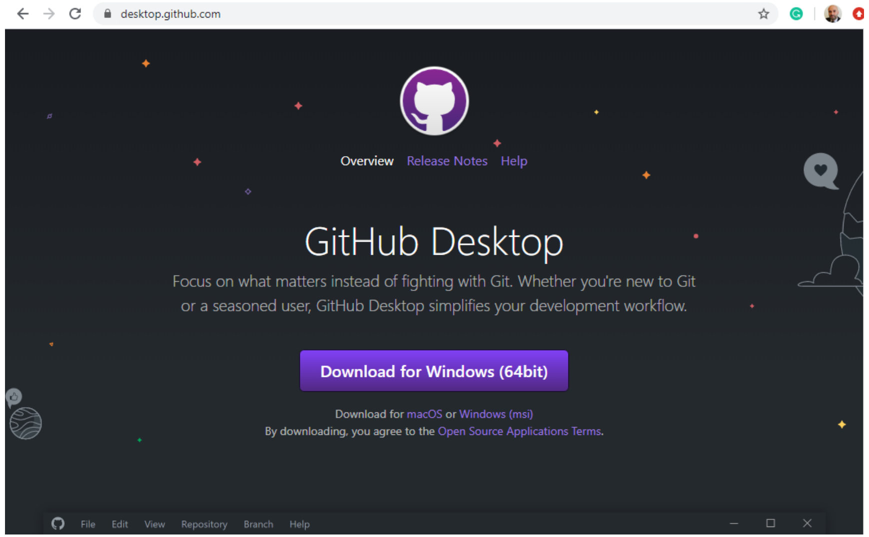
Git

- Git is a distributed version control system.
- Huh?!
- Sorry. Think of "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 that uses Git.
- It can be thought of as "Dropbox" for Git projects, offering advanced features beyond basic version control.
- GitHub is a popular platform for developing open-source projects, including popular R packages and other software libraries.

GitHub Desktp



GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Download for macOS or Windows (msi)
By downloading, you agree to the Open Source Applications Terms.

- GitHub Desktop is a user-friendly graphical interface that allows developers to interact with Git repositories.
- It provides an intuitive way to manage changes to code, create and switch branches, and synchronize local and remote repositories.
- GitHub Desktop also simplifies collaboration by making it easy to create and review pull requests, resolve merge conflicts, and manage code reviews.

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. Navigate to File -> New Project -> New Directory -> New Project.
  3. Name your project in the "Directory name:" field and check "Create git repository".

- **GitHub Desktop:**

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

Suggested Git Workflow (Optional)

- **Pull, Stage, Commit, Push Workflow:**

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

Clone and Sync a Remote GitHub Repository (Optional)

- **Cloning a Repository:**

    1. Launch GitHub Desktop.
    2. Navigate to the remote repository.
    3. Select "Clone or download".
    4. Define the local path for your cloned repo (e.g., "C:/Documents/CLONED_REPO").

- **Synchronizing a Repository:**

    1. Launch GitHub Desktop.
    2. Switch "Current repository" to the cloned repo.
    3. Press the "Fetch origin" button.
    4. **Pull** any updates made on the remote repo.

Your Homework

- **Getting Started with R and Git:**

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

[1] RStudio automatically generates a `.gitignore` file that tells Git which files to ignore. Click here for more details on configuring what to ignore.

`slides %>% end()`

Source code