San Diego County Office of Education

Assessment, Accountability, and Evaluation

Oct 05, 2021

Initializing a New Data Analysis Project

- Create a new repo in GitHub.
  - Make a repository name
  - Add a brief description
- In RStudio, create a new project:
  - Select File "New Project" and "Version Control" from the Project Wizard
  - Select Git
  - Enter GitHub repository URL and project directory name
  - Check sub directory and "Create Project"
  - *Once only:* Install `packrat` by using the command install.packages("packrat")
  - Use the command `packrat::init()` in the existing project
  - Observe "packrat" folder in pane
- Create file folder structure and populate
  - Create a `data` folder with two sub folders: `raw` and `processed.` The raw folder is for all raw data. This folder is read only and no file should be altered. The processed folder is for all processed, cleaned, and tidied data sets. All data in processed folder must be saved in .Rds format. Name processed files to represent what the data mean (e.g., merged_months).
  - Next create an `output` folder with three sub folders: `figures` to store plots, diagrams, and other figures. Save them as .png files and give them a meaningful name; `reports` for communications in R Markdown (or other format); and a `results` sub folder for all the data output of analysis versus an object like a figure or table. Some output files are .Rds files. Use the results sub folder to house these kinds of files.
  - Next, create a `scripts` folder. Here keep all R scripts. Use prefix numbers if scripts need to be run in order (e.g., `01 scrape_data.R`, `02 tidy_data.R`). Here .R or .Rmd files are appropriate. Create a sub folder called deprecated for all scripts that get used initially but are abandoned for one reason or another.
  - All projects must also have `.gitignore`, `readme`, and `.Rproj` files in the main folder. The `.gitignore` file lists files that won's be added to the Git system. The `.Rproj` file contains all the meta-data of the project. The `readme` file briefly describes all high-level information about the project (i.e., use the standardized readme file).
  - Depending on the project, other folder might be necessary such as `references`, `presentations`, `paper`, etc.
- The data analysis workflow

San Diego County Office of Education

Assessment, Accountability, and Evaluation

Sep 21, 2021

Initializing a New Data Analysis Project

- Create a new repo in GitHub.
  - Make a repository name
  - Add a brief description
- In RStudio, create a new project:
  - Select File "New Project" and "Version Control" from the Project Wizard
  - Select Git
  - Enter GitHub repository URL and project directory name
  - Check sub directory and "Create Project"
  - *Once only:* Install `packrat` by using the command install.packages("packrat")
  - Use the command `packrat::init()` in the existing project
  - Observe "packrat" folder in pane
- Create file folder structure and populate
  - Create a `data` folder with two sub folders: `raw` and `processed.` The raw folder is for all raw data. This folder is read only and no file should be altered. The processed folder is for all processed, cleaned, and tidied data sets. All data in processed folder must be saved in .Rds format. Name processed files to represent what the data mean (e.g., merged_months).
  - Next create an `output` folder with three sub folders: `figures` to store plots, diagrams, and other figures. Save them as .png files and give them a meaningful name; `reports` for communications in R Markdown (or other format); and a `results` sub folder for all the data output of analysis versus an object like a figure or table. Some output files are .Rds files. Use the results sub folder to house these kinds of files.
  - Next, create a `scripts` folder. Here keep all R scripts. Use prefix numbers if scripts need to be run in order (e.g., `01 scrape_data.R`, `02 tidy_data.R`). Here .R or .Rmd files are appropriate. Create a sub folder called deprecated for all scripts that get used initially but are abandoned for one reason or another.
  - All projects must also have `.gitignore`, `readme`, and `.Rproj` files in the main folder. The `.gitignore` file lists files that won's be added to the Git system. The `.Rproj` file contains all the meta-data of the project. The `readme` file briefly describes all high-level information about the project (i.e., use the standardized readme file).
  - Depending on the project, other folder might be necessary such as `references`, `presentations`, `paper`, etc.
- The data analysis workflow

- Breakdown the RMarkdown file into modularized chunks:
  - `Load libraries`. Keep all the dependencies clear at the beginning of the file by defining which libraries are needed to execute the code.
  - Define or source `functions.` Functions (such as cleaning functions, or data scraping functions) are external files that are needed for the code to work. Keep these files near the libraries.
  - `Import` data. Next, load the data for the project.
  - `Data wrangling`. Define the code necessary to tidy and transform the data.
  - `Visualize/Model`. Create visuals to explore the data and models (if needed).
  - `Communicate.` Design reports or dashboards or other forms of communication (such as presentations) for the data analysis.
- .R and .Rmd files guidance
  - First, do not use `install.packages()` inside scripts. Use `packrat` to configure all packages or create a separate `configure.R` file that will install packages (e.g., `pkgs <- c("ggplot2", "plyr") install.packages(pkgs)`.
  - Use `library()` to call all package libraries and avoid `require()`.
  - Use the package `here` and set relative paths. For example, `cars <- read.csv(file = here("data", "raw", "cars.csv")`. Avoid relative paths like `"C:/Users/..."`
  - Do not hardcode passwords into the script. Create a parameter and supply when knitting or use another approach.
  - If the project requires random generation, use `set.seed()` to get the same random split each time (e.g., `set.seed(1991)`).
  - Do not repeat yourself (DRY). If the code is repeated more than two times, wrap it in a function.
  - Move deprecated code or unnecessary tests, etc., to a junk folder.
  - Separate function definitions from their applications. Create an `.R` script for functions and name it appropriately such as util.R.
  - Save all data in .Rds format using `saveRDS()`.
  - Follow all scripting conventions using the tidyverse style guide. The package "styler" is the easiest way to configure scripts.
- Commit changes to GitHub regularly.
  - Typically, a research project involves only a small number of collaborators who trust each other. Employ the centralized commit process, where all collaborators push into the central master branch. However, there may be times when major changes need to have a branch before merging back into the master branch.

Resources

Happy Git and the GitHub for the useR

Designing R projects

What they forgot to teach you about R