Reproducible Research: Why and How

Part 3: Analytic Solutions

Sam Harper

SER Pre-Conference Workshop
2020-10-30

3. Analytic Solutions

3.1 Workflow Management

3.2 Documentation

3.3 Literate Programming

3.4 Version Control

3.4 Dynamic Documents

3. Analytic Solutions

3.1 Workflow Management

3.2 Documentation

3.3 Literate Programming

3.4 Version Control

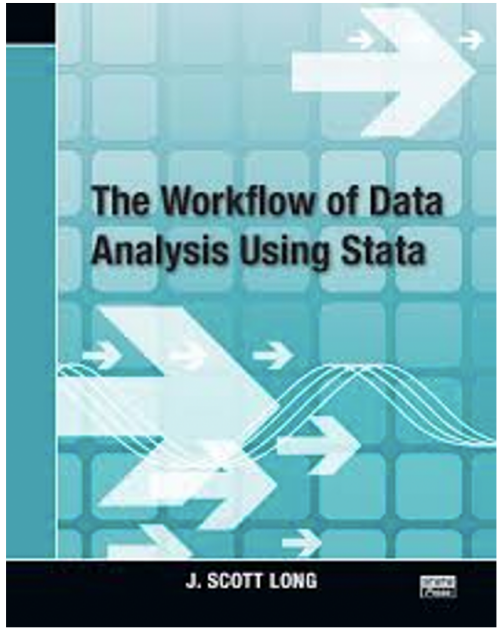
3.4 Dynamic Documents

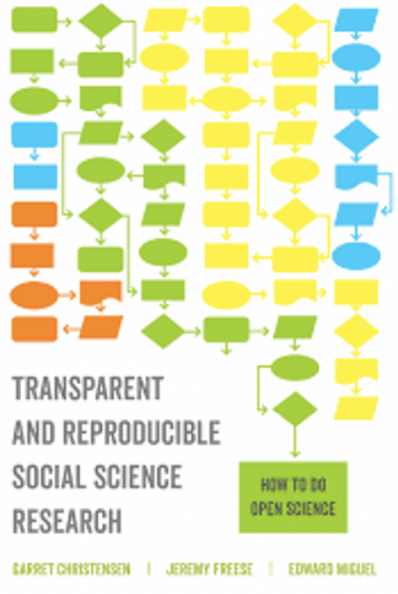
Workflow Advice

Resources for advice on:

- Planning and organization
- Documentation
- Writing / debugging syntax
- Automating your work
- Variable labeling / naming
- Cleaning
- Archiving

Often specific to software but core ideas are the same.



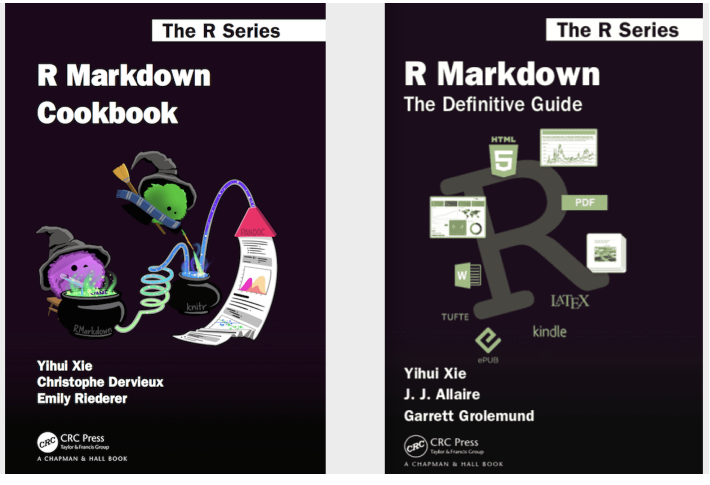


Long (2009); Christensen et al. (2019)

Great online **free** resources for *R*

Technical help for:

- Learning *R*.
- Developing a reproducible workflow.
- Tips & tricks.
- Integration with other software.
- Dealing with frustration.*



*Which is inevitable.

See the series at bookdown.org

Planning your work

Why?

Will save you time.

Plans should help you stay on track.

Hard, and isn't "fun".

What:

- Goals and publishing plans
- Scheduling
- Division of labor
- Datasets needed
- Variable names and labels
- Missing data procedures
- Analysis
- Documentation
- Backing up / archiving

Planning for the entire research pipeline

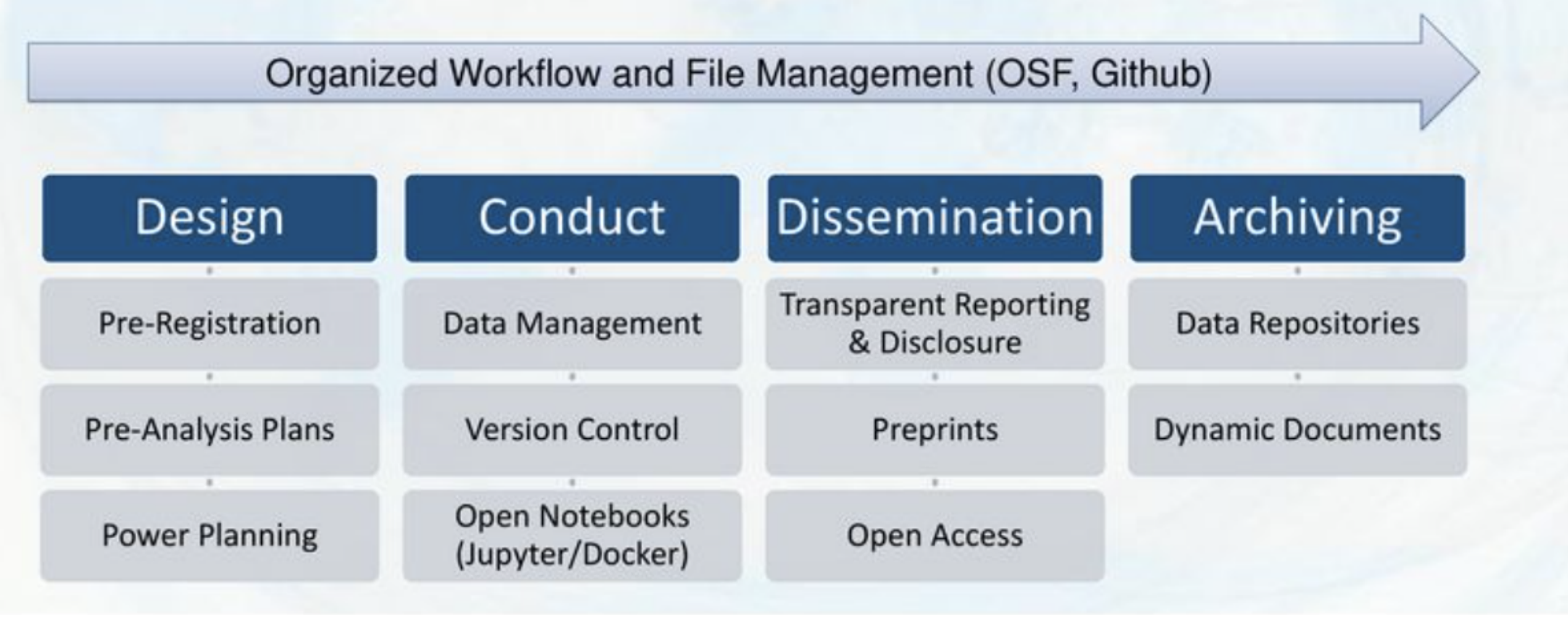
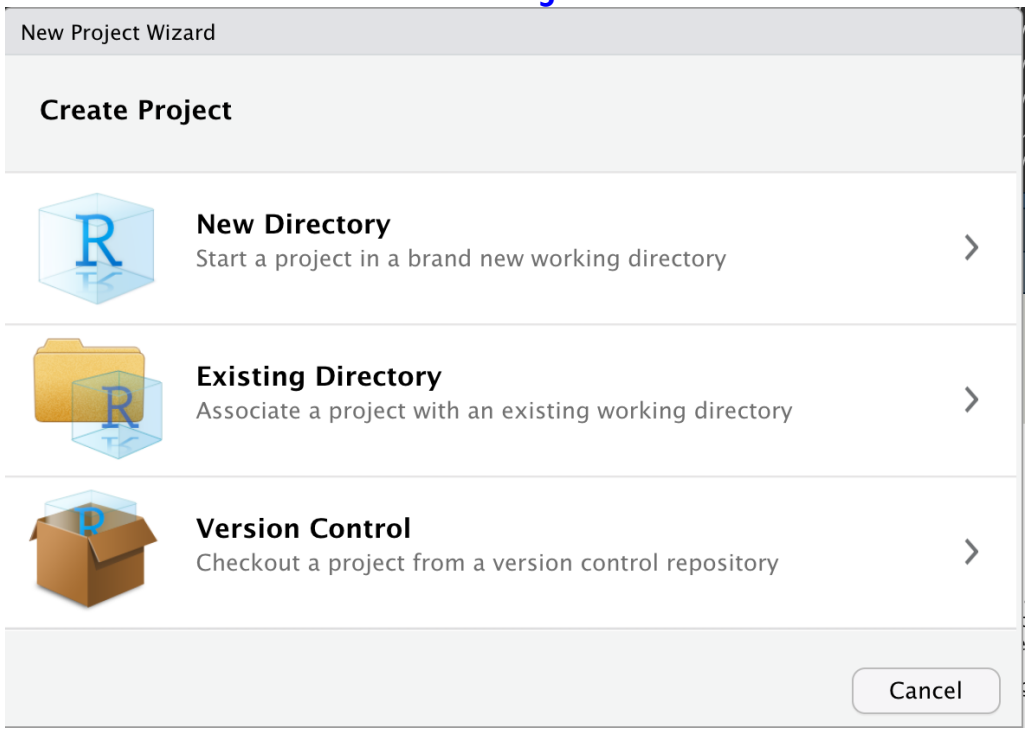


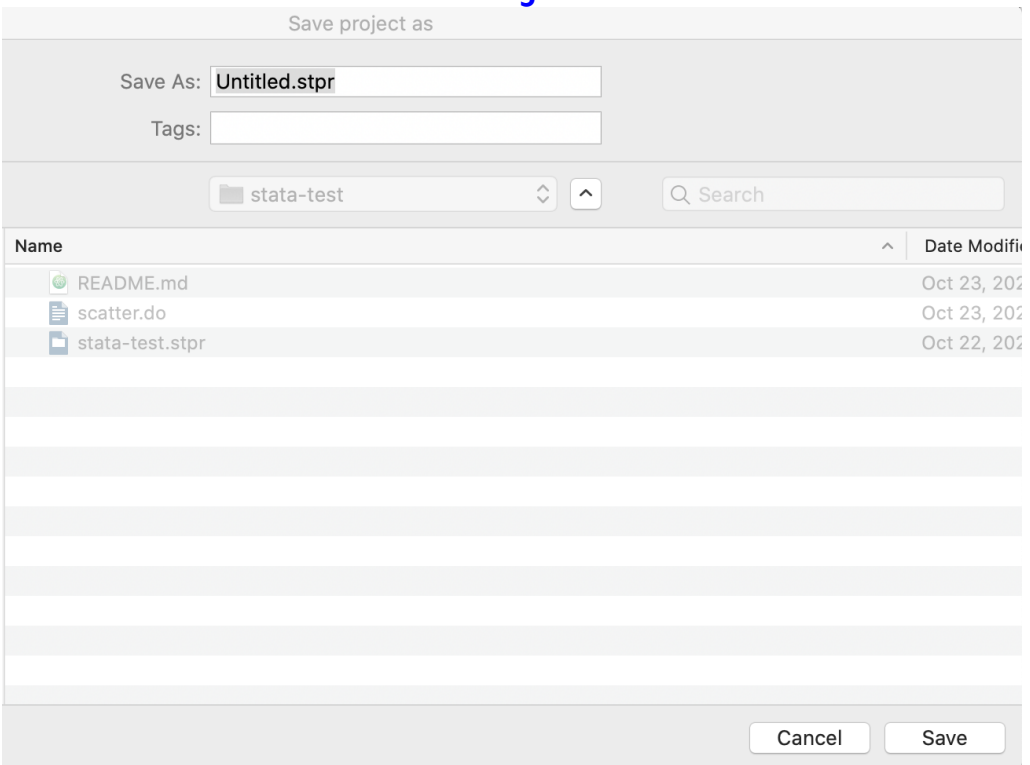
Image credit: OSF

Use projects to keep things organized

RStudio: File > New Project...



Stata: File > New > Project...



Jenny Bryan on Project-oriented workflows. Also possible with SAS.

Why Projects?

By definition, the directory `/User/samharper/project/data` is not reproducible.
But `project/data` works anywhere.

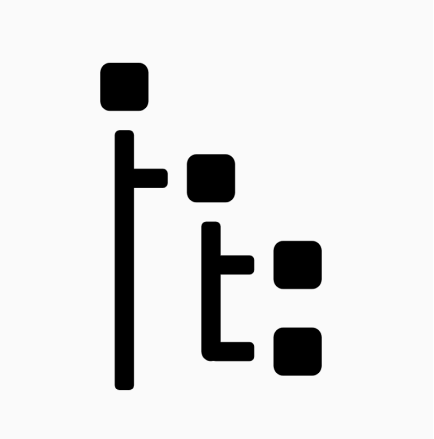
Self contained and portable

Easy to integrate with version control

Working directory set at project location when opened

Fresh session when project is launched

File organization



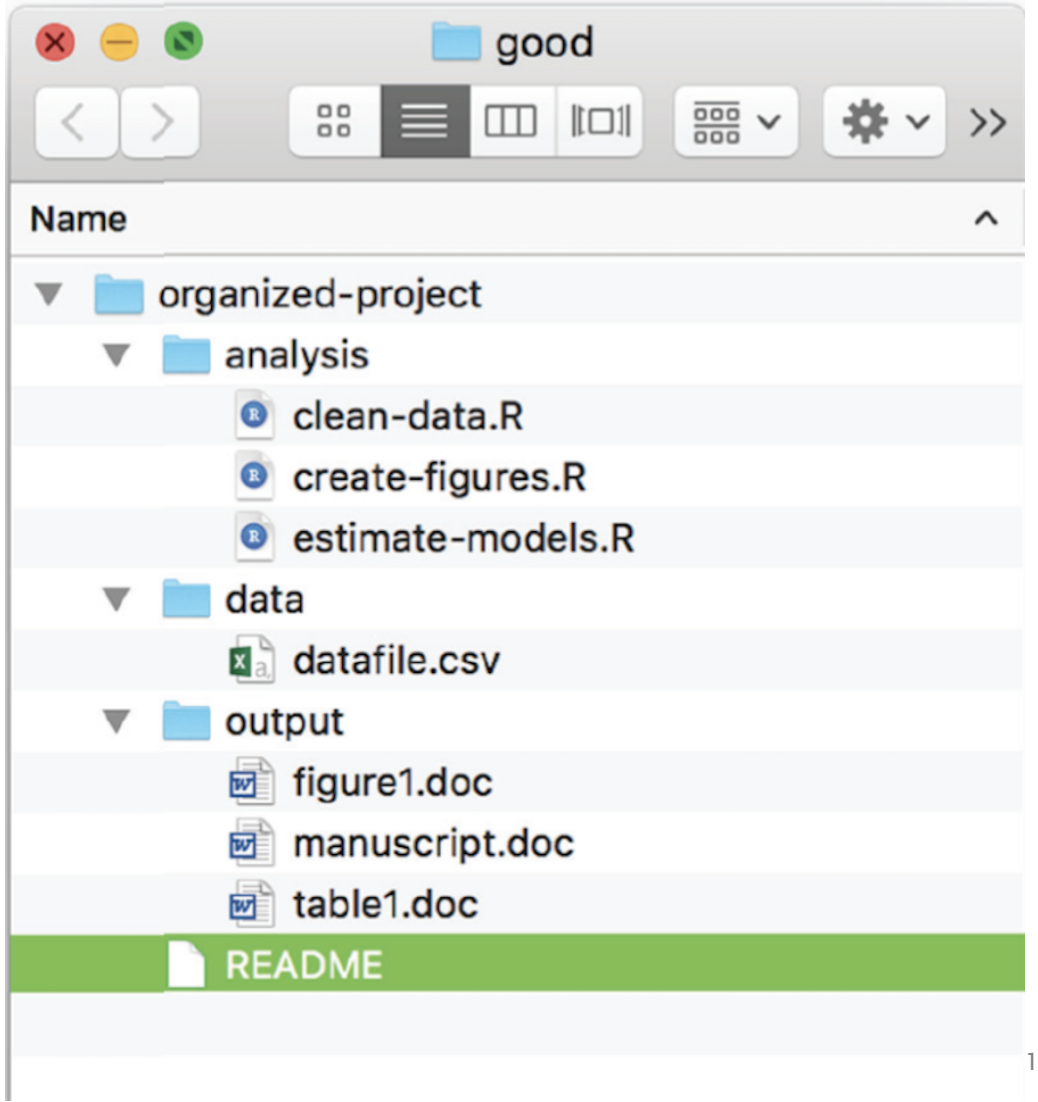
Core idea is to separate raw data, analytic data, code, and outputs.

Why?

- Raw data is **never** altered.
- Separating code from data streamlines re-analysis.
- Outputs (tables, figures) are transient and should be easily regenerated from data + scripts.

Look familar?

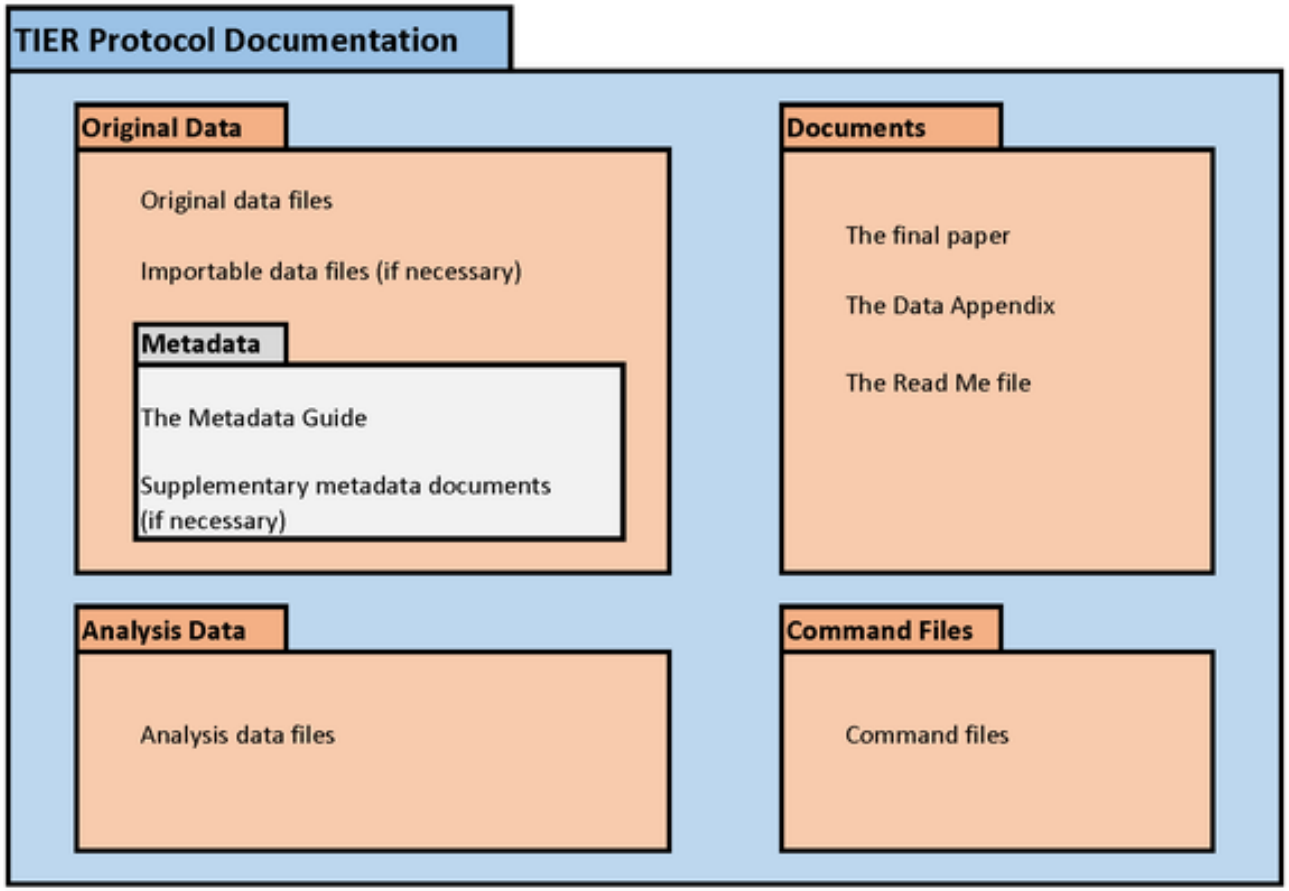
That's more like it.



Many variations (Project TIER scheme)

Also consider including a folder or file for 'metadata'.

Helps computers find your research!



(https://projecttier.org)

Tip: Don't obsess about rigidly following someone else's workflow.

Important thing is to find a workflow that works for you.

Your turn (finally!): Create a project

- Use RStudio or Stata (or something else) to create a new project
- Call it 'myproject'

- Create folders for `data`, `code`, and `output`.

- Now close the program.

- Browse to where the program file exists and double-click.
- Where is the current directory located? (Hint: type `getwd()` into the R console or `pwd` in the Stata command line).

- Are you in the right location? ✅ if yes, 🛑 if no

04:00

Can also be embedded in the context of a much larger project structure.

Note that "\Source" and "\Derived" data are never in the same folder.

Scott Long - Workflow Chapter 2 - designing a directory structure				
2008-04-17				
Project Directory	**Level 1**	**Level 2**	**Level 3**	**Purpose**
\AgeDisc				Project directory.
	\- To file			Files to examine and move to appropriate location.
	\Administration			Administration.
		\Budget		Budget sheets.
		\Correspondence		Letters and e-mails.
		\Proposal		Grant proposal and related materials.
	\Documentation			Documentation for project.
		\Codebooks		Codebooks for source and constructed variables.
	\Hold then delete			Delete when project is complete.
		\2007-06-13 submited		Do, data and text when paper was submitted.
		\2008-04-17 revised		Do, data and text when revisions are sumbitted.
		\2008-01-02 accepted		Do, data and text when paper is accepted.
	\Posted			Completed files that cannot be changed.
		\- Datasets		Datasets.
			\Derived	Dataset constructed from original data files.
			\Source	Original data without modifications.
		\- Text		Completed drafts of paper.
		\DataClean		Data cleaning and variable construction.
		\DescStats		Descriptive statistics and sample selection.
		\Figures		Graphs of data.
		\PanelModels		Panel models for discrimination.
	\Readings			Articles related to project; bibliography.
	\Work			Work directory.
		\- To do		Work that hasn't been started.
		\Text		Active drafts of paper.

Example from Long (2009)

3. Analytic Solutions

3.1 Workflow Management

3.2 Documentation

3.3 Literate Programming

3.4 Version Control

3.4 Dynamic Documents

"It is always faster to document it today than tomorrow."

-J Scott Long

What should be documented?

- Data sources
- Data decisions
- Statistical analysis
- Software
- Storage

Levels of documentation

- Research log
- Codebooks
- Dataset documentation

Research Log

The research log should help with:

- Keeping the work on track. Ideas, decision about analysis, papers you read that are relevant...document your thinking!
- Dealing with interruptions.
- Facilitating replication by others (especially by you 6 months later).

Ideally this should document what you did, why you did it, and how you did it.

Alexander Graham Bell did it, so can you!

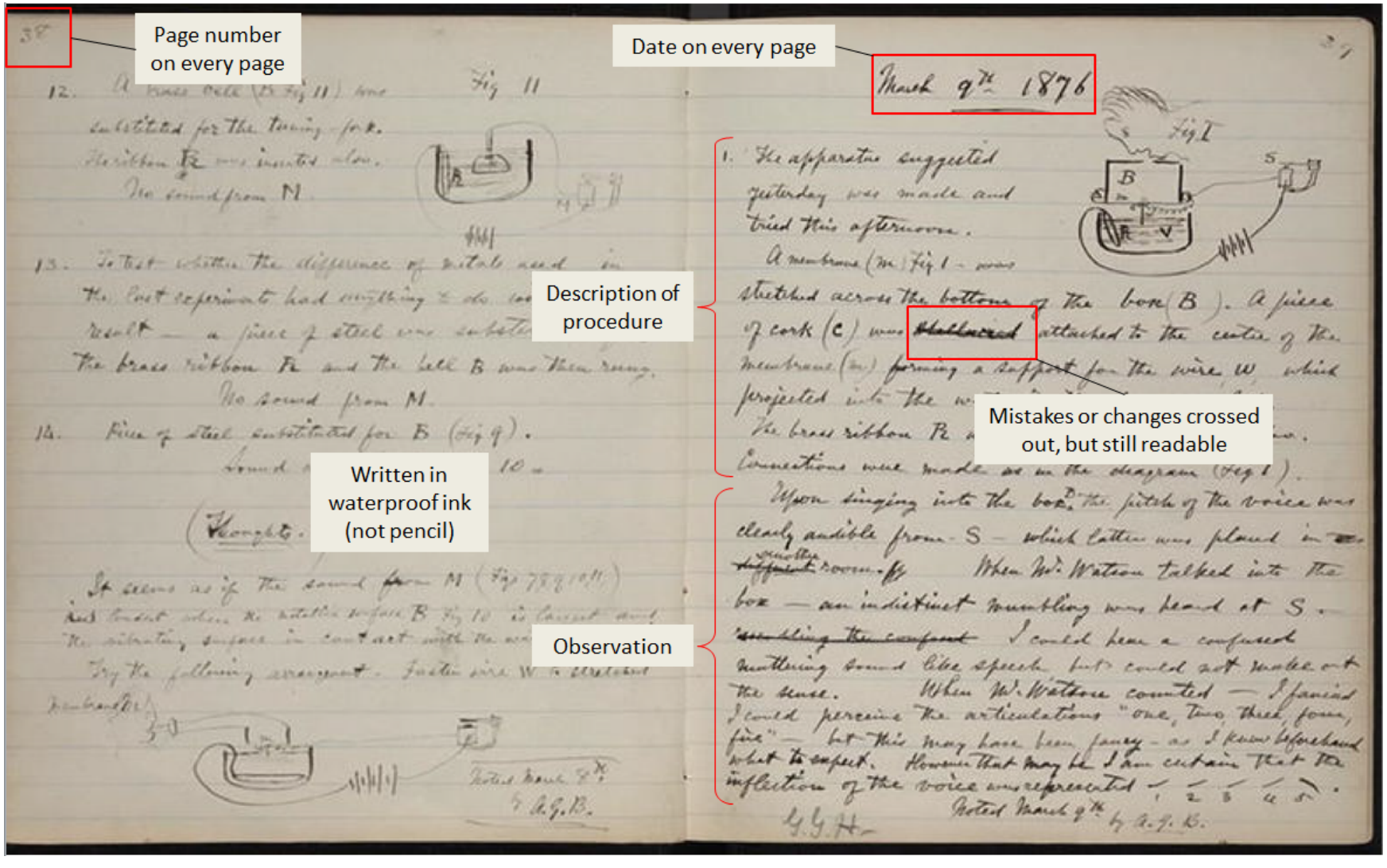


Image credit

Example from a (poor) research log

Any format is fine (text, Word, markdown). Requires discipline, but pays dividends later.

```
### 13 May 2018

* Set up as r-project. Collapsed all FARS data for each year 1975-2016 to zipped file, to be extracted during Stata run to
create analytic dataset, then deleted.
* Need to integrate data management with Stata but cannot figure out how to run a more complex .do file, so must resort to
running this bit in Stata and doing the rest of the analysis in R
* However, the package *RStata* does seem useful for running Stata from *within* R, which may prove very useful. A simple
example below:
```` {r RStata example}
library(RStata)
options("RStata.StataVersion" = 14)
options("RStata.StataPath"= '/Applications/Stata/StataMP.app/Contents/MacOS/stata-mp')
x <- data.frame(case = c(1356,2444), exp = c(1,0), pt = c(1,2))
stata("ir case exp pt", data.in = x)
````


14 May 2018

* Still not quite sure what to do about "special days" that are celebrated on different dates each year (Memorial Day, Labor
Day, Thanksgiving). Could potentially recode these and include as separate coefficients, similar to what can be seen in the
code [here](http://research.cs.aalto.fi/pml/software/gpstuff/demo_births.shtml) by collaborators of Gelman and the "birthday
problem" modeling births across days of the year:
```

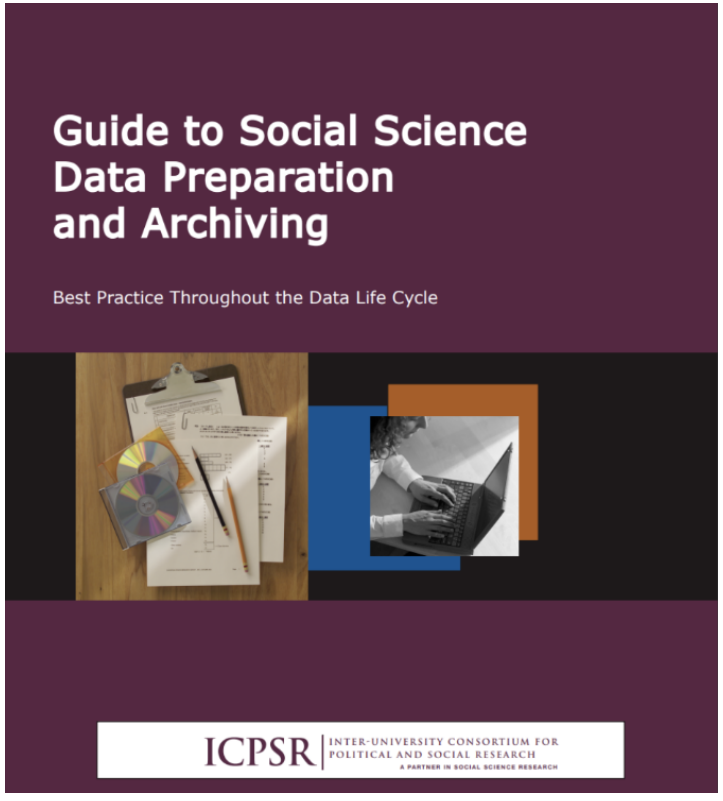
Codebooks

Codebook is crucial, especially if you are collecting your own data.

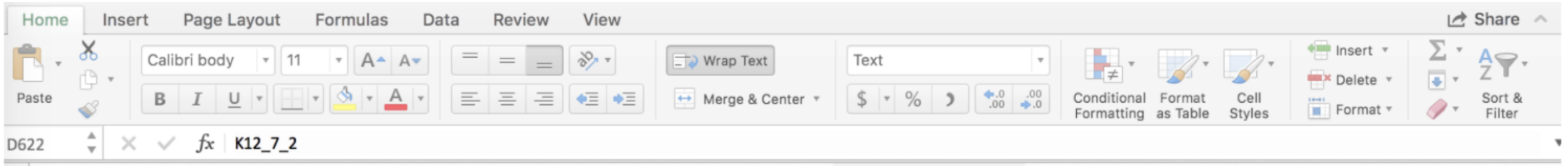
Should include:

- Variable name and question number.
- Survey questions.
- Skip patterns.
- Descriptive statistics.
- Missing values and reasons.
- Data imputation.

Good advice 👇



Codebook example (Excel)



	A	B	C	D	E	F	
1	**WAVE**	**QUESTION**	**QUESTION_TEXT**	**VARIABLE NAME**	**SECTION**	**POSSIBLE RESPONSES**	**NOTES**
2	*Wave number*	*Question number from survey*	*Exact text from the survey*	*Variable name from dataset*	*Section in the survey*	*List of response options*	*Please here, a (YYYY*
3	1	n/a	n/a	Interviewer_Name		n/a	RR 13 some a
4	1	A.1	Name of respondent	A1_1	Household characteristics	N/A	
5	1	A.2	Respondent ID number	id	Household characteristics	Integer, assigned to each respondent	AN 201 respon is actua Since A each ch househ only sa
6	1	A.3	Birthdate	A3_1_1 A3_2_1 A3_3_1	Household characteristics	Continuous date, month, and year	RR 201 variable reporte
7	1	A.4	How old are you?	A4_1	Household characteristics	Continuous number of years	RR 201
8	1	A.5	Have you ever attended school?	A5_1	Household characteristics	(1) Yes (2) No	

Codebooks can also be automated (using R 📦 `codebook`)

Import dataset: `codebook_data <- rio::import("fakedata.dta")`
Generate codebook: `codebook(codebook_data)` 👇

Codebook table

Copy CSV Excel PDF Print

Search:

name	label	type	type_options	data_type	value_labels	optional	scale_item_names	item_order	n_missing	complete_rate
session				character					0	1
created	user first opened survey			POSIXct					0	1
modified	user last edited survey			POSIXct					0	1
ended	user finished survey			POSIXct					0	1
expired				logical					28	0

Screenshot

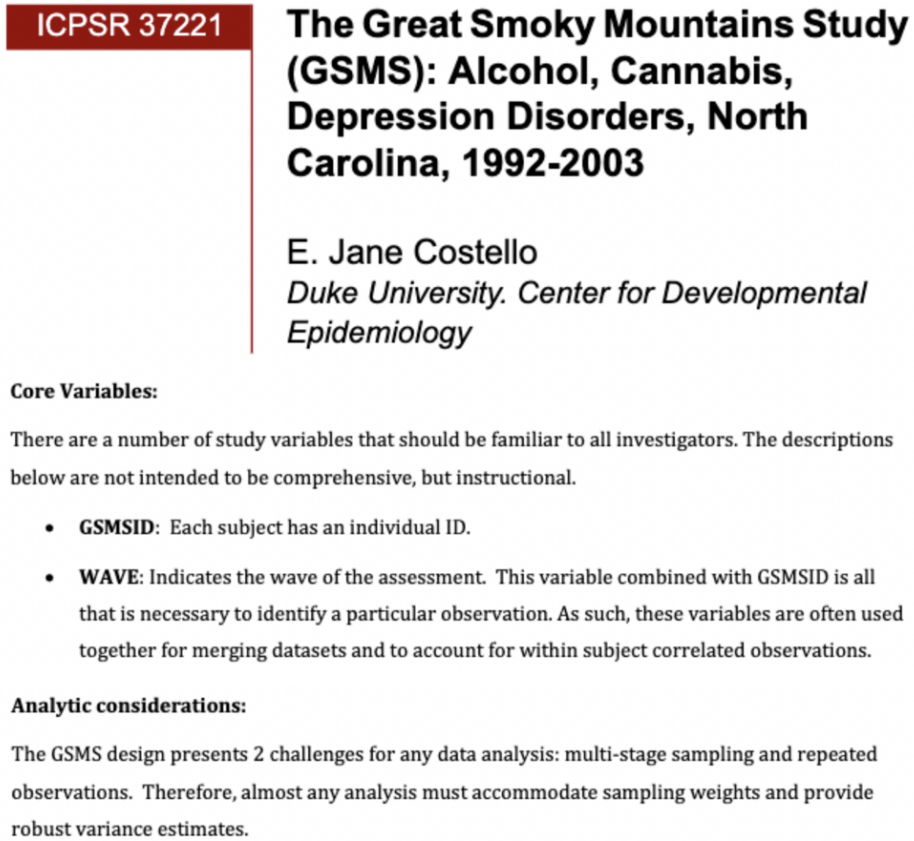
Dataset documentation

Why?

Provides key details about who, what, where, and which processes led to the created datasets.

How?

Ideally, using a reproducible format that can be updated regularly.



ICPSR 37221

The Great Smoky Mountains Study (GSMS): Alcohol, Cannabis, Depression Disorders, North Carolina, 1992-2003

E. Jane Costello
Duke University. Center for Developmental Epidemiology

Core Variables:

There are a number of study variables that should be familiar to all investigators. The descriptions below are not intended to be comprehensive, but instructional.

- **GSMSID**: Each subject has an individual ID.
- **WAVE**: Indicates the wave of the assessment. This variable combined with GSMSID is all that is necessary to identify a particular observation. As such, these variables are often used together for merging datasets and to account for within subject correlated observations.

Analytic considerations:

The GSMS design presents 2 challenges for any data analysis: multi-stage sampling and repeated observations. Therefore, almost any analysis must accommodate sampling weights and provide robust variance estimates.

One simple way: use a registry

Create a directory containing meta-data on each dataset created for the project.

Links directly to time-stamped source files that create data.

Dataset registry for `myproject`
Created by: NAME1
Last modified by: NAME2

Dataset	Filename	Date	Source	Comments
1	wave1.dta	2020-09-01	01-build-w1.do	Fix missing
2	wave2.dta	2020-10-05	02-build-w2.do	Ready to merge
...	...	...	...	...

3. Analytic Solutions

3.1 Workflow Management

3.2 Documentation

3.3 Literate Programming

3.4 Version Control

3.4 Dynamic Documents

"Literate" programming is:

Code that humans understand.

Computers will get it.

Core ideas for coding practices

Don't modify data by hand

This means no Excel for analysis, and no copy/paste.

Don't point-and-click

Typing commands or graphical interfaces are not reproducible.

Write code scripts

Coding creates reproducible workflow.

More general coding advice

Comment extensively!

Short lines with no wrapping

Give functions, objects, and variables intuitive names:

- `edu_percent` 👍
- `v76` 👎

Label variables and values (remember codebook advice).

Avoid abbreviations for commands (Stata).

Consider breaking long files into several shorter ones.

Essential parts of a syntax file

This example is for Stata, but the same principles apply to any software.

```
capture log close
log using _name_, replace text

//  program:    _name_.do
//  task:
//  project:
//  author:     _who_ \ _date_

//  #0
//  program setup

version 14
clear all
set linesize 80
macro drop _all

//  #1
//  describe task 1

//  #2
//  describe task 2

log close
exit
```

Generates a record of the session

Purpose, project, who made this file

Note version, since algorithms change

Better looking log

What is this bit of code doing?

And this one?

Same idea for SAS

01-processing.sas

CODE LOG RESULTS

```
/*
THIS DO-FILE OPENS THE DATA FROM THE
HARVARD SCHOOL OF PUBLIC HEALTH COLLEGE
ALCOHOL STUDY (ICPSR 4291),

THEN PROCESSES THE DATA TO PREPARE THEM
FOR ANALYSIS,

THEN SAVES THE PROCESSED DATA IN A FILE
CALLED analysis.dta

*WHEN YOU RUN THIS DO-FILE, MAKE SURE THAT
**********1) SAS's WORKING DIRECTORY IS SET
**********TO THE "Command-Files" FOLDER
**********2) THE HARVARD ALCOHOL STUDY DATA FILE
**********04291-0001-Data.dta IS SAVED IN THE
**********"Original-Data" FOLDER
*/

libname original  "/home/sam.harper/Original-Data";
libname analysis "/home/sam.harper/Analysis-Data";
filename ogdata "/home/sam.harper/Original-Data/04291-0001-Data.stc";

/* #1
  Read in the original dataset */

*import the original data;
proc cimport
  infile = ogdata
  library =  work
  ISFILEUTF8 = TRUE;
run;
```

Similar for R, all are adaptable

When commenting *your* code, consider how you feel when you read someone *else's* code!

```
# PROJECT
  # Paper:
  # Authors:

# R Script
  # Purpose:
  # Created: <date> by <author>
  # Updated: <date>
  # Inputs: <files required>
  # Outputs: <tables and figures>

######## PACKAGES ########
# Check system for packages
  need <- c("dplyr", "foreign") # list packages
```

```
######## ANALYSIS ########
# First fit crude model
fit1 <- lm(...)

# Now add sets of confounders
fit1 <- lm(...)

# Non-linear model for binary outcome
fit3 <- glm(...)

######## ROBUSTNESS ########
# Exposure measured with error (95% Se)
# Quantitative bias analysis using
# parameters from Smith et al. (2014)
fit_corrected <- lm(...)
```

Alignment, indentation, automation are your friends...

Harder to read

```
keep sid class_id teacher grade1 ///
grade2 grade3 pass
```

Easier to read

```
keep   sid class_id teacher grade1 ///
       grade2 grade3 pass
```

Without automation

```
//  create binary variables without loop
generate y_lt2 = y<2 if !missing(y)
generate y_lt3 = y<3 if !missing(y)
generate y_lt4 = y<4 if !missing(y)
```

With automation

```
//  create binary variables using a loop
foreach cutpt in 2 3 4 {
    gen y_lt`cutpt' = y<`cutpt' if !missing(y)
  }
```

Save outputs as objects to insert into papers

R:

```r
# a boring regression
lm(dist ~ 1 + speed, data = cars)

# save regression results for later
fit <- lm(dist ~ 1 + speed, data = cars)
```

Note that `fit` contains our coefficients and SEs. Now write to a table:

```r
library(broom) # need broom pkg
write.csv( tidy( fit ), "t1.csv")
```

Now `t1.csv` has estimates.

Stata:

```
webuse auto, clear

// another boring regression
regress price mpg
estimates store m1
```

Here `m1` contains our model results. Write to a table:

```
// need eststo package
esttab m1 using t1.csv
```

Now `t1.csv` has estimates.

Can be adapted to other software

Key idea:

- Results are *static* tables.
- Inserted into compiled manuscript.
- No copy/paste.

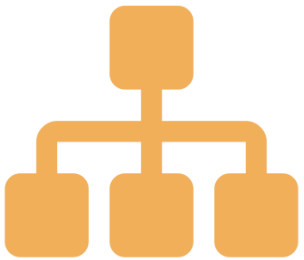
Example using SAS's **O**utput **D**elivery **S**ystem (ODS):

```
data etoh;
set analysis.analysis;
run;

ods select ParameterEstimates;
ods csv file = "/home/sam.harper/Output/model.csv";
proc logistic data=etoh;
model drunk = free;
run;
ods csv close;
```

Results written to "model.csv" file

Structure and name files intuitively.



Give code, data files, and output logical names where possible.

Number scripts sequentially in the order they should be run, e.g.,:

- 01_clean_data.R (or .do, .sas, etc.)
- 02_main_analysis.R
- 03_robust_checks.R

Label output figures with descriptive names, but ones that aren't likely to change (e.g., figure_hte.png is better than figure_1.png)

See Jenny Bryan's talk on naming things.

"One script to rule them all"

Many papers require multiple scripts.

Dependencies create problems.

Create a master-script to run everything.

Facilitates '1-click' reproduction.

Example of a "master" script (Stata)

This is the content of `code/mvc-laws-master.do`:

```
capture log close master
log using "code/mvc-laws-master", name(master) replace text

//  program:  mvc-laws-master.do
//  task:     run all analyses
//  project:  mandatory seat belt laws and traffic deaths
//  author:   sam harper \ 23feb2017

// Required user-written programs: -tabout, -metan, -estout, -egenmore
// Can be downloaded by typing "ssc install tabout, replace"  and
// "ssc install metan, replace", etc. into the Stata command line


do "code/mvc-laws-data.do"                    // create analytic dataset
do "code/mvc-laws-analysis-descriptives.do"   // descriptive table
do "code/mvc-laws-analysis-models.do"         // model results
do "code/mvc-laws-analysis-appendix.do"       // appendix tables/figures

log close master
exit
```

Your turn #2: Let's create a master script.

1) Open the `myproject` you created in R or Stata and create a new script (File>New File> R Script or File> New > Do-file. Type the following code:

- R: `print("Hello world!")`
- Stata: `disp "Hello world!"`

2) Save it as `1-hello.R` or `1-hello.do` in the "code" folder.

3) Repeat, creating another new script for `2-goodbye.R`, but type `"Goodbye world!"` instead of `"Hello world!"`

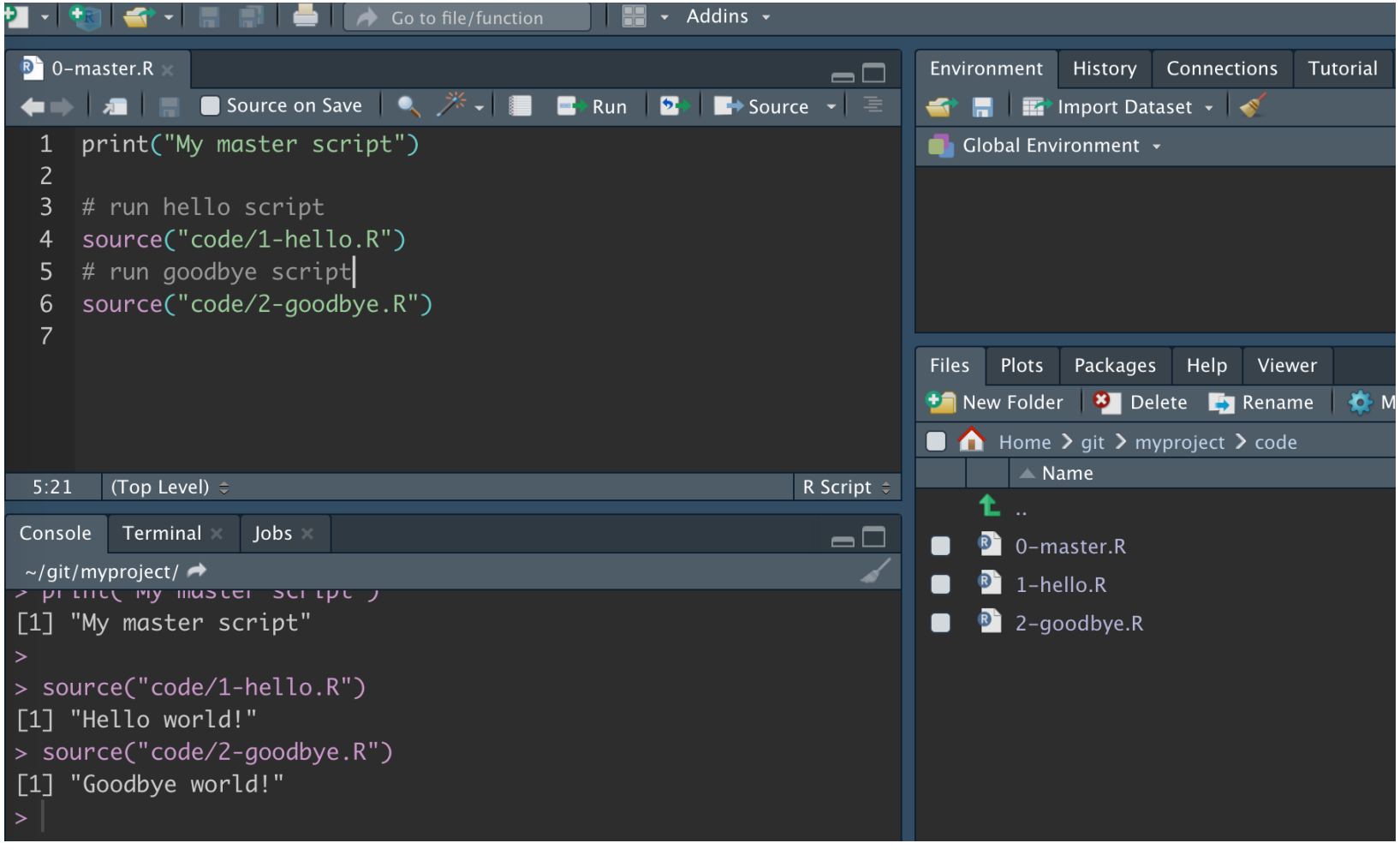
Now create a master script called `0-master.R` or `0-master.do` that will run the two other scripts you created. The command to execute another file is `source` in R and is `do` in Stata.

4) Run the master script.

- Success? ✅ if yes, 🛑 if no

05:00

Something like?



Break! ☕

10:00

3. Analytic Solutions

3.1 Workflow Management

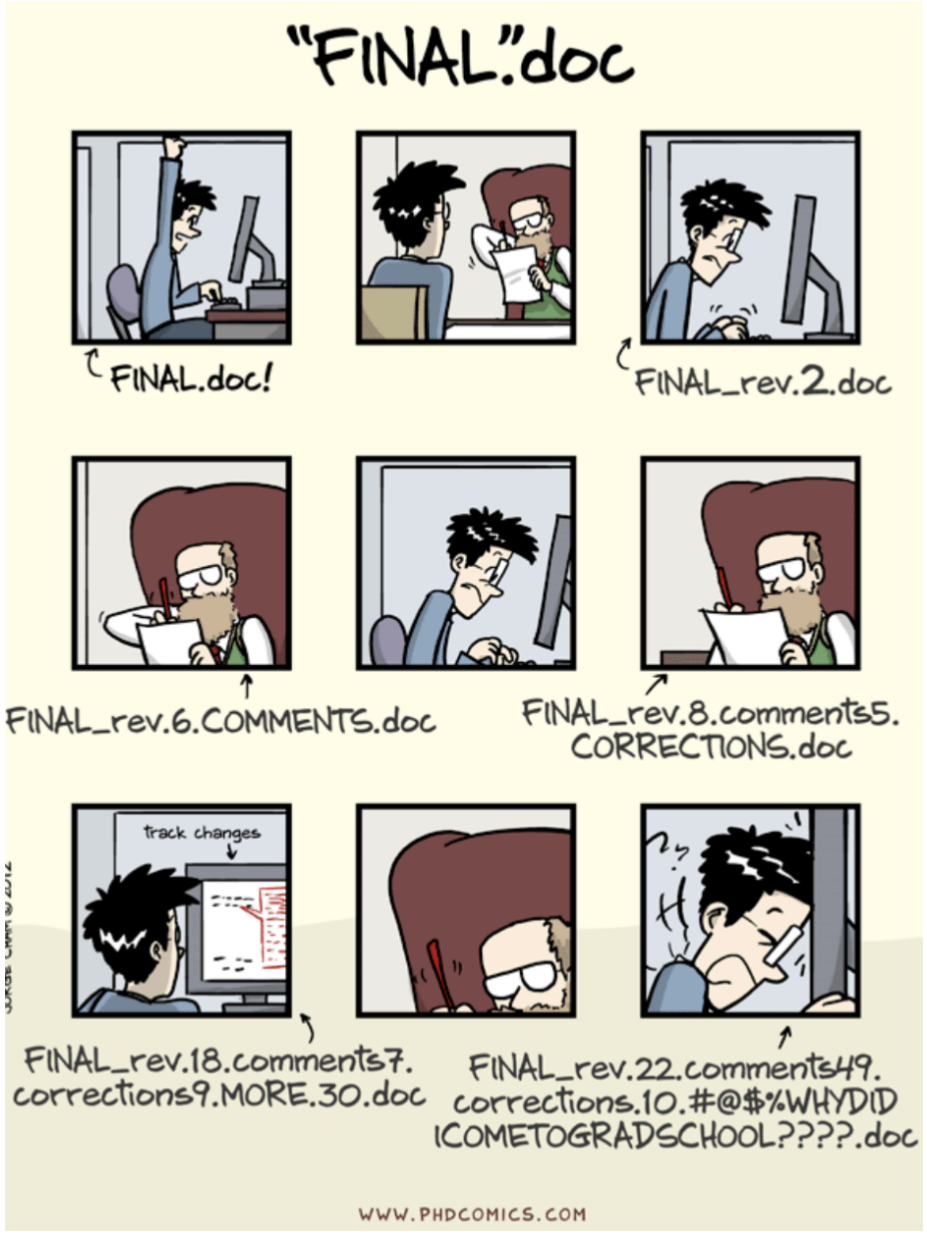
3.2 Documentation

3.3 Literate Programming

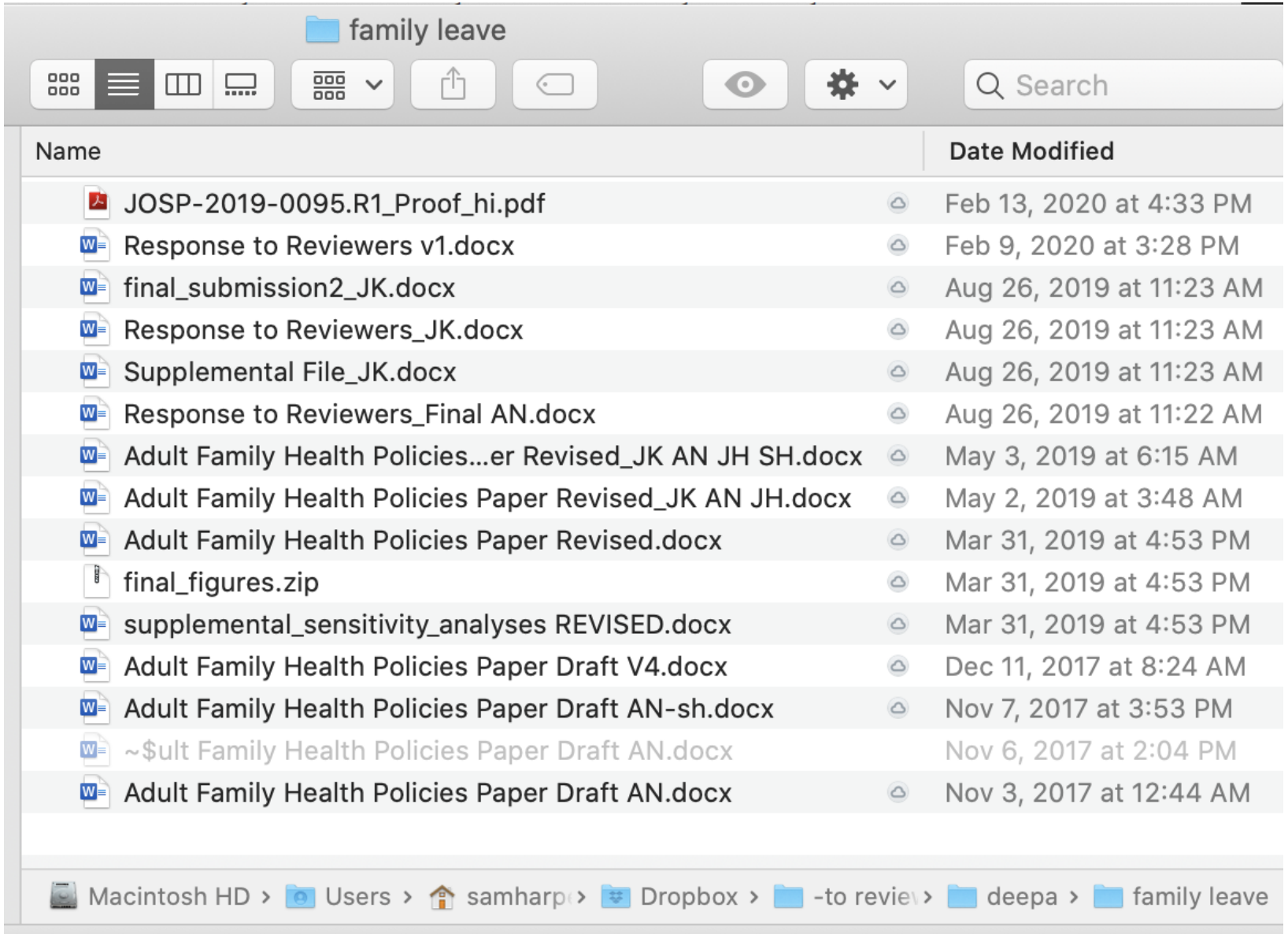
3.4 Version Control

3.4 Dynamic Documents

Obligatory picture for every discussion of version control.



But not a joke 😞



One kind of collaborative workflow

Collaborator 1:

- cleans data → `data.do`
- analysis → `analysis.do`
- generates Excel tables → `results.xls`
- draft manuscript in Word → `manuscript.docx`

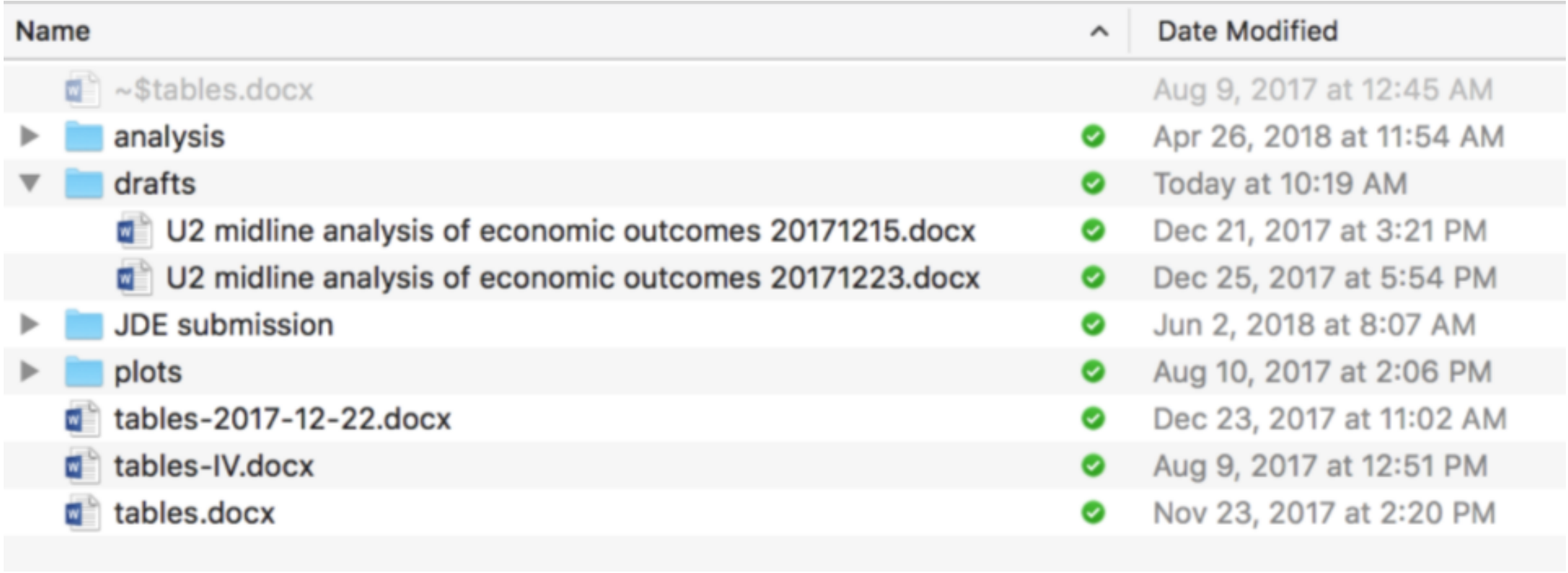
Collaborator 2:

- adds new variables → `data_new.do`
- new analysis → `analysis2.do`
- new tables → `results_final.xls`
- revises draft with new results → `manuscript-v2.docx`

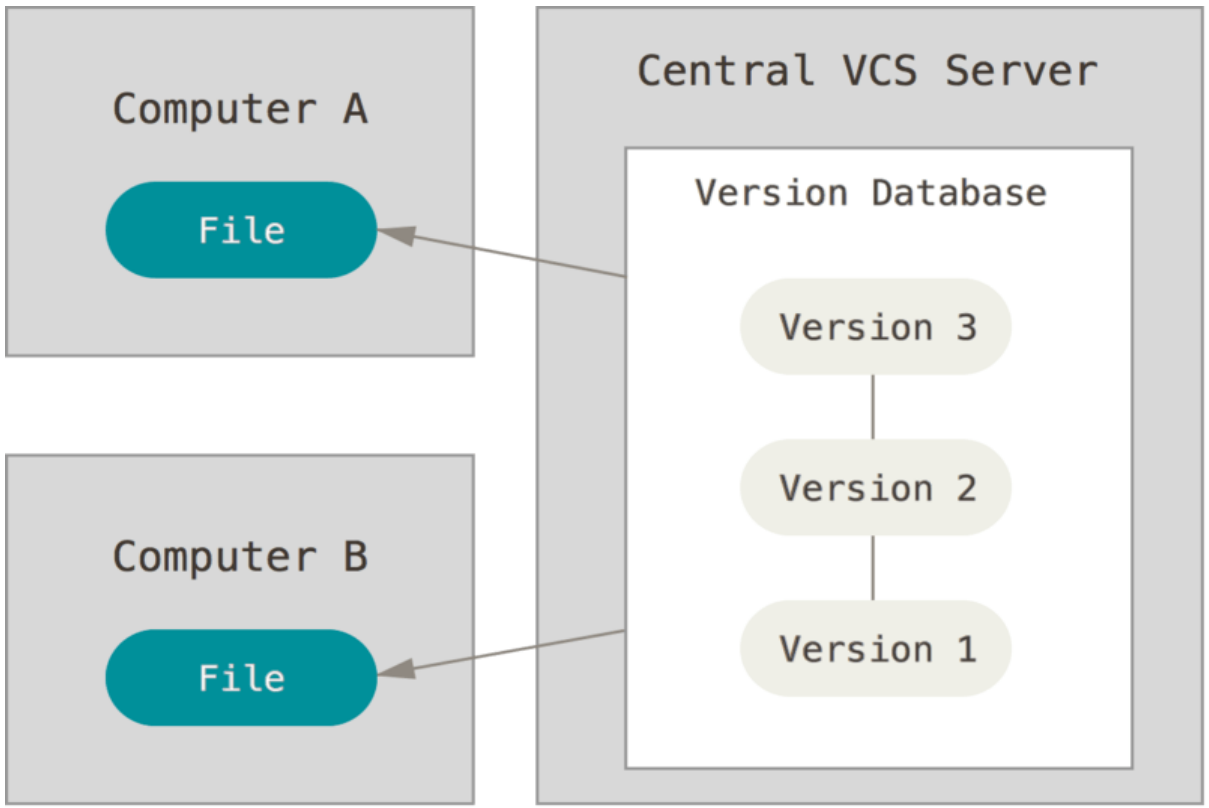
This can work! But it can also create problems.

What is the problem?

Which draft is the current one? Which set of tables contain the "right" results?



A version control system (VCS) automatically keeps track of changes to files and code.



How can version control help?

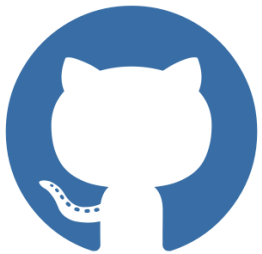
Under VCS the prior collaboration would only have 4 files:

- cleans data → data
- analysis → analysis
- generates results → results
- generates manuscript → manuscript

A VCS keeps track of changes to each file.

We just don't usually see it.

How can version control help?



With version control you can:

- Collaborate
- Track each change and **who** made it
- Easily switch between versions of files (i.e. go backward in time)
- Compare versions of files
- Backup
- Work with the same files on different machines
- Collaborators can work simultaneously on the same files on different machines
- Experiment with a new version of code without permanently ruining anything

Manual solutions (works, but not ideal)

Create dated versions of files (save-as) for each substantive change

With each modification, re-run ALL code to make sure nothing is broken—helps if you have a master file to run all scripts!

Check-in with coauthors to ensure multiple people aren't working on the same files at the same time.

Keep a simple log to remind yourself of the location/content of major changes.

Software solutions

Version control software: helps manage versions and edits to files.

Git: Open-source, “distributed model” of version control developed by creator of Linux.

GitHub / GitLab: Free, web-based service that hosts Git “repositories” and offers a variety of features for collaboration:

- Online
- Desktop apps
- Command line (more technical)

How does Git work?

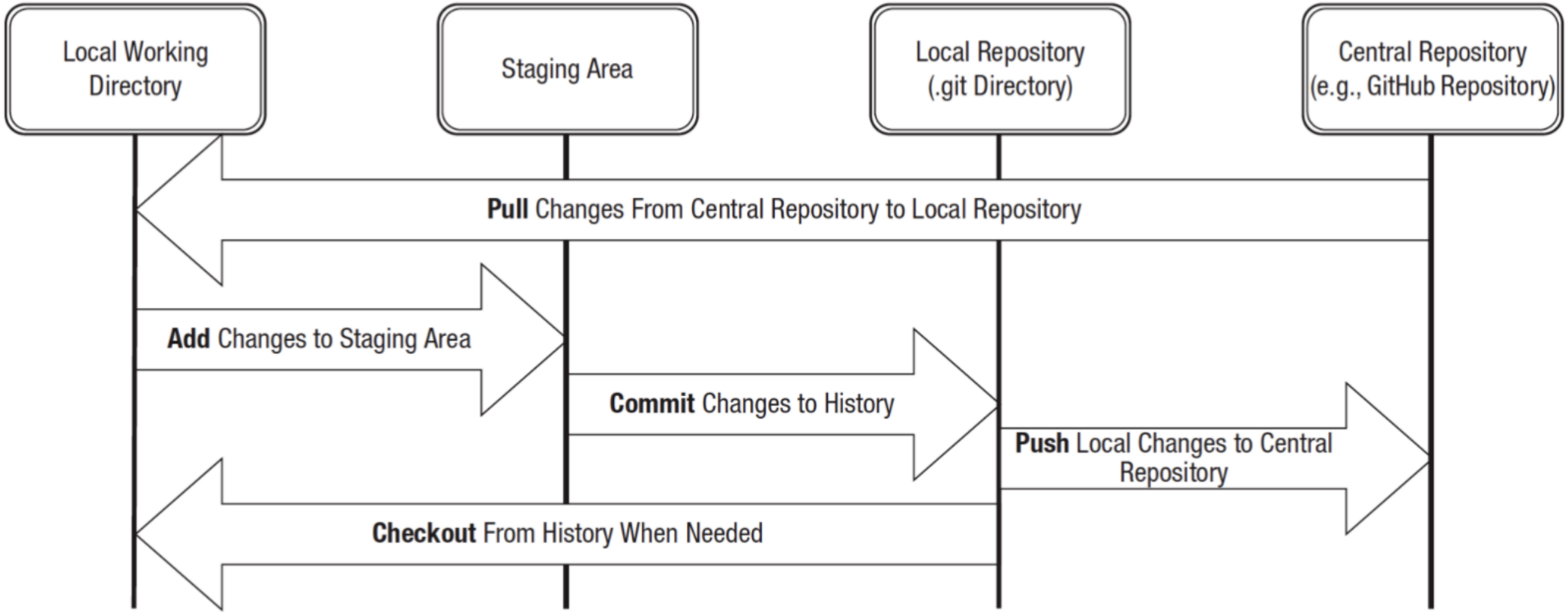


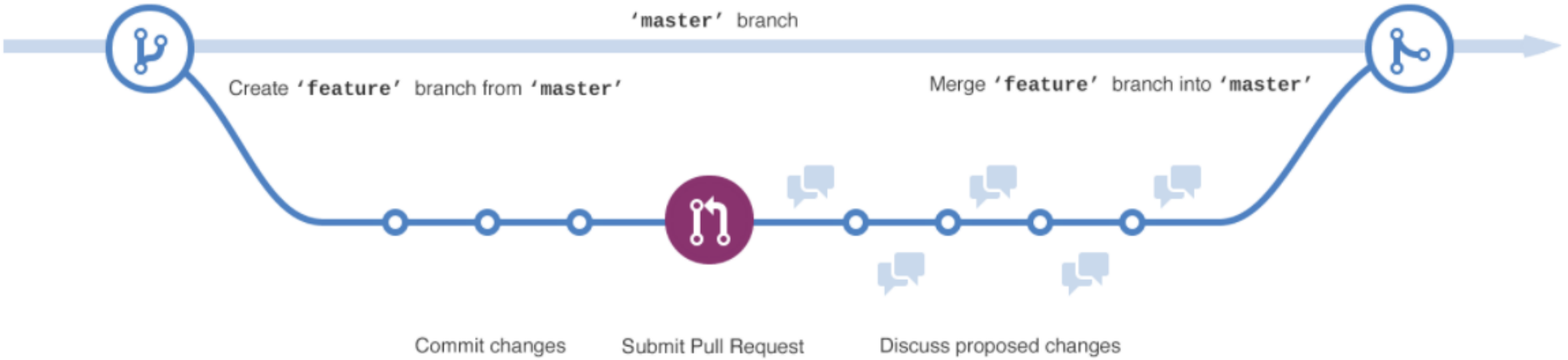
Fig. 9. A diagram illustrating the typical collaborative Git workflow with a central repository (e.g., GitHub). As when a central repository is not involved, users work in their local repositories, making changes to files and viewing prior versions as needed. In addition, they push changes from their local repositories to the central repository, so that collaborators have access to them, and pull changes from the central repository to their local repositories, to see their collaborators' work. Verbs in boldface are Git operations and explained in the main text.

Advantages of Git

Provides the entire narrative of your project.

Allows you to go back in time.

Experiment without breaking things.

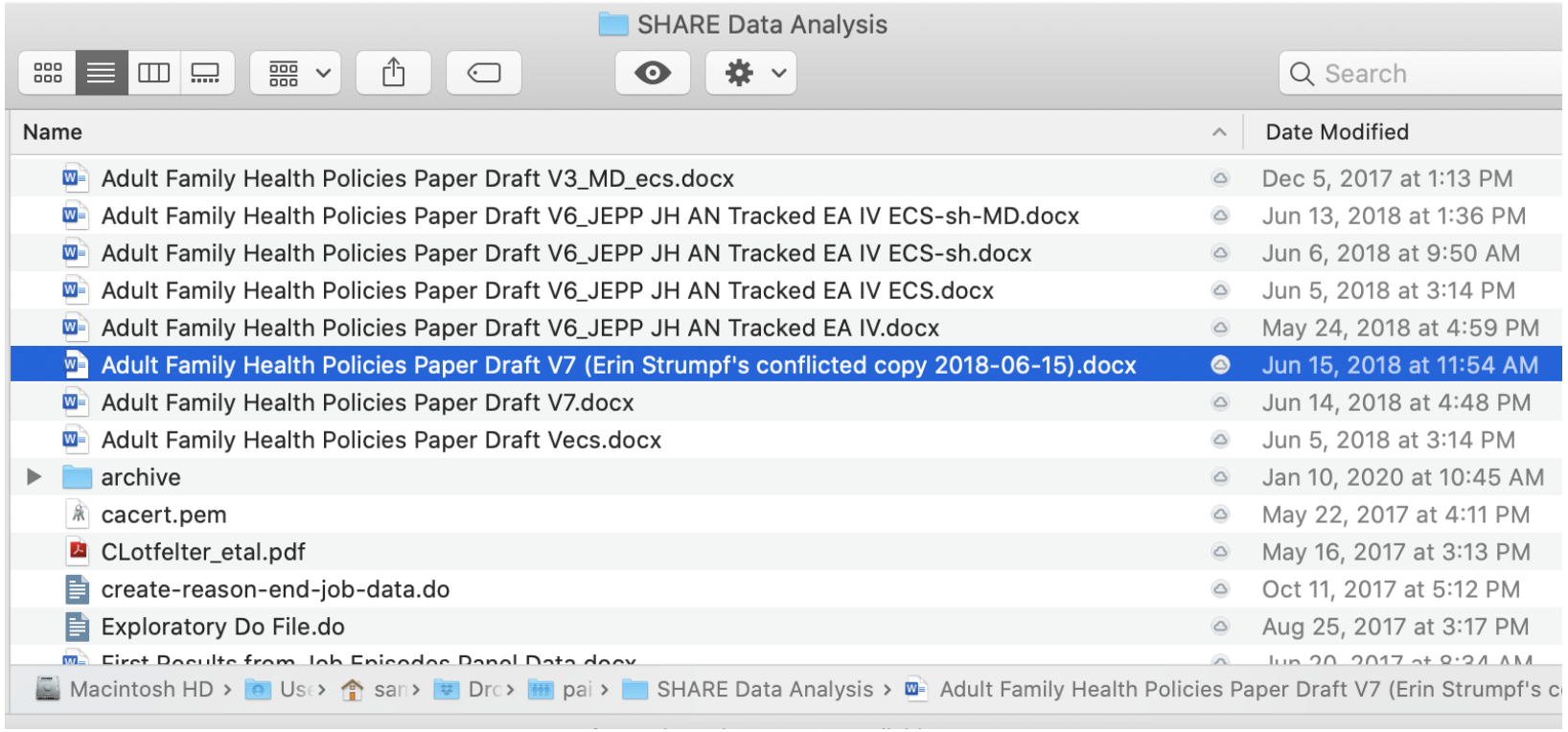


How is that better than Dropbox?

Simultaneous editing in Dropbox leads to "conflicted" copies of files.

Only one person can edit the "live" version of the file.

Can "rollback" to earlier version, but that affects everyone.



Additional resources for Git

- The Basic Workflow of Git (an infographic explaining how Git's version control system works): https://www.git-tower.com/blog/workflow-of-version-control
- Git + GitHub (information on using Git and GitHub in an R programming context): http://r-pkgs.had.co.nz/git.html
- GitHub's Git cheat sheets (reference sheets on the most commonly used Git commands)
- GitHub Glossary (a glossary of Git and GitHub terminology)
- Pro Git (Chacon & Straub, 2014; a complete manual of Git)
- tryGit (an interactive Web site for learning the basics of Git)
- Jenny Bryan's HappyGit

Vuorre & Curley (2018)

3. Analytic Solutions

3.1 Workflow Management

3.2 Documentation

3.3 Literate Programming

3.4 Version Control

3.4 Dynamic Documents

What are dynamic documents?

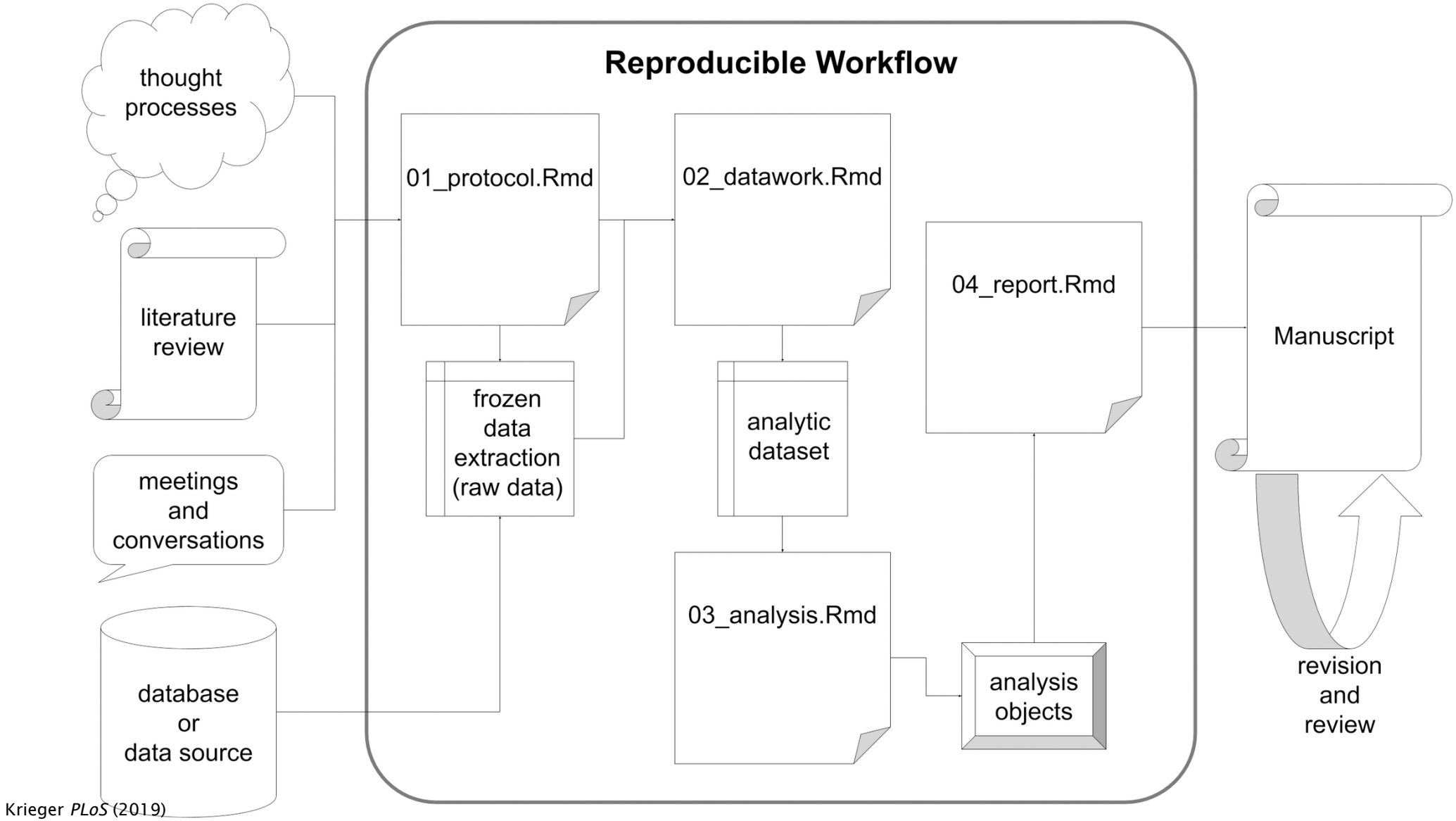
Even with perfect (version controlled) code, you can still run into problems going from your code to paper. (copy → paste problems)

This is where dynamic documents come in.

A dynamic document includes your data, code, analysis, and output all in one place.

Fully automated so no mistakes from copying and pasting.

Several formats, but most commonly using R Markdown in RStudio or Markdoc in Stata.



Krieger *PLoS* (2019)

How can dynamic documents help?

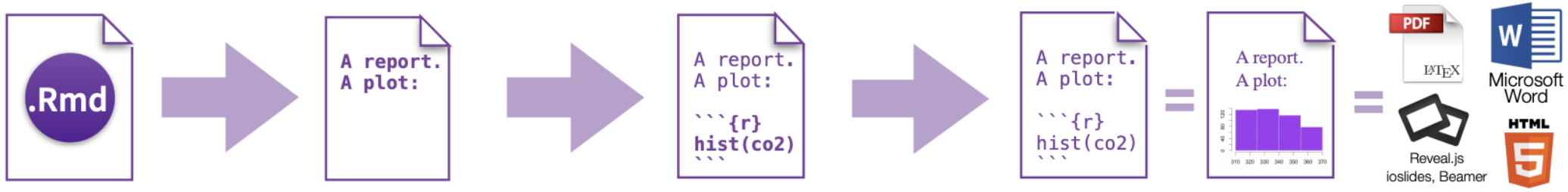
- Include tables by linking to a file, instead of a static image.
- Reference an estimate (e.g. risk ratio) by linking to a value calculated by an analysis file, instead of a static number typed manually.
- Automatically update tables and numbers if models and/ or data changes.
- Produce entire paper with one or two clicks.
- Most written using markdown language.

i. Open - Open a file that uses the .Rmd extension.

ii. Write - Write content with the easy to use R Markdown syntax

iii. Embed - Embed R code that creates output to include in the report

iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.



	Markdown	Result

Headers	`# Header 1`
	`## Header 2`
	`### Header 3`
	`#### Header 4`
	`##### Header 5`
	`###### Header 6`
		Header 1
		Header 2
		Header 3
		Header 4
		Header 5
		Header 6

Text	`*italic* _italic`
	`**strong** __strong__`
	`***italic & strong*** ___i&s___`
	`~~strikethrough~~`
		italic
		strong
		italic and strong
		~~strikethrough~~

Links	`https://arminreiter.com`
	`[MyLink](https://arminreiter.com)`
	`[MyLink](https://arminreiter.com "Title")`
	`[MyLink][1]`
	`[MyLink][URL to AR]`
	`See my [link]`
	
	`[1]: https://arminreiter.com`
	`[URL to AR]: https://arminreiter.com`
	`[link]: https://arminreiter.com`
		https://arminreiter.com
		MyLink
		MyLink
		MyLink
		MyLink
		See my link

Images	`![Logo](/images/logo.png)`	

Example of a whole paper written and submitted with Rmarkdown

Stata catching up

Since Stata v15, now the `dyndoc` command can produce Word or HTML documents from Markdown.

More limited functionality than `markstat` but will likely improve.

Creating a **dyndoc** file is as easy as creating a do-file. Here's one that creates an HTML file with only Stata output:

```
──────────────────────────────────────────── example1.txt ────
~~~~
<<dd_do>>
webuse auto, clear
summarize price
<</dd_do>>
~~~~
──────────────────────────────────────────────────────────────
```

The four tildes in a row, **~~~~**, are Markdown syntax to start and end a code block.

Terms in **<<** … **>>** are called Stata dynamic-document tags. The code block is bounded by **<<dd_do >>** … **<</dd_do>>**. Stata code inside **<<dd_do>>** … **<</dd_do>>** is executed and its output substituted into the HTML document. We merely save the file above as **example1.txt**, type **dyndoc example1.txt** in Stata, and **example1.html** is created for us:

```
. webuse auto, clear
(1978 Automobile Data)

. summarize price

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
```

Main elements of a dynamic document

Header information

Document metadata (author, title, date, formatting options).

Document text

Write using markdown language (incl. equations, refs)

Code: both 'inline' and as 'chunks'

Performs data maniupation, analysis, tables, figures.

Header (also called YAML*)

Rmarkdown:

```
---
title: "My Reproducible Paper"
author: Sam Harper
date: 2020-10-30
{{output: pdf}}
bibliography: myrefs.bib
csl: vancouver.csl
---
```

Other output formats available (e.g., .html or .docx output), many other formatting options.

Same structure for Stata Markdown:

```
---
title: "My Reproducible Paper"
author: Sam Harper
date: 2020-10-30
abstract: |
  I give a brief summary here.
keywords: |
  markdown, reproducible research.
bibliography: myrefs.bib
---
```

Document format specified in another step.

*YAML stands for YAML Ain't Markup Language. Ha.

Document text

Rmarkdown:

```
---
header:
---
```

Text written using markdown language:

```
# Introduction
This is why this paper is necessary, and here
are the gaps it will fill.

# Methods
## Data
We used some cool data.
```

Same structure for Stata Markdown:

```
---
header:
---
```

Can add references or equations as well:

```
## Statistical analysis
We ran a linear regression [@Galton:1875]:

$$
y_{it}=\beta_{0} + \beta_{1}*X
$$

where $ \beta_{1} $ is what matters.
```

Adding inline or 'chunks' of code

Rmarkdown:

```
---
header:
---
```

Document text including code 'chunk', set apart by 3 backticks:

````
```r{eval=FALSE}
fit1 <- lm(y ~ x, data=d)
```
````

Inline code: single backtick `` `r 1+1` `` .

Structure for Stata Markdown:

```
---
header:
---
```

Same idea for code 'chunks':

````
```s
// Write stata code inside chunks
sum mpg
```
````

Inline: single backtick `` `s r(mean)` `` .

Allow text to update automatically when results change.

Rmarkdown:

Header + document text followed by code chunk:

````
```r{eval=FALSE}
fit1 <- lm(y ~ x, data=d)
est <- summary(fit1)$coefficients["x","Estimate
```
````

We find x increases y by `` `r est` `` .

Will update if data or estimates change.

Structure for Stata Markdown:

Header + text with Stata code chunks:

````
```s
reg price mpg
```
````

Mpg increases price by `` `s _b[mpg]` `` with a standard error of `` `s _se[mpg]` `` .

These are dynamic values.

Stata markdown example (start at project directory)

Stata do-file (`code/paper.do`) calls the markdown file using the `markstat` command:

```
// change to writing directory
cd writing


// generate the manuscript
markstat using "paper.stmd", pdf

// back to the main directory
cd ..
```

Markdown text (`writing/paper.stmd`)

````
---
title: My New Paper
author: Sam Harper
date: 2020-10-28
---

# Introduction

We did some stuff. And made a plot

```s/
    quietly webuse auto, clear
    quietly scatter weight length, legend(off)
    quietly graph export scatter.png, width(800) replace
    quietly sum price
    scalar mp = round(r(mean),1)
```

![Correlation between weight and length](scatter.png)

We found the mean price was `s mp`.
````

My New Paper

Sam Harper

2020-10-28

Introduction

We did some stuff. And made a plot

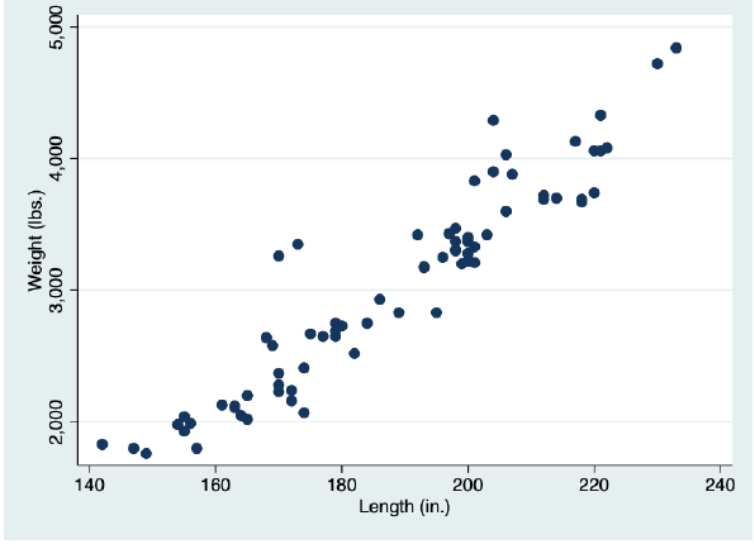


Figure 1: Correlation between weight and length

We found the mean price was 22.

Knit Insert Run

```
---
title: "My SAS Example"
author: "Sam Harper"
date: "26/10/2020"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(here)
library(kableExtra)
```

Results

We ran a logistic model:

```{r model, echo=F, message=FALSE}
t1 <- read_csv(here("slides/03-analytic", "model.csv"), skip=2,
  col_names=c("Parameter", "df", "Estimate", "SE", "X2", "pvalue"))
kable(t1, format = 'latex', digits=2,
      caption = "Logistic Estimates", booktabs = T) %>%
      kable_styling(latex_options = "hold_position")
```

Results look good!
```

Can be adapted to other software

Rmarkdown options, packages required

Here are your model results

Rmarkdown + results from SAS

1. Use SAS to fit models, generate results.
2. Export results to files.
3. 'Knit' together with text using Rmarkdown.

My SAS Example

Sam Harper

26/10/2020

Results

We ran a logistic model:

Table 1: Logistic Estimates

Parameter	df	Estimate	SE	X2	pvalue
Intercept	1	0.79	0.05	257.07	<.0001
free	1	0.00	0.09	0.00	0.9574

Results look good!

Break! ☕

10:00