PLOS | COMPUTATIONAL BIOLOGY

PERSPECTIVE

Good enough practices in scientific computing

Greg Wilson[1]☯*, Jennifer Bryan[2]☯, Karen Cranston[3]☯, Justin Kitzes[4]☯, Lex Nederbragt[5]☯, Tracy K. Teal[6]☯

1 Software Carpentry Foundation, Austin, Texas, United States of America, **2** RStudio and Department of Statistics, University of British Columbia, Vancouver, British Columbia, Canada, **3** Department of Biology, Duke University, Durham, North Carolina, United States of America, **4** Energy and Resources Group, University of California, Berkeley, Berkeley, California, United States of America, **5** Centre for Ecological and Evolutionary Synthesis, University of Oslo, Oslo, Norway, **6** Data Carpentry, Davis, California, United States of America

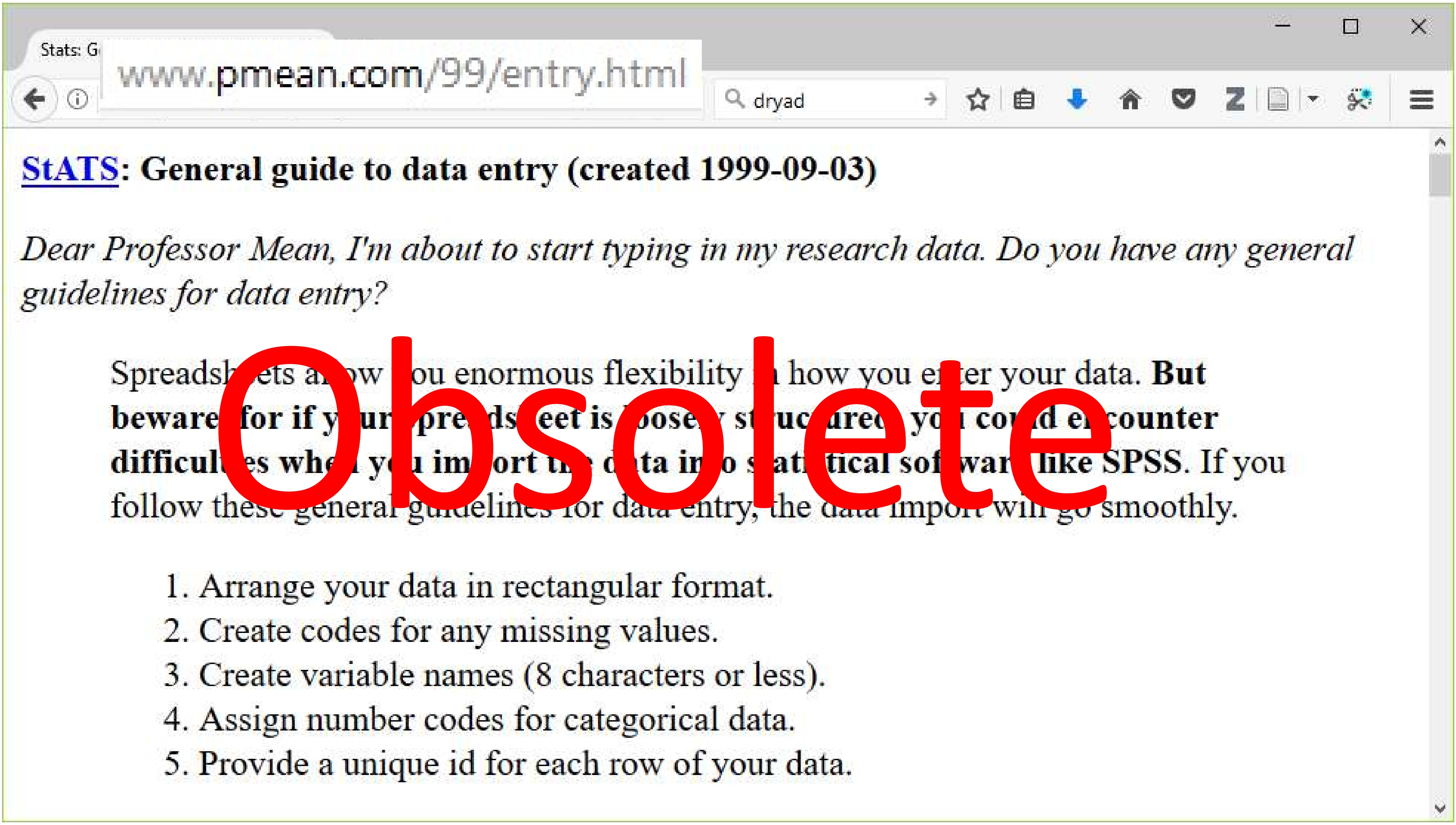
☯ These authors contributed equally to this work.
* gvwilson@software-carpentry.org

journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510

Abstract

- Many of the tools and methods proven to work in the field of software development can also help you with your data analysis. Following closely the recommendations in Wilson et al “Good enough practices in scientific computing” published in PLoS Computational Biology, you will learn how to apply modern techniques that will improve your ability to work with the other members of your research team, help to attract new collaborators, and insure the production high quality reproducible results. These techniques require that you break long entrenched habits, but you can adopt them no matter what your level of computer expertise.

- Plagiarism disclaimer. All the major headings (1. Data management) and first level subheadings (Save the raw data (1a)) are direct quotes from the paper. Second level subheadings are paraphrases or independent observations.



StATS: General guide to data entry (created 1999-09-03)

Dear Professor Mean, I'm about to start typing in my research data. Do you have any general guidelines for data entry?

Spreadsheets allow you enormous flexibility in how you enter your data. **But beware for if your spreadsheet is loosely structured you could encounter difficulties when you import the data into statistical software like SPSS.** If you follow these general guidelines for data entry, the data import will go smoothly.

1. Arrange your data in rectangular format.
2. Create codes for any missing values.
3. Create variable names (8 characters or less).
4. Assign number codes for categorical data.
5. Provide a unique id for each row of your data.

Obsolete

Author summary

Computers are now essential in all branches of science, but most researchers are never taught the equivalent of basic lab skills for research computing. As a result, data can get lost, analyses can take much longer than necessary, and researchers are limited in how effectively they can work with software and data. Computing workflows need to follow the same practices as lab projects and notebooks, with organized data, documented steps, and the project structured for reproducibility, but researchers new to computing often don't know where to start. This paper presents a set of good computing practices that every researcher can adopt, regardless of their current level of computational skill. These practices, which encompass data management, programming, collaborating with colleagues, organizing projects, tracking work, and writing manuscripts, are drawn from a wide variety of published sources from our daily lives and from our work with volunteer organizations that have delivered workshops to over 11,000 people since 2010.

Why should you adopt “good enough programming practices”?

- If you adopt good programming practices, you will…
  - make your job easier today. **(only partially true!)**
  - make your job easier six months down the road when you have to dust off your work and dig in again.
  - help everyone who is currently working with you.
  - encourage future collaborators to work with you.
  - make your research more reproducible.
  - enhance the quality of your publications.
- But, it requires time and energy and the breaking of many old habits.

Recommendations in six areas

1. Data management. Work incrementally, backup your efforts, and document everything. This allows you to reverse course easily if you've gone down a blind alley. It saves you time if you are presented later with "the correct" data. Finally, good data management makes it easier for someone else to replicate your work.
2. Software.
3. Collaboration.
4. Project organization.
5. Keeping track of changes.
6. Manuscripts.

1. Data management

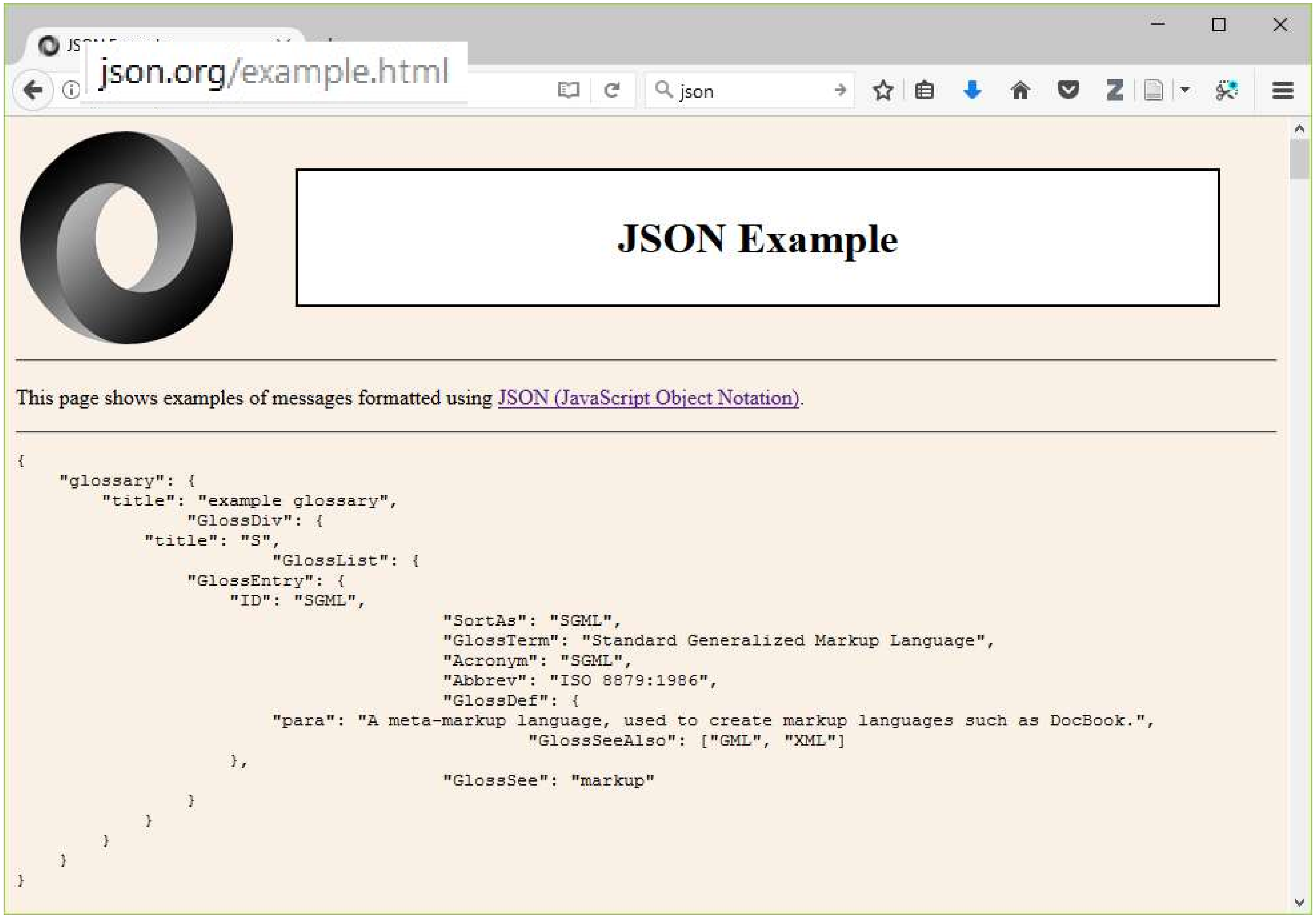
- Save the raw data (1a, page 2).
  - Never make changes directly on top of the data you receive.
  - Note the version number and date of any data you receive remotely.
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).

1. Data management

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
  - Your company/university network may include remote back up, but check first.
  - You can also use cloud services or your own USB stick/drive, but be sure to encrypt anything confidential.
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).

1. Data management

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
  - Use open formats like csv, json, yaml, or xml.
  - Replace cryptic names with self-explaining alternatives.
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).



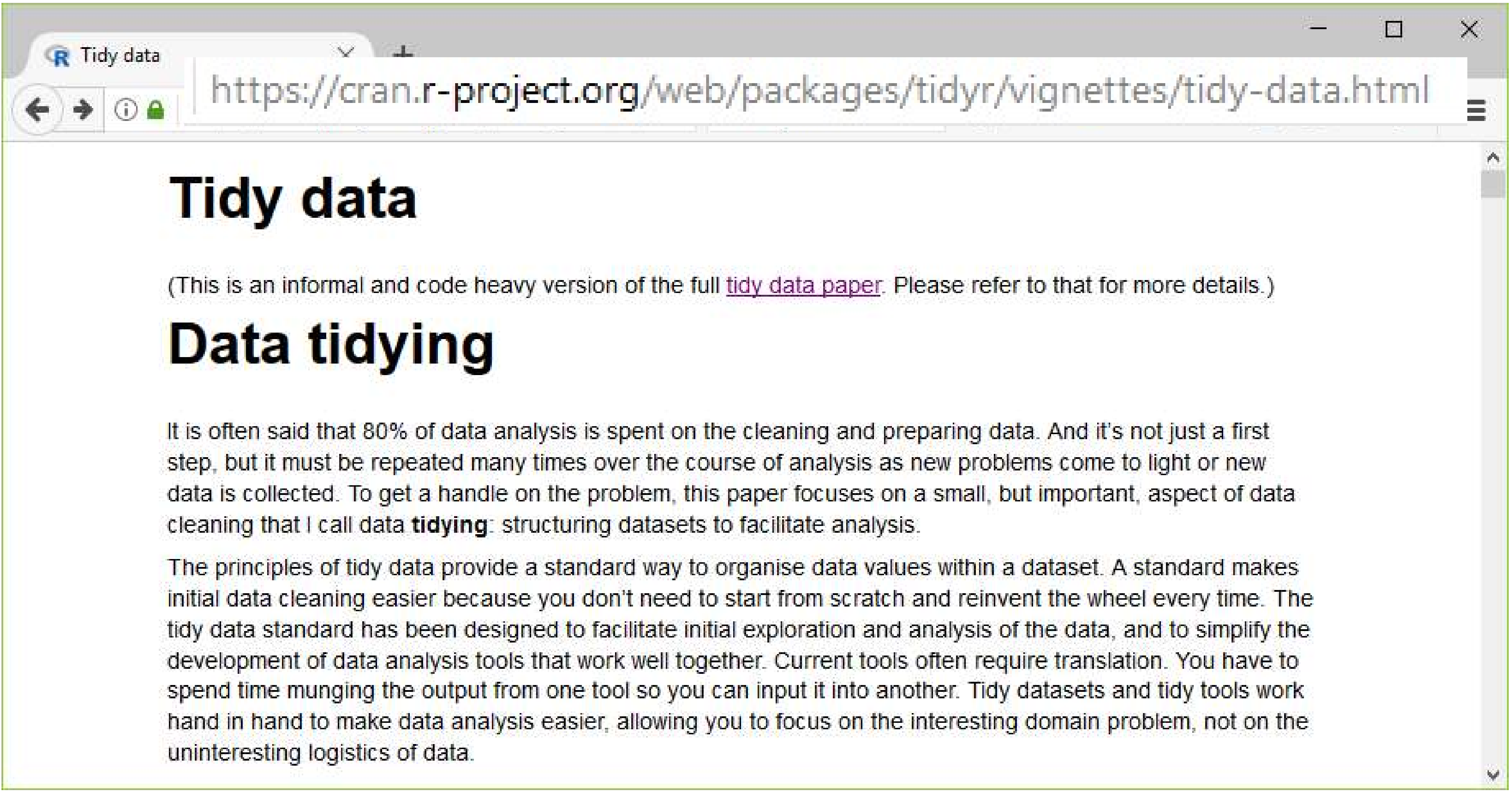
JSON Example

This page shows examples of messages formatted using JSON (JavaScript Object Notation).

```
{
    "glossary": {
        "title": "example glossary",
		"GlossDiv": {
            "title": "S",
			"GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
					"SortAs": "SGML",
					"GlossTerm": "Standard Generalized Markup Language",
					"Acronym": "SGML",
					"Abbrev": "ISO 8879:1986",
					"GlossDef": {
                        "para": "A meta-markup language, used to create markup languages such as DocBook.",
						"GlossSeeAlso": ["GML", "XML"]
                    },
					"GlossSee": "markup"
                }
            }
        }
    }
}
```

1. Data management

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
  - Each column is a variable. (No double dipping).
  - Make each row an observation (use a tall/thin format).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).


Tidy data
https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidy data

(This is an informal and code heavy version of the full tidy data paper. Please refer to that for more details.)

Data tidying

It is often said that 80% of data analysis is spent on the cleaning and preparing data. And it's not just a first step, but it must be repeated many times over the course of analysis as new problems come to light or new data is collected. To get a handle on the problem, this paper focuses on a small, but important, aspect of data cleaning that I call data **tidying**: structuring datasets to facilitate analysis.

The principles of tidy data provide a standard way to organise data values within a dataset. A standard makes initial data cleaning easier because you don't need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together. Current tools often require translation. You have to spend time munging the output from one tool so you can input it into another. Tidy datasets and tidy tools work hand in hand to make data analysis easier, allowing you to focus on the interesting domain problem, not on the uninteresting logistics of data.

1. Data management

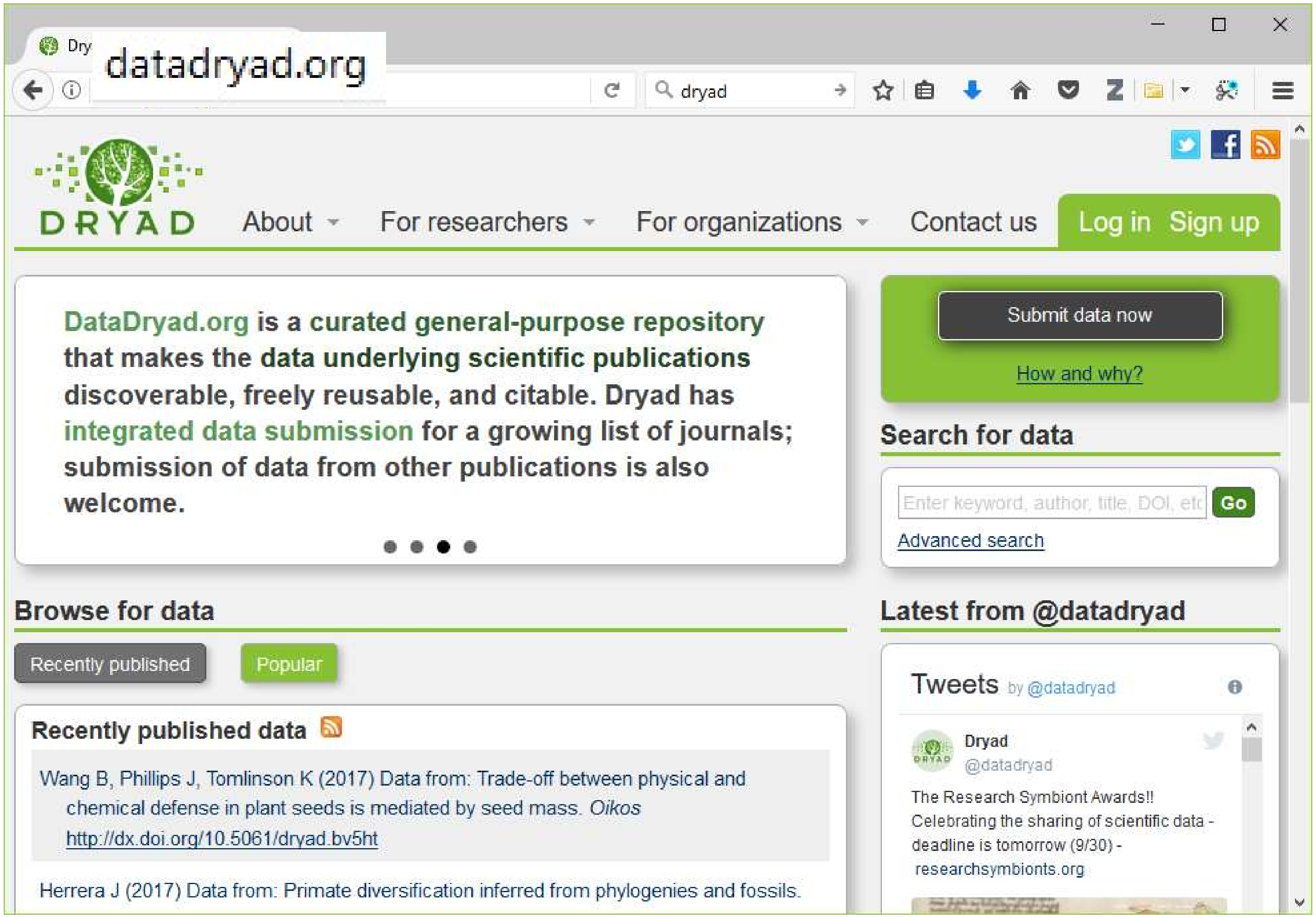
- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
  - Write scripts/use syntax instead of a graphical user interface.
  - If you can’t use scripts/syntax, hand document all your steps.
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).

1. Data management

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
    - This allows you to split your data into pieces (e.g., time-constant and time-varying data in a longitudinal study).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).

1. Data management

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).
  - DOI is Digital Object Identifier, which assigns a permanent and unchanging URL.
  - Include a README file with information that will simplify the task of others using your data.



datadryad.org

DataDryad.org is a curated general-purpose repository that makes the data underlying scientific publications discoverable, freely reusable, and citable. Dryad has integrated data submission for a growing list of journals; submission of data from other publications is also welcome.

Search for data

Browse for data

Recently published data

Wang B, Phillips J, Tomlinson K (2017) Data from: Trade-off between physical and chemical defense in plant seeds is mediated by seed mass. *Oikos* http://dx.doi.org/10.5061/dryad.bv5ht

Herrera J (2017) Data from: Primate diversification inferred from phylogenies and fossils.

Latest from @datadryad

Tweets by @datadryad

Dryad @datadryad

The Research Symbiont Awards!! Celebrating the sharing of scientific data - deadline is tomorrow (9/30) - researchsymbionts.org

1. Data management. What do you think?

- Save the raw data (1a, page 2).
- Ensure that raw data are backed up in more than one location (1b, page 4).
- Create the data you wish to see in the world (1c, page 4).
- Create analysis-friendly data (1d, page 5).
- Record all the steps used to process data (1e, page 5).
- Anticipate the need to use multiple tables and use a unique identifier for every record (1f, page 5).
- Submit data to a reputable DOI-issuing repository so that others can access and cite it (1g, page 6).

Recommendations in six areas

1. Data management.
2. Software. The program that you write to analyze your data is software, and software can be improved through the use of well-tested software development methods.
3. Collaboration.
4. Project organization.
5. Keeping track of changes.
6. Manuscripts.

2. Software (a-e)

- Place a brief explanatory comment at the start of every program (2a, page 6).
  - Give an example of how it is used, and explain what parameters are needed.
  - Write the explanatory comment BEFORE you write your program, and revise it as needed as you revise your program.
- Decompose programs into functions (2b, page 7).
- Be ruthless about eliminating duplication (2c, page 7).
- Always search for well-maintained libraries that do what you need (2d, page 7).
- Test libraries before relying on them (2e, page 7).

2. Software (a-e)

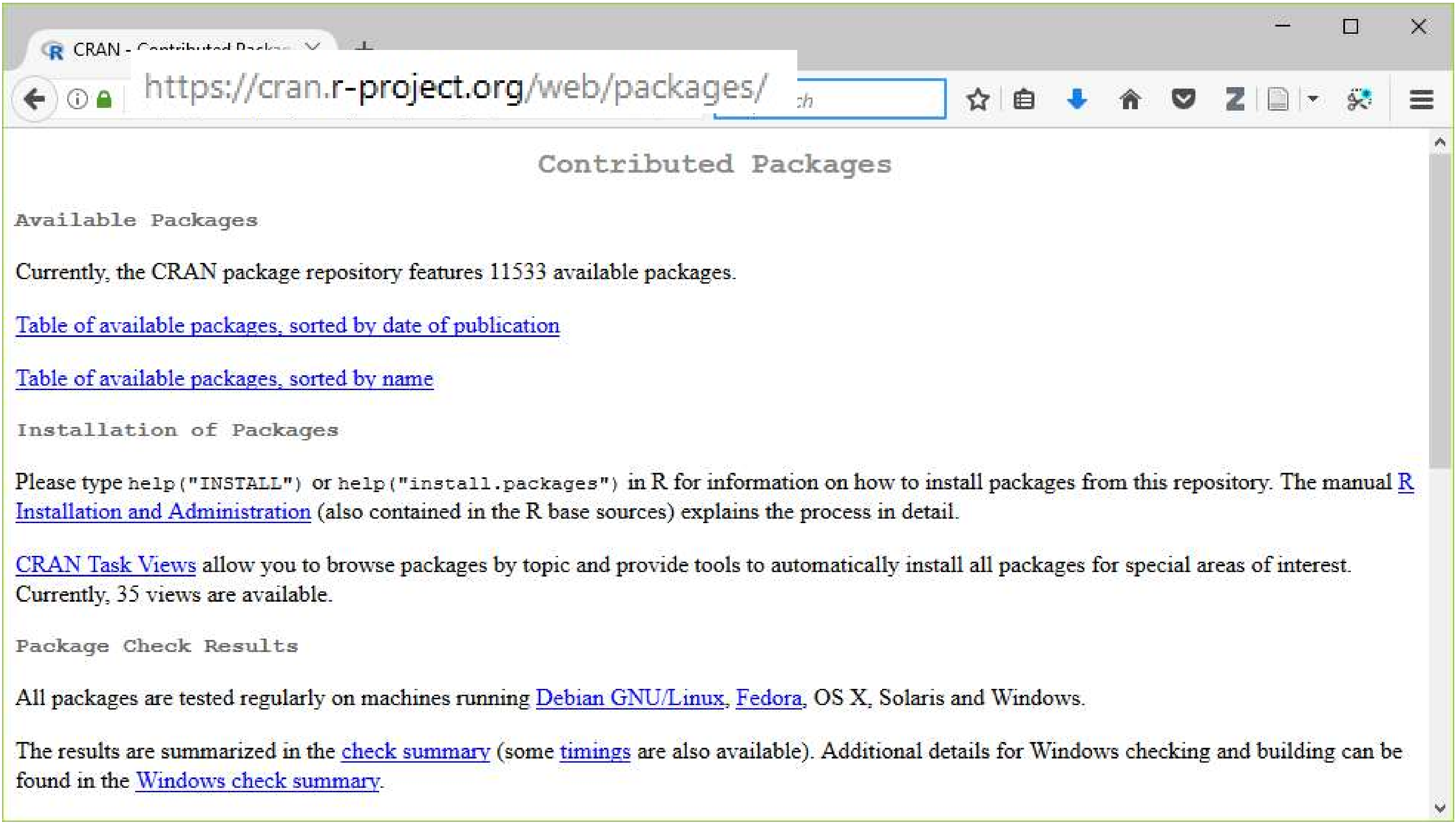
- Place a brief explanatory comment at the start of every program (2a, page 6).
- Decompose programs into functions (2b, page 7).
  - This really means using macros for programs like SPSS and SAS.
  - Keep it short: one page and no more than 6 input parameters.
  - Do not reference information outside the function.
- Be ruthless about eliminating duplication (2c, page 7).
- Always search for well-maintained libraries that do what you need (2d, page 7).
- Test libraries before relying on them (2e, page 7).

2. Software (a-e)

- Place a brief explanatory comment at the start of every program (2a, page 6).
- Decompose programs into functions (2b, page 7).
- Be ruthless about eliminating duplication (2c, page 7).
    - DRY code (don’t repeat yourself).
    - Cut, paste, and slightly modify is your enemy.
- Always search for well-maintained libraries that do what you need (2d, page 7).
- Test libraries before relying on them (2e, page 7).

2. Software (a-e)

- Place a brief explanatory comment at the start of every program (2a, page 6).
- Decompose programs into functions (2b, page 7).
- Be ruthless about eliminating duplication (2c, page 7).
- Always search for well-maintained libraries that do what you need (2d, page 7).
    - Don’t be a Frank Sinatra programmer (“I did it my way.”)
- Test libraries before relying on them (2e, page 7).


CRAN - Contributed Packa…
https://cran.r-project.org/web/packages/

Contributed Packages

Available Packages

Currently, the CRAN package repository features 11533 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 35 views are available.

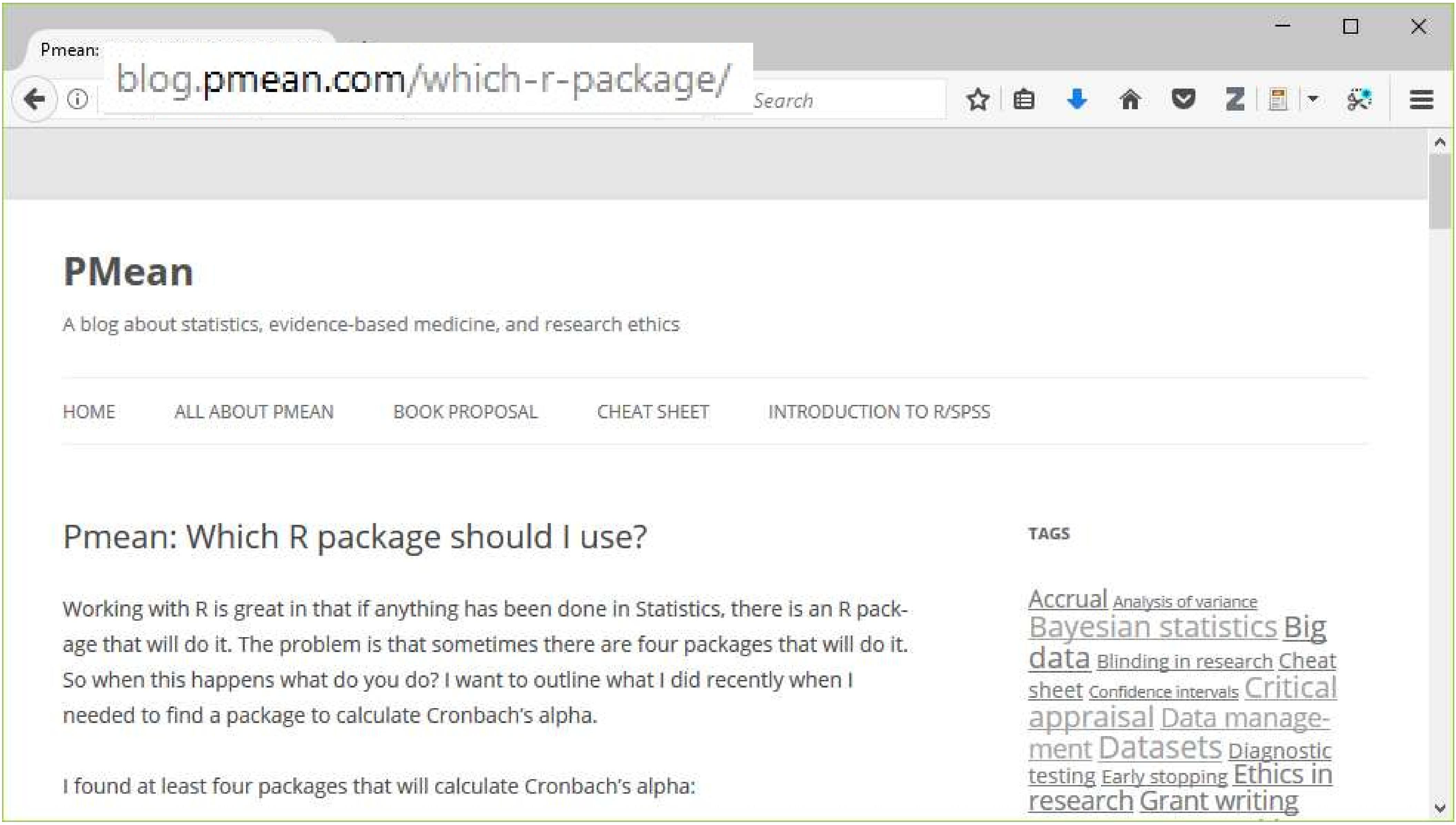
Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, OS X, Solaris and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

2. Software (a-e)

- Place a brief explanatory comment at the start of every program (2a, page 6).
- Decompose programs into functions (2b, page 7).
- Be ruthless about eliminating duplication (2c, page 7).
- Always search for well-maintained libraries that do what you need (2d, page 7).
- Test libraries before relying on them (2e, page 7).
  - It might be more practical to say “evaluate libraries” rather than “test libraries.”


blog.pmean.com/which-r-package/

PMean

A blog about statistics, evidence-based medicine, and research ethics

HOME ALL ABOUT PMEAN BOOK PROPOSAL CHEAT SHEET INTRODUCTION TO R/SPSS

Pmean: Which R package should I use?

Working with R is great in that if anything has been done in Statistics, there is an R package that will do it. The problem is that sometimes there are four packages that will do it. So when this happens what do you do? I want to outline what I did recently when I needed to find a package to calculate Cronbach's alpha.

I found at least four packages that will calculate Cronbach's alpha:

TAGS

Accrual Analysis of variance Bayesian statistics Big data Blinding in research Cheat sheet Confidence intervals Critical appraisal Data management Datasets Diagnostic testing Early stopping Ethics in research Grant writing

2. Software (a-e). What do you think?

- Place a brief explanatory comment at the start of every program (2a, page 6).
- Decompose programs into functions (2b, page 7).
- Be ruthless about eliminating duplication (2c, page 7).
- Always search for well-maintained libraries that do what you need (2d, page 7).
- Test libraries before relying on them (2e, page 7).

2. Software (f-j)

- Give functions and variables meaningful names (2f, page 7).
    - Functions are verbs and variables are nouns.
    - With the exception of loop counters, avoid one character variable names.
    - Be consistent with your names (CamelCase, dot.delimiters, or underscore_delimiters) and with case (lower_case, Title_Case, or UPPER_CASE).
- Make dependencies explicit (2g, page 7).
- Do not comment and uncomment sections of code to control a program's behavior (2h, page 8).
- Provide a simple example or test data set (2i, page 8).
- Submit your code to a reputable DOI-issuing repository (2j, page 8).

2. Software (f-j)

- Give functions and variables meaningful names (2f, page 7).
- Make dependencies explicit (2g, page 7).
  - Dependencies include what version of the software you need (e.g., 3.1.0 or later) and what additional software or libraries you need. Document this as part of README, or in REQUIREMENTS.TXT.
- Do not comment and uncomment sections of code to control a program's behavior (2h, page 8).
- Provide a simple example or test data set (2i, page 8).
- Submit your code to a reputable DOI-issuing repository (2j, page 8).

2. Software (f-j)

- Give functions and variables meaningful names (2f, page 7).
- Make dependencies explicit (2g, page 7).
- Do not comment and uncomment sections of code to control a program's behavior (2h, page 8).
  - Use if/else statements and verbose/debug flags instead.
  - Don't keep old code around for historical reference, use version control instead.
- Provide a simple example or test data set (2i, page 8).
- Submit your code to a reputable DOI-issuing repository (2j, page 8).

2. Software (f-j)

- Give functions and variables meaningful names (2f, page 7).
- Make dependencies explicit (2g, page 7).
- Do not comment and uncomment sections of code to control a program's behavior (2h, page 8).
- Provide a simple example or test data set (2i, page 8).
  - Run these tests when you switch computers, upgrade to a new software version, or add new pieces to your program.
- Submit your code to a reputable DOI-issuing repository (2j, page 8).

2. Software (f-j)

- Give functions and variables meaningful names (2f, page 7).
- Make dependencies explicit (2g, page 7).
- Do not comment and uncomment sections of code to control a program's behavior (2h, page 8).
- Provide a simple example or test data set (2i, page 8).
- Submit your code to a reputable DOI-issuing repository (2j, page 8).
  - When you share both your data and your code, you help insure reproducibility and you encourage new collaborations.

2. Software (f-j). What do you think?

- Give functions and variables meaningful names (2f, page 7).
- Make dependencies explicit (2g, page 7).
- Do not comment and uncomment sections of code to control a program’s behavior (2h, page 8).
- Provide a simple example or test data set (2i, page 8).
- Submit your code to a reputable DOI-issuing repository (2j, page 8).

Recommendations in six areas

1. Data management.
2. Software.
3. Collaboration: use development approaches that allow a team to work well together. Even if you are a team of one, use these tools to make it easy for others to follow in your footsteps.
4. Project organization.
5. Keeping track of changes.
6. Manuscripts.

3. Collaboration

- Create an overview of your project (3a, page 8).
  - This is useful even if you work alone, because it gives a focus to your work.
  - Store the overview in a README file.
  - Include information about opportunities for collaboration in a CONTRIBUTING file.
- Create a shared to-do list (3b, page 8).
- Decide on communication strategies (3c, page 8).
- Make the license explicit (3d, page 9).
- Make the project citable (3e, page 10).

3. Collaboration

- Create an overview of your project (3a, page 8).
- Create a shared to-do list (3b, page 8).
  - Document this well enough for an outsider to understand.
- Decide on communication strategies (3c, page 8).
- Make the license explicit (3d, page 9).
- Make the project citable (3e, page 10).

3. Collaboration

- Create an overview of your project (3a, page 8).
- Create a shared to-do list (3b, page 8).
- Decide on communication strategies (3c, page 8).
    - Where and how will your team meet to resolve issues.
    - Explain how new collaborators can get involved.
    - Specify where key documents are stored and who can modify them.
- Make the license explicit (3d, page 9).
- Make the project citable (3e, page 10).

3. Collaboration

- Create an overview of your project (3a, page 8).
- Create a shared to-do list (3b, page 8).
- Decide on communication strategies (3c, page 8).
- Make the license explicit (3d, page 9).
  - Use a liberal license that makes it easy for future collaborators to join in the fun.
  - Remember that you are far better off being the pioneer in a new research area that everyone is working in than a new research area that no one is working in.
  - "A candle loses nothing by lighting another candle." James Keller
- Make the project citable (3e, page 10).

3. Collaboration

- Create an overview of your project (3a, page 8).
- Create a shared to-do list (3b, page 8).
- Decide on communication strategies (3c, page 8).
- Make the license explicit (3d, page 9).
- Make the project citable (3e, page 10).
  - Include a suggested citation for your work in a CITATION file.
  - Put a footer at the end of every text document like
    - “This text is licensed under the CC BY 4.0 license (creativecommons.org/licenses/by/4.0/). Feel free to use or modify anything in this file, but please give appropriate credit when you do. An example of appropriate credit would be ‘thanks to Steve Simon (blog.pmean.com) for sharing this material.’”

3. Collaboration. What do you think?

- Create an overview of your project (3a, page 8).
- Create a shared to-do list (3b, page 8).
- Decide on communication strategies (3c, page 8).
- Make the license explicit (3d, page 9).
- Make the project citable (3e, page 10).

Recommendations in six areas

1. Data management.
2. Software.
3. Collaboration.
4. Project organization. Standardize where everything goes (data, doc, results, src, and bin subdirectories), so you don’t lose it and others will know where to look.
5. Keeping track of changes.
6. Manuscripts.

4. Project organization

- Put each project in its own directory, which is named for the project (4a, page 9).
  - If two projects share more than 50% of the code or data, then they can be safely combined into one project.
  - Two projects that share no code or data belong in separate directories.
- Use standard names for subdirectories.
- Name all files to reflect their content or function (4f, page 11).

4. Project organization

- Put each project in its own directory, which is named for the project (4a, page 9).
- Use standard names for subdirectories.
  - README, LICENSE, CITATION, REQUIREMENTS go in root directory.
  - DOC for most other documentation (4b, page 10).
  - DATA for raw data (4c, page 10).
  - RESULTS for intermediate data sets and program output (also 4c, page 10).
  - SRC for source code (4d, page 10).
  - BIN for compiled programs (4e, page 10).
- Name all files to reflect their content or function (4f, page 11).

Box 3. Project layout

```
.
|-- CITATION
|-- README
|-- LICENSE
|-- requirements.txt
|-- data
|    |-- birds_count_table.csv
|-- doc
|    |-- notebook.md
|    |-- manuscript.md
|    |-- changelog.txt
|-- results
|    |-- summarized_results.csv
|-- src
|    |-- sightings_analysis.py
|    |-- runall.py
```

4. Project organization

- Put each project in its own directory, which is named for the project (4a, page 9).
- Use standard names for subdirectories.
- Name all files to reflect their content or function (4f, page 11).
    - Avoid sequence names like logistic_model1, logistic_model2, … as these are not descriptive and subject to frequent renumbering. Better would be simple_logistic_model, cubic_spline_logistic_model, …

4. Project organization. What do you think?

- Put each project in its own directory, which is named for the project (4a, page 9).
- Use standard names for subdirectories.
- Name all files to reflect their content or function (4f, page 11).

Recommendations in six areas

1. Data management.
2. Software.
3. Collaboration.
4. Project organization.
5. Keeping track of changes. Adopt a formal system to track changes. It keeps everyone on your team “on the same page” and allows to back out gracefully from a disastrous change. There are manual approaches (using a shared file, CHANGELOG.txt) and automated software (git or subversion) for tracking changes.
6. Manuscripts.

5. Keeping track of changes (manual or automated)

- Back up (almost) everything created by a human being as soon as it is created. (5a, page 12).
  - This is easily automated by programs like dropbox.
  - If your team is all within the same organization, a shared network folder works well.
- Keep changes small (5b, page 12).
- Share changes frequently (5c, page 12).

5. Keeping track of changes (manual or automated)

- Back up (almost) everything created by a human being as soon as it is created. (5a, page 12).
- Keep changes small (5b, page 12).
    - Any group of edits that you might think about undoing as a batch should be considered a single change.
- Share changes frequently (5c, page 12).

5. Keeping track of changes (manual or automated)

- Back up (almost) everything created by a human being as soon as it is created. (5a, page 12).
- Keep changes small (5b, page 12).
- Share changes frequently (5c, page 12).
  - Do not allow different investigator versions to drift apart.
  - This is even a problem if you run analyses on both a work and home computer.

5. Keeping track of changes (manual or automated). What do you think?

- Back up (almost) everything created by a human being as soon as it is created. (5a, page 12).
- Keep changes small (5b, page 12).
- Share changes frequently (5c, page 12).

5. Keeping track of changes (manually)

- Create, maintain, and use a checklist for saving and sharing changes to the project (5d, page 12).
  - This is stored as TODO.txt.
- Store each project in a folder that is mirrored off the researcher's working machine (5e, page 12).
- Add a file called CHANGELOG.txt to the project's DOC subfolder (5f, page 13).
- Copy the entire project whenever a significant change has been made. (5g, page 13).

5. Keeping track of changes (manually)

- Create, maintain, and use a checklist for saving and sharing changes to the project (5d, page 12).
- Store each project in a folder that is mirrored off the researcher's working machine (5e, page 12).
  - Dropbox can automate this process.
- Add a file called CHANGELOG.txt to the project's DOC subfolder (5f, page 13).
- Copy the entire project whenever a significant change has been made. (5g, page 13).

5. Keeping track of changes (manually)

- Create, maintain, and use a checklist for saving and sharing changes to the project (5d, page 12).
- Store each project in a folder that is mirrored off the researcher's working machine (5e, page 12).
- Add a file called CHANGELOG.txt to the project's DOC subfolder (5f, page 13).
  - Date entries and put most recent changes as the top.
- Copy the entire project whenever a significant change has been made. (5g, page 13).

```
## 2016-04-08
* Switched to cubic interpolation as default.
* Moved question about family's TB history to end of questionnaire.
## 2016-04-06
* Added option for cubic interpolation.
* Removed question about staph exposure (can be inferred from blood
test results).
```

5. Keeping track of changes (manually)

- Create, maintain, and use a checklist for saving and sharing changes to the project (5d, page 12).
- Store each project in a folder that is mirrored off the researcher’s working machine (5e, page 12).
- Add a file called CHANGELOG.txt to the project’s DOC subfolder (5f, page 13).
- Copy the entire project whenever a significant change has been made. (5g, page 13).
  - It is better to waste hardware resources than to waste your time.

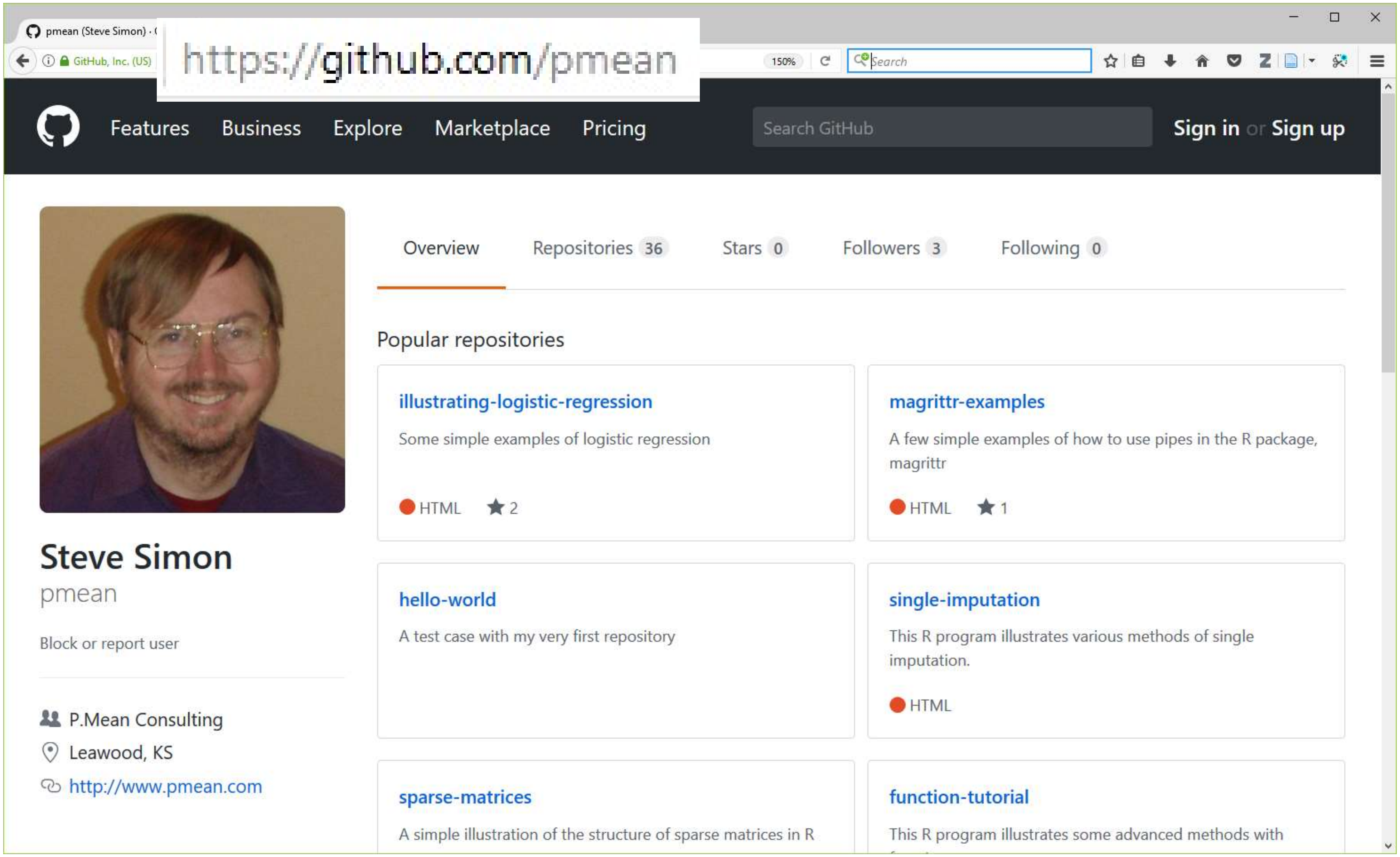
```
|--project_name
|    |--current
|    |    |--...project content as described earlier...
|    |--2016-03-01
|    |    |--...content of 'current' on Mar 1, 2016
|    |--2016-02-19
|    |    |--...content of 'current' on Feb 19, 2016
```

5. Keeping track of changes (manually). What do you think?

- Create, maintain, and use a checklist for saving and sharing changes to the project (5d, page 12).
- Store each project in a folder that is mirrored off the researcher's working machine (5e, page 12).
- Add a file called CHANGELOG.txt to the project's DOC subfolder (5f, page 13).
- Copy the entire project whenever a significant change has been made. (5g, page 13).

5. Keeping track of changes (automated)

- Use a version control system to manage changes to a project (5h, page 14).
  - Version control tracks the actual changes rather than what you think the changes were.
  - Version control handles dating automatically.
  - Version control can rollback changes that you end up regretting.
- But…
  - Do not put large binary files in your version control system.
  - Avoid putting confidential or proprietary information on a public version control system.
  - Don’t store passwords for database access on a public version control system.



https://github.com/pmean

Steve Simon
pmean

Block or report user

P.Mean Consulting
Leawood, KS
http://www.pmean.com

Popular repositories

illustrating-logistic-regression
Some simple examples of logistic regression
● HTML ★ 2

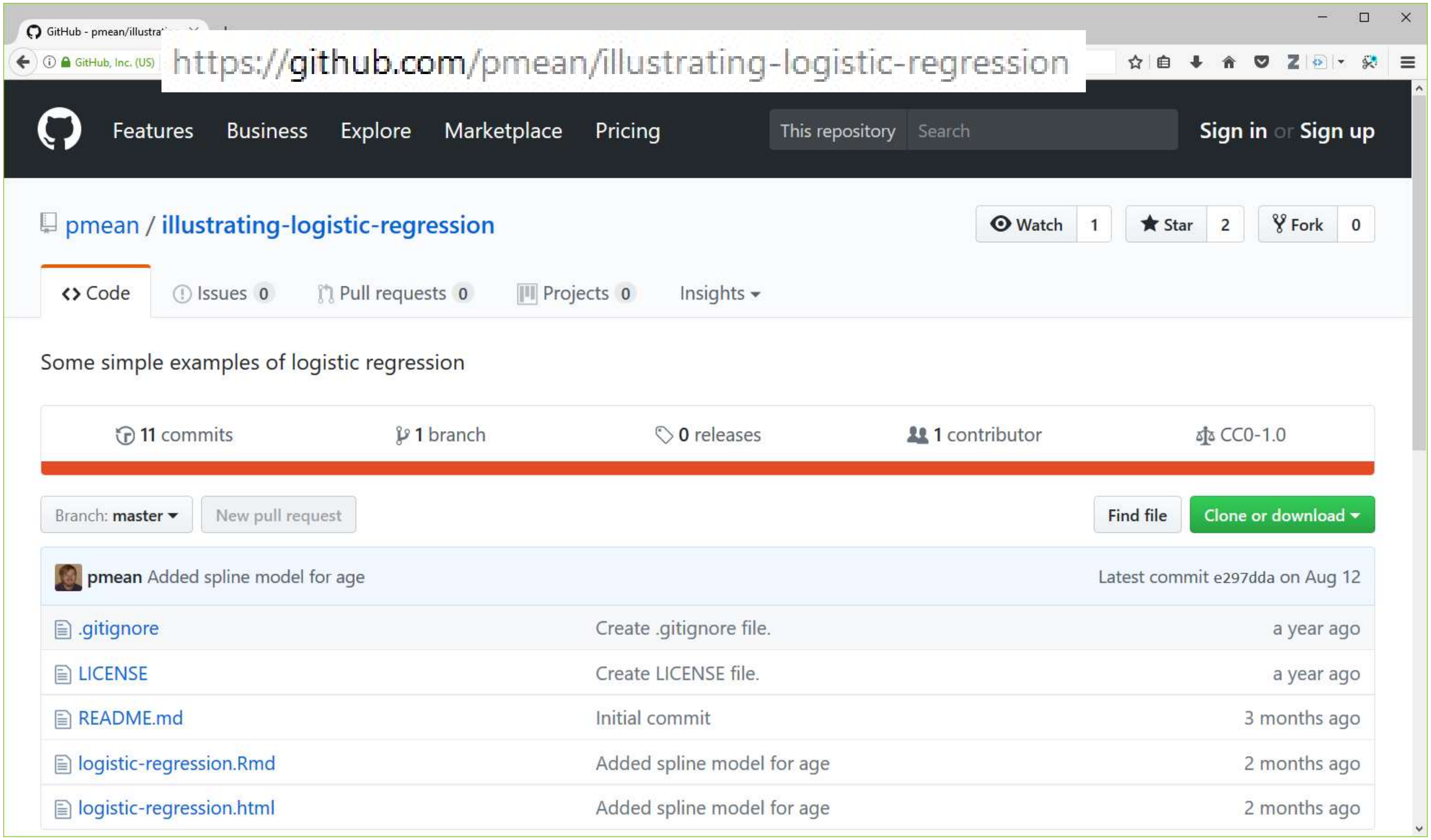
magrittr-examples
A few simple examples of how to use pipes in the R package, magrittr
● HTML ★ 1

hello-world
A test case with my very first repository

single-imputation
This R program illustrates various methods of single imputation.
● HTML

sparse-matrices
A simple illustration of the structure of sparse matrices in R

function-tutorial
This R program illustrates some advanced methods with


https://github.com/pmean/illustrating-logistic-regression

pmean / illustrating-logistic-regression

Some simple examples of logistic regression

11 commits | 1 branch | 0 releases | 1 contributor | CC0-1.0

pmean Added spline model for age — Latest commit e297dda on Aug 12

.gitignore	Create .gitignore file.	a year ago
LICENSE	Create LICENSE file.	a year ago
README.md	Initial commit	3 months ago
logistic-regression.Rmd	Added spline model for age	2 months ago
logistic-regression.html	Added spline model for age	2 months ago

Thank you in advance for consideration of my application. Please find appended to this letter a current curriculum vitae detailing my experience, education, and skill set. For a specific example of my approach to statistical modeling, I encourage you to visit the public GitHub repository hosting the programs for my dissertation: github.com/[redacted]. In addition, if you have any questions concerning any of the material in this letter, please feel free to reach out to me at [redacted]@gmail.com. I look forward to the opportunity to contribute to academic excellence at the UMKC School of Medicine.

Sincerely,

[redacted]

Subject: Re: Update of "How Big Is Your Graph" cheat sheet
From: Garrett Grolemund [redacted]
Date: 8/28/2017 1:05 PM
To: "Stephen Simon, P.Mean Consulting" [redacted]
CC: cheatsheets [redacted], Iserman [redacted]

Thanks Stephen. I've updated the cheatsheet at https://www.rstudio.com/resources/cheatsheets/.

I've also pointed the links to a copy of the cheatsheet hosted here: https://github.com/rstudio/cheatsheets/blob/master/source/pdfs/how-big-is-your-graph.pdf. Now (if you are comfortable with github) you can update the sheet whenever you like by forking the github.com/rstudio/cheatsheets repository, updating the sheet, and submitting the change as a pull request.

You're also welcome to just email me updates if you prefer.

Cheers,
Garrett

On Wed, Aug 23, 2017 at 7:04 PM, Stephen Simon, P.Mean Consulting
[redacted] wrote:

> Someone pointed out a minor error on my "How Big Is Your Graph" cheat sheet. I was talking about the "mgp" argument in the par function when I should have talked about the "mfg" argument.
>
> I fixed this (along with a very minor typo) and would like you to post the revision, please. I am attaching the file, or you can find it on the web at
>
> http://www.pmean.com/0000images/hiw-big-is-your-graph.pdf
>
> --
> Steve Simon, [redacted]
> I'm blogging now! blog.pmean.com

Summary: Keeping track of changes (automated). What do you think?

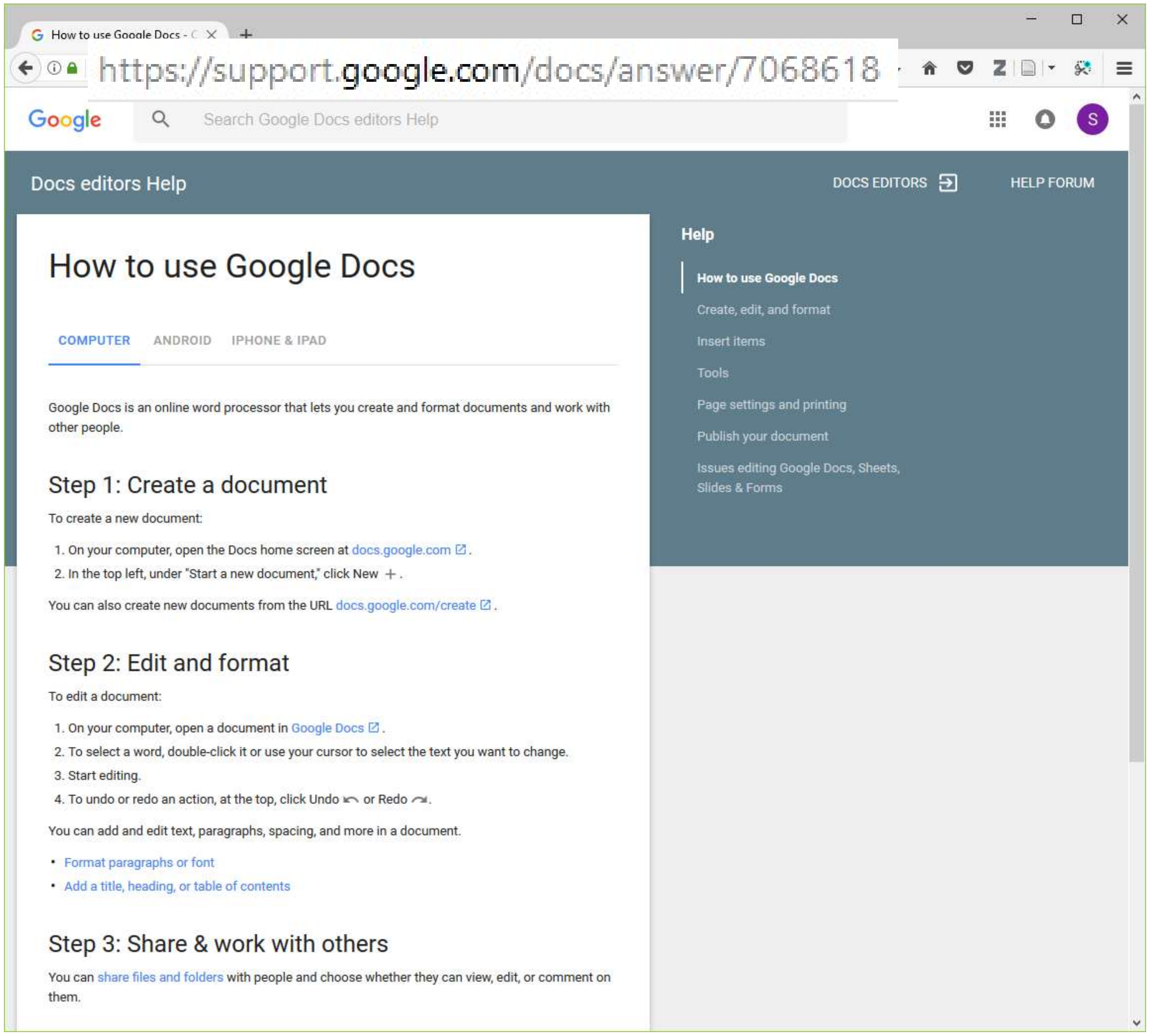
- Use a version control system to manage changes to a project (5h, page 14).

Recommendations in six areas

1. Data management.
2. Software.
3. Collaboration.
4. Project organization.
5. Keeping track of changes.
6. Manuscripts. Prepare your manuscripts using a system that lets everyone collaborate easily. Microsoft Word does not have robust collaboration tools.

6. Manuscripts

- Either
  - Write manuscripts using online tools with rich text formatting, change tracking, and reference management.
    - Google Docs
    - ~~Microsoft Word~~
- Or
  - Write the manuscript in a plain text format that allows version control.
    - LaTeX.
    - Markdown.



How to use Google Docs - C

https://support.google.com/docs/answer/7068618

Search Google Docs editors Help

Docs editors Help

DOCS EDITORS

HELP FORUM

How to use Google Docs

COMPUTER ANDROID IPHONE & IPAD

Google Docs is an online word processor that lets you create and format documents and work with other people.

Step 1: Create a document

To create a new document:

1. On your computer, open the Docs home screen at docs.google.com.
2. In the top left, under "Start a new document," click New +.

You can also create new documents from the URL docs.google.com/create.

Step 2: Edit and format

To edit a document:

1. On your computer, open a document in Google Docs.
2. To select a word, double-click it or use your cursor to select the text you want to change.
3. Start editing.
4. To undo or redo an action, at the top, click Undo or Redo.

You can add and edit text, paragraphs, spacing, and more in a document.

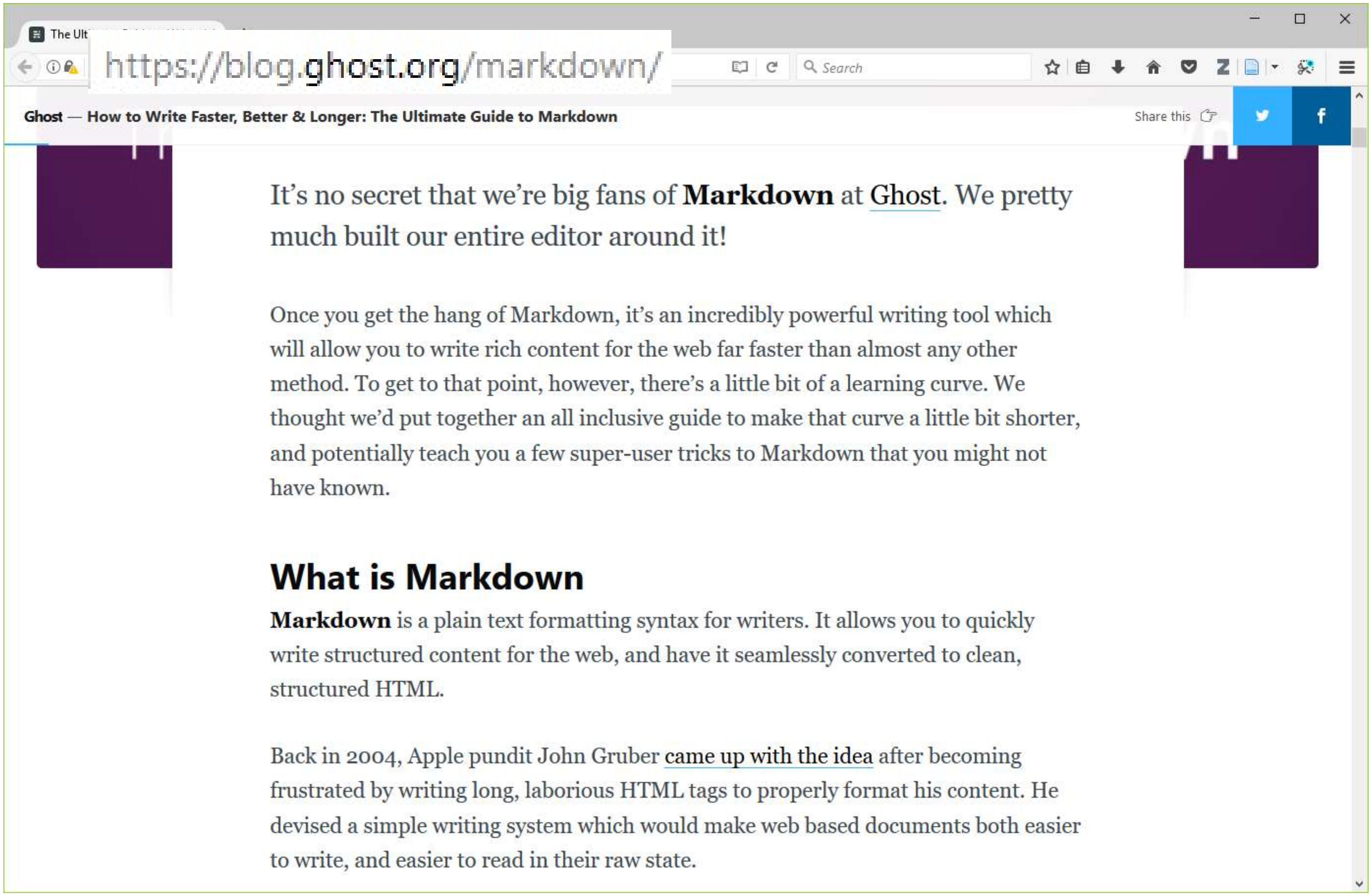
- Format paragraphs or font
- Add a title, heading, or table of contents

Step 3: Share & work with others

You can share files and folders with people and choose whether they can view, edit, or comment on them.

Help

- How to use Google Docs
- Create, edit, and format
- Insert items
- Tools
- Page settings and printing
- Publish your document
- Issues editing Google Docs, Sheets, Slides & Forms



https://blog.ghost.org/markdown/

Ghost — How to Write Faster, Better & Longer: The Ultimate Guide to Markdown

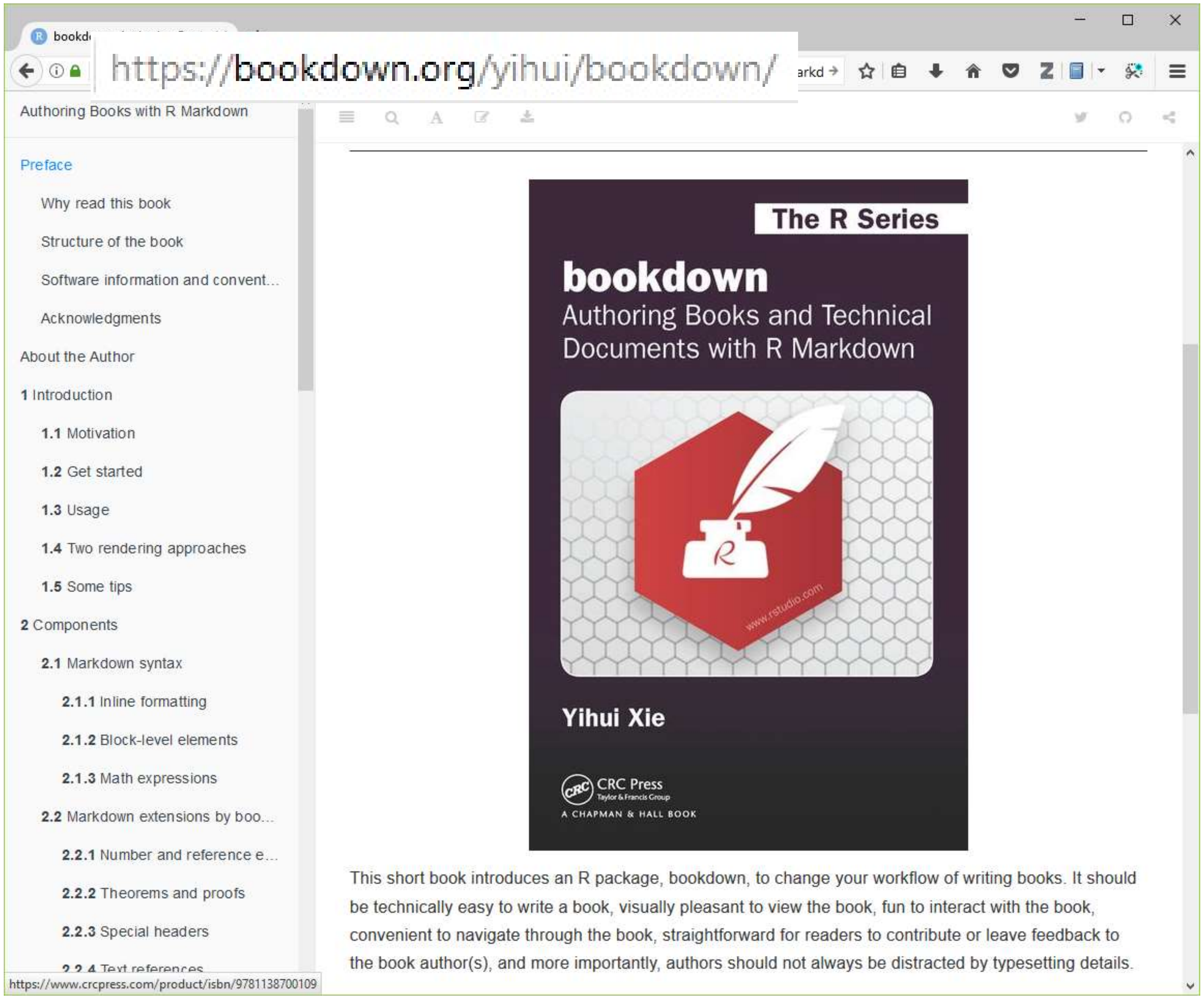
It's no secret that we're big fans of **Markdown** at Ghost. We pretty much built our entire editor around it!

Once you get the hang of Markdown, it's an incredibly powerful writing tool which will allow you to write rich content for the web far faster than almost any other method. To get to that point, however, there's a little bit of a learning curve. We thought we'd put together an all inclusive guide to make that curve a little bit shorter, and potentially teach you a few super-user tricks to Markdown that you might not have known.

What is Markdown

Markdown is a plain text formatting syntax for writers. It allows you to quickly write structured content for the web, and have it seamlessly converted to clean, structured HTML.

Back in 2004, Apple pundit John Gruber came up with the idea after becoming frustrated by writing long, laborious HTML tags to properly format his content. He devised a simple writing system which would make web based documents both easier to write, and easier to read in their raw state.


https://bookdown.org/yihui/bookdown/

Authoring Books with R Markdown

This short book introduces an R package, bookdown, to change your workflow of writing books. It should be technically easy to write a book, visually pleasant to view the book, fun to interact with the book, convenient to navigate through the book, straightforward for readers to contribute or leave feedback to the book author(s), and more importantly, authors should not always be distracted by typesetting details.

https://www.crcpress.com/product/isbn/9781138700109

Paper 324-2012

Using SAS® and LaTeX to Create Documents with Reproducible Results

Tim Arnold and Warren F. Kuhfeld
SAS Institute Inc., Cary, NC

Abstract

Reproducible research is an increasingly important paradigm, and tools that support it are essential. Documentation for many SAS analytical products has long been created from a single-source system that embeds SAS® code in LaTeX files and generates statistical results from those files. This system is now available to SAS users as an open-source package, which is similar in spirit to Sweave (Leisch 2002) and SASweave (Lenth 2007). The system automatically generates the SAS program file, which includes SAS macros that use the ODS document for capturing the output as external files. Listing and Graphic tags display the captured tabular and graphical output. This paper describes how to access and implement the package, and it illustrates typical usage with several examples.

Introduction

At the end of a research project, one of the most difficult tasks remains: documentation. The task is especially difficult with computational research because you must ensure that the displayed program code works as expected and exactly produces the displayed output.

Summary: Manuscripts

- Either
  - Write manuscripts using online tools with rich text formatting, change tracking, and reference management.
    - Google Docs
    - ~~Microsoft Word~~
- Or
  - Write the manuscript in a plain text format that allows version control.
    - LaTeX.
    - Markdown.

But wait, there’s more!

- Once you get really good at this, consider some more advanced methods (but don’t overwhelm yourself).
  - Branches: allows better control over changes in version control.
  - Build tools: “re-compile” everything down to the final paper if your data changes.
  - Unit tests: rigorous approach to insure quality of your functions.
  - Coverage: measures what lines are used/ignored to winnow out unused code.
  - Continuous integration: automate your testing process whenever you make changes.
  - Profiling: identify parts of your code that are bottlenecks.
  - The semantic web: using consensus standards for naming things.
  - Documentation: meaning more than just a few comments here and there.
  - Bibliography manager: EndNotes / Zotero / Mendeley and get an ORCID number.
  - Code reviews and pair programming: have a partner to check your work.

Conclusion

- Here are the recommendations I see as most important:
  - Document and save all your data preparation work.
  - Take the time to write decent code.
  - Standardize how you archive information from your work.
  - Share liberally, if you can.