JOURNERA

Better Data Science Team Collaboration with R

@data_stephanie
www.stephaniekirmer.com
https://github.com/skirmer/team_collab

About Me and Journera

My role includes: modeling, internal tool development, exploratory analysis, customer interaction

Journera:

- We work with travel and hospitality companies (airlines, hotel brands, etc) to help them securely, safely share data with each other for better customer experiences.
- We provide analysis and data science insights that nobody else can do because we have data on multiple sectors of hospitality industry all under one roof.

Find us at www.journera.com!

JOURNERA

Problem:

Data science teams without structured, intentional collaboration leak knowledge and waste resources.

Solution:

Build your team's culture, infrastructure, and tools so that building, retaining, and sharing knowledge is easy and valued.

R packages are a great tool to help you get there.

A Moment on Infrastructure

Part 1. Version Control

How you share ad hoc code, data, and work product

- Lousy: Email
- Mediocre: Mix of shared folders and git
- Good: diligent, consistent use of git

Part 2. Packaged Code

How you organize, share, and develop repeatedly used code

- Lousy: No system at all
- Mediocre: Shared scripts
- Good: Packages!

You need strong version control to make collaboration work!

(Briefly) Introducing Packages

“In **R**, the fundamental unit of shareable code is the **package**. A **package** bundles together code, data, documentation, and tests, and is easy to share with others.”

- Hadley Wickham

Can include:

- Functions
- Markdown templates
- Vignettes and documentation
- Sample data
- Tests to ensure long term code quality
- and more!

Every R package uses a basic architectural framework, so if you open a package folder, you will be able to recognize what’s being done.

Why Use Packages?

Quality and Reproducibility

Improve quality and reproducibility by using peer reviewed, validated tools in package form.

- Rewriting code = opportunity for mistakes.

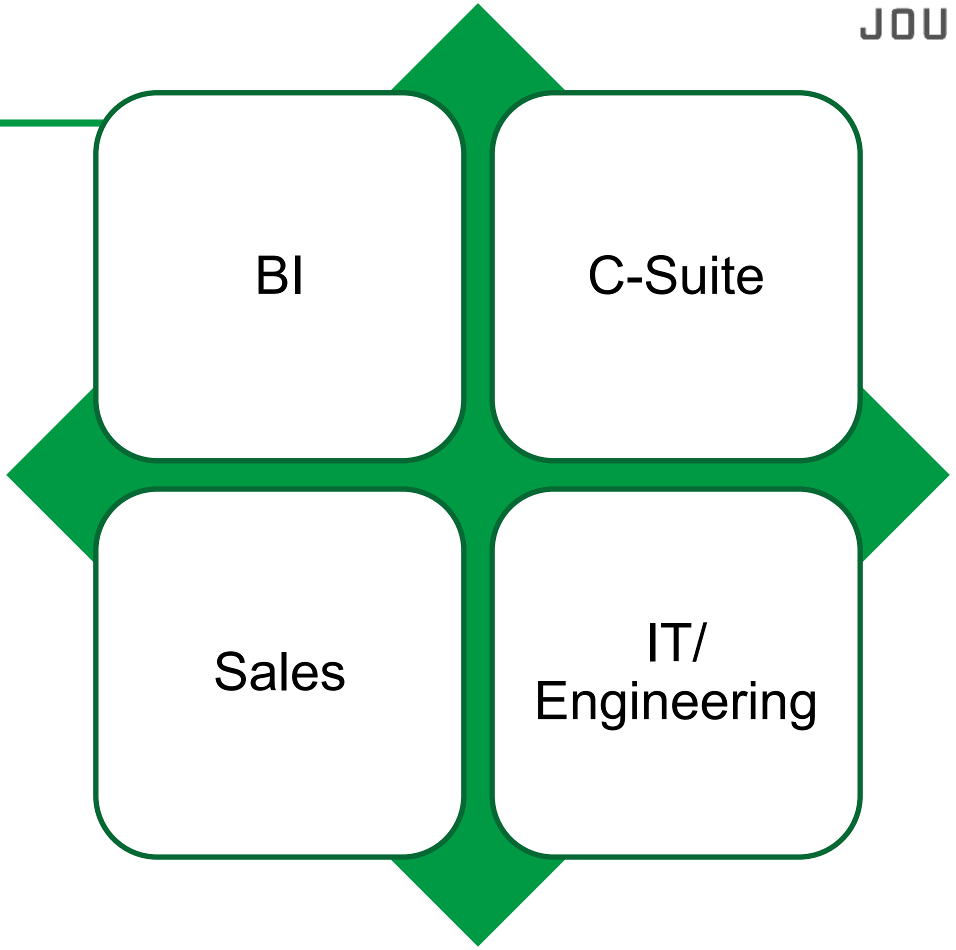
- A shared set of tools creates consistency across projects.
  - Definitions of measures and metrics
  - Data cleaning and management steps

Other Teams

Build package elements for use by other teams.

- Package functionalities can include:
  - Data ETL
  - Basic analysis
  - Visualizations
  - .. and more!

- Teams can also build useful tools for their specific tasks



Empower your colleagues to do routine work, and free up data science to do the complex and innovative projects.

Personnel Changes

Don't wait until personnel changes to think about your information.

With information managed in packages:

New Hire

- Goes from beginner to contributor – fast!
- Has a welcoming onboarding experience.

Staff Departure

- Causes no panic or scramble.
- Results in no lost information or knowledge.

Community tools increase the stability of your team for the long term.

Expand Your Culture of Education

Develop Internal Learning

- Working sessions
- Tech talks

Increase DS Visibility

- Interact with non-DS colleagues
- Add value in tools as well as insights

Build Thought Leadership

- Doorway to open source ecosystem participation

How?

Consider Your Culture

A group of individual data scientists =/= a collaborative team of data scientists.

Develop a paradigm of thinking that is productive and works towards your objectives.

- Common, understood goals
- Trust in colleagues
- Willingness to share information/tools and help each other

Leadership Is Required

Organizational leadership has to commit to changes.

People need to be incentivized to maintain and contribute to packages and collaborative tools through:

- Time
- Compensation
- Recognition

Have a vision for what sort of system you want your team to have, and build infrastructure that can scale and evolve.

Start With Thinking and Planning

Explore your undocumented resources	• Learn your institutional history/knowledge. • Investigate existing best practices.
Catalog user needs	• Who do you want using your tools? What will they need?
Sketch out your packages	• How many? Defined by theme, or user base?
Think for the long haul	• Your needs today will change tomorrow.

Write Functions and Documentation Intelligently

Make modular functions, and follow best practices.

- One operation/task = one function.
- Think about readability and future editability.
- Parameterize any aspects the user may want to specify.

(Remember to plan for growth and change!)

Good jobs for one function:

- Calculate a value
- Generate one plot
- Produce one dataframe

BAD job for one function:

- Calculate seventeen values, produce six dataframes, and produce four plots based on those dataframes

Many useful tutorials exist to help you along (see reference slide)

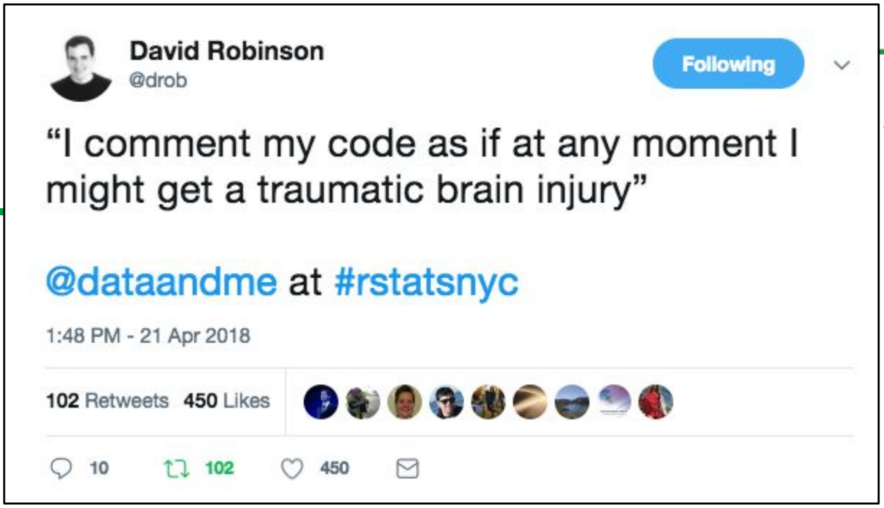
Make Documentation a Habit

At the earliest stages of team infrastructure, documentation is haphazard.

Write for the layperson or for ‘future you’.

R packages have a huge assortment of documentation possibilities!

- Function help docs
- Vignettes
- Package level descriptive documents
- Websites via pkgdown (http://pkgdown.r-lib.org/)


David Robinson
@drob

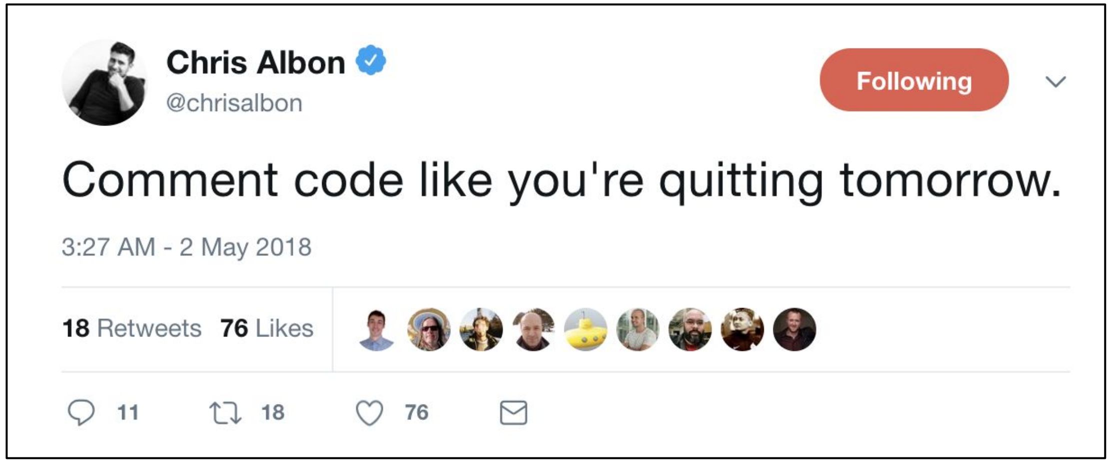
“I comment my code as if at any moment I might get a traumatic brain injury”

@dataandme at #rstatsnyc

1:48 PM - 21 Apr 2018

102 Retweets 450 Likes

10 102 450


Chris Albon
@chrisalbon

Comment code like you're quitting tomorrow.

3:27 AM - 2 May 2018

18 Retweets 76 Likes

11 18 76

Share the Knowledge

When you have tools ready to be used, teach your colleagues.

Let data scientists who authored functions teach their peers.

Proactively teach BI, analytics, executive, and sales staff

- What is this, and why should they care?
- Empower your colleagues to be your collaborators.

Over time, add more things that people need – new functions or whole new packages.

Be friendly and encouraging!

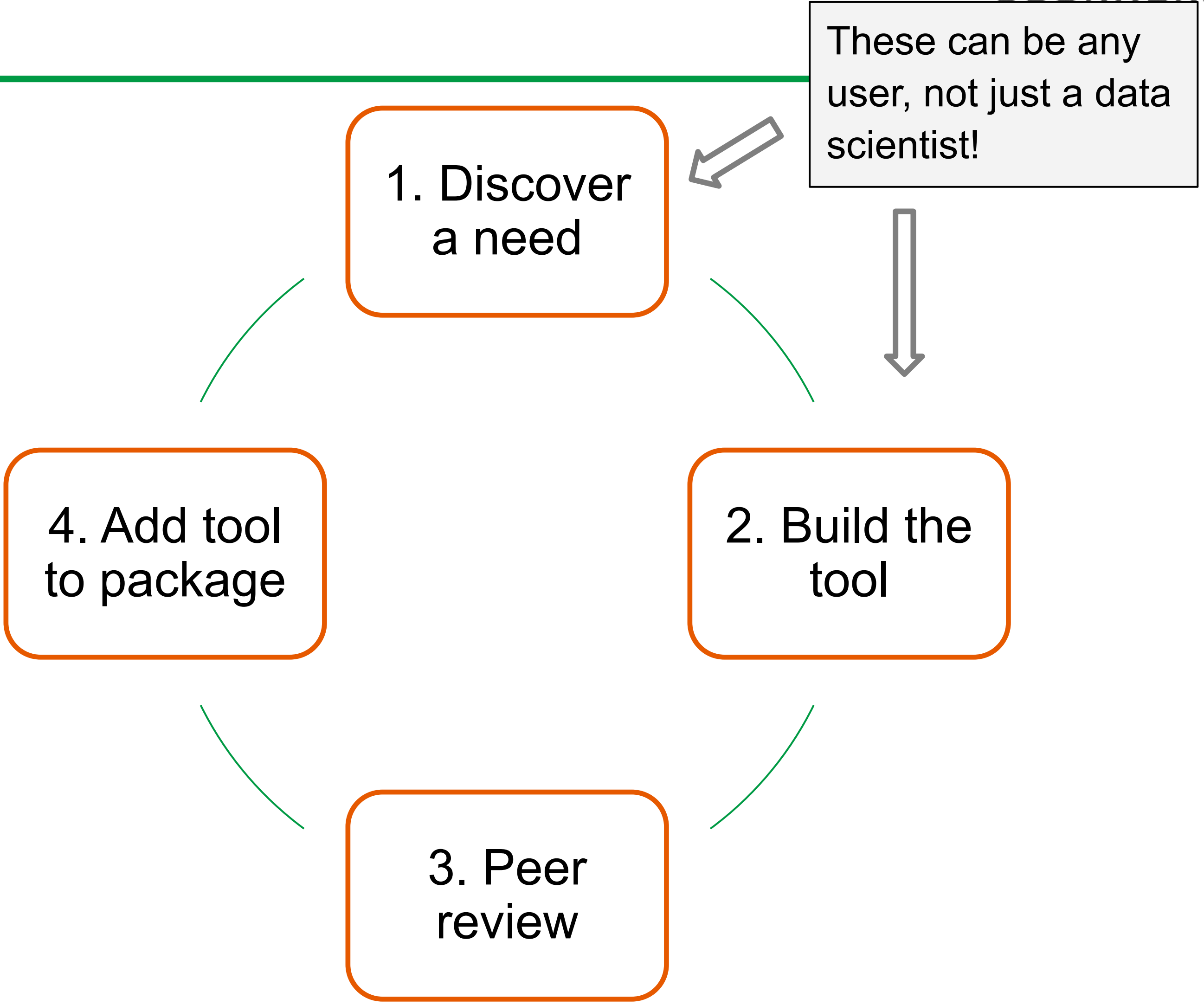
Keep Iterating and Improving

Add new features when the need arises, in a systematic way.

Ideas for new features can come from **any user** – explicitly welcome feedback and contributions.

Don't skimp on the peer review element!

Do occasional housecleaning to ensure things are still working, efficient, and relevant.



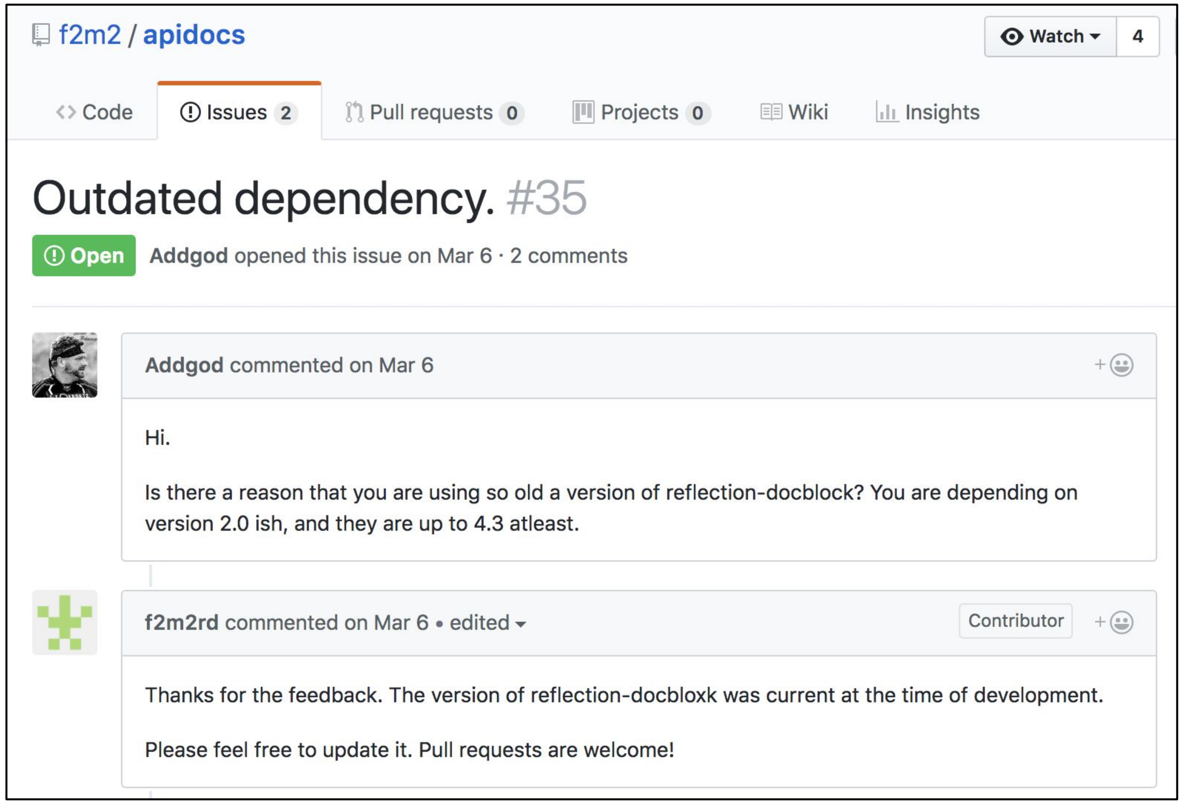
Make Collaboration Everyone's Job

Support improving the package, not just creating the package.

Stagnating documentation/tools < than no tools at all.

Move from single user ownership to team ownership.

- How you talk about the tool matters. Get people invested in it!



Outdated dependency. #35

Open Addgod opened this issue on Mar 6 · 2 comments

Addgod commented on Mar 6

Hi.

Is there a reason that you are using so old a version of reflection-docblock? You are depending on version 2.0 ish, and they are up to 4.3 atleast.

f2m2rd commented on Mar 6 • edited Contributor

Thanks for the feedback. The version of reflection-docbloxk was current at the time of development.

Please feel free to update it. Pull requests are welcome!

Other Technical Notes

Unit testing

- Programmatic checks that automatically run when a package is built
- Prevents unintentional breaking changes
- When to write unit tests, and how many, is a topic of debate.
- devtools::use_testthat()

Continuous Integration

- Contributions are built automatically in test environment to ensure constant compatibility

Internal CRAN

- Set up an internal-only repository where your packages live
- They can be installed with a traditional `install.packages()` call with a few small setup steps
- https://cran.r-project.org/doc/manuals/R-admin.html#Setting-up-a-package-repository

Wrapping Up

Having smooth functioning, collaborative data science is possible!

You can do work that is reproducible, high quality, and less effort.

Your entire organization can be more data literate and self supporting, freeing up data scientists for innovation.

You can build a culture of learning and education for your whole organization, starting from data science.

Suggested Roadmap

1. Develop collaborative culture;
2. Get leadership support;
3. Plan and think ahead;
4. Build tools and write code thoughtfully;
5. Don't skip documentation;
6. Educate your organization;
7. Maintain the tools!

Reference Material

How-To:

- Write your own functions: https://nicercode.github.io/guides/functions/
- Write great documentation: http://r-pkgs.had.co.nz/man.html
- Construct an R package:
  - https://github.com/skirmer/r_packages
  - https://github.com/jtleek/rpackages
- Build websites from package documentation: http://pkgdown.r-lib.org/
- Get familiar with git:
  - https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners
  - https://guides.github.com/
  - https://learngitbranching.js.org/
- Write unit tests:
  - https://katherinemwood.github.io/post/testthat/
  - http://r-pkgs.had.co.nz/tests.html

Case Study:

Airbnb: https://peerj.com/preprints/3182/

Tools:

- https://bitbucket.org/
- https://about.gitlab.com/
- https://github.com/

- https://jenkins.io/
- https://travis-ci.org/
- https://circleci.com/

@data_stephanie
www.stephaniekirmer.com
https://github.com/skirmer/team_collab/