THATS
SO
RANDOM

That's so Random

A playground for data analysis and R programming.

R • DATA SCIENCE • SOFTWARE ENGINEERING • WORKFLOW

A two-stage workflow for data science projects

BY EDWIN THOEN NOVEMBER 27, 2017 9 COMMENTS

If you are a data scientist who primarily works with R, chances are you had no formal training in software development. I certainly did not pick up many skills in that direction during my statistics masters. For years my workflow was basically load a dataset and hack away on it. In the best case my R-script came to some kind of conclusion or final data set, but usually it abruptly ended. Complex projects could result in a great number of scripts and data exports. Needless to say, reproducability was typically low and stress could run high when some delivery went wrong.

Picking up software skills

Colleagues pointed me to some customs in software design, such as creating functions and classes (and documenting them), writing unit tests, and packaging-up your work. I started to read about these concepts and was soon working my way through Hadley's R packages to see how it was done in R. I came to a adopt a rigorous system of solely writing functions, and thoroughly documenting and unit testing them. No matter the nature of the project I was working on, I treated is as a package that should be published on CRAN. This slowed me

down considerably, though. I needed increasing amounts of time to answer relatively simple questions. Maybe now I was over engineering, restricting myself in such a way I was losing productivity.

Data science product

In my opinion, the tension between efficiency and rigor in data analysis, is because the data scientist's product usually is not software, but insight. This goal is not always served best by writing code that meets the highest software standards, because this can make an insight more expensive than necessary, timewise. A software engineer's job is much more clear cut. His product is software, and the better his software, the better he does his job. However, when a data scientist is asked if she can deliver an exploratory analysis on a new data set, should she create a full software stack for it right away? I would say not. Of course clean, well documented scripts will lead to greater reproducibility of and confidence in her results. However, writing solely functions and classes and put a load of unit tests on them is overkill in this stage.

A two-stage workflow

I came to a workflow, with which I am quite happy. It has a hard split between the exploratory stage and the product stage of a project.

Exploratory stage

Starting an analysis, I want to get to first results as quickly as possible. R markdown files are great for this, in which you can take tons of notes about the quality of the data, assumptions tested, relations found, and things that are not yet clear. Creating functions in this stage can be already worthwhile. Functions to do transformations, create specific plots or test for specific relationships. I aim for creating functions that are as generic as is feasible from the start. It is my experience that data is often refreshed or adjusted during a project, even when this was not anticipated at the start of the project. Don't hardcode names of dataframes and columns, but rather give them as arguments to functions. Optionally with the current names as their default values. These functions can be quickly lifted to the product stage. Functions

are only written though, if they facilitate the exploration. If not, I am happy to analyse the data with ordinary R code. Typically, the exploratory stage is wrapped-up by a report in which the major insights and next-steps are described. More often than not, the exploratory stage is the only part of a project as it is concluded to take no further action on the topic researched. The quicker we can get to such a conclusion, the less resources are wasted.

Product stage

A product, to me, is everything designed to share insights, model predictions, and data with others. Whenever a product gets built, whether its a Shiny app, an export of model productions, or a daily updated report, the product stage is entered. In more ignorant days I used to gradually slide from exploratory stage into product stage. In the worst cases commenting out the parts of code that were not needed for building the product. This got increasingly messy of course, yielding non reproducable results and projects that grinded to a halt. Nowadays, as soon as a product gets built I completely start over. Here the software rigor kicks in. All code written in this part only serves the final product, no explorations or asides allowed. I use R's package structure, writing functions that work together to create the product, including documentation and unit tests. When functions are used in the exploratory stage, many components of the product are already available at the start of this stage.

Further cycles

When during product stage it becomes evident that further research is needed, switch back to the exploratory code base. I think it is crucial to be very strict about this. No mess in the product repository! The product directory only contains the code needed for the product to work. When new insight requires the product to change, maybe updating a model or add an extra tab to a dashboard, adjust the product. Again only code necessary for the product to function allowed. Legacy code no longer needed should be removed, not be commented out. This workflow prevents a large set of scripts with a great number of models, many exported data sets, several versions of a Shiny app etc. The product repository always contains the latest version and nothing more.

I hope these hard-learned lessons might be to some value for some of you. I am very curious what others designed as their system. Please share it in the comments of this blog, on Twitter, or in you own blog and notify me.

Adrian • 17 hours ago

"A software engineer's job is much more clear cut. His product is software, and the better his software, the better he does his job." ... consider ... s/w engineering is much in the same pickle as you aptly describe in the article; however, their issue is based on time constraint to deliver "just good enough" to meet some business request (this is the usual case, not all cases). Thus, the s/w eng must 1) get something working as quickly as possible, 2) determine what risks exist having done (1), then (3) package it in a way to provide bz acceptance yet not cause issues with other deployed apps or infrastructure. So, it's not clear cut for s/w eng's at all. Their plight can be tapped to gain experience for the non-programming data scientists. The result is rarely the "best software" possible. But it is "good enough" to achieve the bz objective in the relatively shortest amount of time without compromising other deliverables.

Edwin Thoen Mod → Adrian • 16 hours ago

Hi Adrian, thank you so much for sharing this. I see your point and I very much agree. I am sorry I might have oversimplified here. The point I wanted to make is that finding insight does not have to equate applying all the software principles you know. Data scientists skilled in software design should question if applying those skills does serve the goal of gaining insight, if insight is the overarching goal at the project stage. Once the insight can be leveraged into a product, they should get more rigorous in applying coherent and clean software.

What I mean by clear cut I guess, is that the software engineer always has the overall goal of delivering working software. For a data scientist this can be the goal too, but it does not have to be if it hampers getting insight quickly. I am aware that within software design there are many nuances as you so clearly explained. Thanks again!

Adrian → Edwin Thoen • 11 hours ago

I think we're on the same page here. This discussion supports one of my beliefs for s/w development ... it is more art than science. Data science solutions, if you'll allow me to so describe, will likely meander a bit to find the sweet spot of bz value and acceptable repeatability in the production space.

Reply • Share ›

Ger • 7 days ago

Interesting approach, thanks for sharing. I have a software engineering background and I am working with statisticians now that have basic knowledge about best practices in this area. Especially reproducing results in a team can take quite some time if there is no version control, dependency management, etc.

Reply • Share ›

Edwin Thoen Mod ➚ Ger • 7 days ago

Many statisticians that have to produce software are very eager to learn best practises from engineering. They quickly realise their projects getting easier to maintain and consume less time in the long run. Is this your experience also?

Reply • Share ›

Ger ➚ Edwin Thoen • 6 days ago

yes, definitely for most of them!

Reply • Share ›

Jorge Cimentada • 8 days ago

Nice post. I guess it's important to highlight the need for version control here. It makes the process of adding new code between exploratory and production phases much easier.

Reply • Share ›

Edwin Thoen Mod ➚ Jorge Cimentada • 8 days ago

Very good point. Indeed is the proper use of version control crucial, imo especially in the product stage. The product stage is basically equal to a regular software project, so all the best practises from software engineering apply.

Reply • Share ›

KwangChun Lee • 8 days ago

+1

Reply • Share ›

Check Data Quality with padr – That's so Random

8 comments • 5 months ago

Avatar **Edwin Thoen** — Might it be that you have missing values in the Time column, so that min and max return NA instead of a value? In that

A ggplot-based Marimekko/Mosaic plot

2 comments • a month ago

Avatar **Edwin Thoen** — Thanks! will fix it

Previous