Applied Projects in Statistics

Niels Richard Hansen

April 30, 2018

Contents

Disclaimer: **This document is a draft.**

Introduction

The topic of this document is how to do an applied statistics project and how to write a report. There can be many outcomes of an applied statistics project (a data analysis project) and writing a report is just one. It's, however, still the typical end product for student projects, and this document is written for students, who want to learn how to write such reports. In the not-so-distant future other end products – such as dynamic or interactive online reports, movies, presentations or software products – may become more common. Some advice presented pertains to alternative media formats with minor modifications, while other advice is more specific to the (printed) report.

This document will not cover statistical tools for data analysis, but it will cover data analysis as a process, the organization and documentation of the analysis, the presentation of results, and the organization and writing of a larger document like a written report.

There can be several ways to implement the advice of this document in practice. In particular, there may be alternative software solutions to those described. The advice and proposed solutions should all be relatively easy to implement by using R and RStudio for the data analysis and project writing.

Executive summary

The main points of this document are the following:

- Do reproducible data analysis by using a pipeline of documented code and results. This can be implemented using R Markdown documents.
- Use a standard report structure like IMRaD as a starting point for reporting your results.
- Respect your report structure. In particular, no discussions except in the discussion section.
- Write the report using a *need to know* and not a *nice to know* approach.
- Use high quality graphs for presenting results.

The R Markdown documents will constitute a technical documentation of your data analysis, while the report will be focused on communicating the results clearly and without digressions.

Workflow

When you work with data analysis you should:

- Avoid manually changing or overwriting the original data set.
- Avoid an interactive, undocumented and non-reproducible workflow.
- Avoid manually saving or copying figures or results.

Data analysis via manual interactions with data is an outdated and ineffective mode of data analysis that will be called *low level interactive data analysis*.

Instead you should:

- Write and use code to clean, transform and reformat data.
- Use documented code. Keep the code clean and executable at all times.
- Build pipelines that work without manual intervention.
- Automate what can be automated. Even if it initially takes a little longer.

RStudio and knitr support all four points above via R Markdown documents. But it's also possible to implement a good workflow using ordinary R scripts, or by using the Jupyter Notebook, say. For larger projects involving several people, R packages are useful for developing functionality that is needed by all.

You should use code for data manipulation because otherwise it will be difficult, if not impossible, to reproduce how the data you are analyzing came about. It's tempting to store intermediate "cleaned" data sets, but it's very easy to loose track of how they were actually generated, and you quickly find yourself in a mess with five, ten or more versions of a data set.

It's fast to get started with the data analysis by doing it interactively. However, you quickly find that you want to do the same things you just did but in a slightly different way. Then you either have to do a lot of retyping, or you will become a heavy user of the R command history. Neither results in a reproducible analysis. And even if you can survive for quite some time using the workspace as a "current state of your analysis", you will eventually want to wipe the slate clean.

Note that even if you use R Markdown or an R script, you may still have some bad interactive habits. Maybe you rely on executing particular code segments in a particular order, and maybe your script contains a lot of experimental or broken code. If you cannot execute the entire script or R Markdown file, then the analysis is not reproducible. Of course, you can do interactive experiments, but do one at a time, and leave the code in an executable state.

Everything that can be done interactively can be done programmatically. That includes saving output and figures. Again, it's much easier later to change a figure if you have the code.

If you try to implement everything using R scripts, the following questions arise:

- Why should I run the data cleaning all the time? It seems much better just to do it once, store the result in a file and load that instead.
- Why should I run a heavy computation every time I run my script? It seems much better just to put it in a separate script that I (manually!) run once and then rely on the result floating around in the workspace.

These are valid points, and there is a need for a tool that automatically keeps track of changes and only run code and update data affected by such changes. The solution in knitr is the *cache* functionality. When activated for a given code chunk in your R Markdown document, it stores the results from the code chunk. The code chunk is then not evaluated anymore unless either the code in the chunk or in chunks that it depends upon change. Using the cache resolves the two questions above without jeopardizing reproducibility.

Analysis pipelines

It's often beneficial to view data analysis from the perspective of a production pipeline. Data analysis includes many of the following steps

- acquiring data
- reading, cleaning and reformatting data
- exploratory data analysis, figures and summary statistics
- initial model building / first analysis
- model diagnostics
- improving models / advanced analysis
- internal model validation
- reporting
- deployment
- external model validation

and perhaps several others. Some steps require several cycles to find a adequate form. Data analysis is not a linear process, but it's still helpful to think about all the steps that the data has to pass through as a linear pipeline. The cycles arise when you improve on the steps in the pipeline.

A keyword to success is *modularity*. It's very useful to break down the pipeline as above into smaller units, with each unit having a clear interface towards the surrounding units. When you write code that does model diagnostics, you don't need to think about how the data was actually read in and reformatted. When you write code that does model validation, you don't want to make it specific to a single model. Rather, you should write code that applies to a broad class of models that support a particular set of interfaces.

An *interface* is your own specification of what each unit requires as input and what it produces as output that can be used by other units. It's not much different from writing an R function (what are the arguments and the return value?), but in data analysis it is typically not worth the effort to turn units into functions. However, the minute you start copy-pasting code you should consider writing a function instead.

It's impossible to get all the details and interfaces in the pipeline right the first time, nor is it possible to implement a completely successful pipeline out of the blue. The development of a useful pipeline requires iterations. You start from the end goal by thinking about what the problem or question that your data analysis should deal with is. Ask yourself: what kind of output should I aim for, if I need to solve this problem or answer that question?

The problem could be to build a predictive model (a prognostic model of residual survival time at the time of a breast cancer diagnosis, say). Then a solution is a predictive model, and a good pipeline will have a step on validating the predictive performance of the model. Thus you want the pipeline to eventually produce validation results.

The question could be a scientific hypothesis that you want to shed light on via the data analysis (is a mutation in the BRCA1 gene causally linked to the development of breast cancer, say?). In that case the solution might be a statistical test, or preferably the identification of a parameter that describes the causal

link and an estimate of the causal effect size including an assessment of uncertainty. Then the pipeline should eventually produce such an effect estimate and uncertainty assessement.

Once you have identified what you want to achieve, you can build the first pipeline. For the first pipeline you shouldn't try to find all the "optimal" steps. As Donald Knuth wrote "premature optimization is the root of all evil", and you cannot optimize the individual steps before you can test the impact of what you do. That doesn't mean that you should write sloppy or incorrect code! But you should go for simple and standard solutions for all steps. Maybe you need to swallow some slightly unrealistic assumptions about the data, but rather that then getting stuck. And there is no reason to waste time on solving a problem that you have imagined, which turns out to be irrelevant in the end.

When you have a pipeline that can actually give you an answer, you can start improving the pipeline. This is where modularity is key. If you have developed the pipeline as clearly separated steps (units or modules), you can swap any single step with an improved and optimized step. And you can check how the swap affects the ultimate results, you are interested in.

Version control

There is no need to hide the fact that the pipeline framework is just interactive data analysis at a higher level of abstraction. Though an R Markdown document gives you reproducibility of the actual computations leading from data to result, it does not give you reproducibility of the pipeline development process itself. It's a giant leap forward compared to low level interactive data analysis, but undocumented decisions in the development process may also be problematic.

In addition, mistakes or unfortunate decisions, that are not found immediately, might be difficult to undo. The standard solution (long known and used for software development) for keeping track of the development process is *version control*. Often implemented today via the program Git and perhaps using the repository GitHub for sharing and collaborating.

RStudio supports using Git. It takes a little time to understand the jargon used with such words as *staging*, *pushing*, *committing*, *pull request* and *branching*. But it's not terribly difficult. Once you get used to using Git, you can roll back changes, and you can compare different versions of your code to identify differences (and thus causes of mistakes). It's also a good way to collaborate with others so that multiple users can work on the same document without accidentally overwriting each other's changes.

Can you write the final report using R Markdown?

R Markdown is excellent for creating a working document, and with a little exercise (using e.g. the Notebook format) the result can be quite sharable. However, it's first of all a commented log of the computations, which can be used as a technical report.

It requires an awful lot of work if the output should be a written report of publication quality. First, you do typically not want to share code in the main document, and you only want to present a selected set of figures and tables. Second, a written report should most likely not follow the pipeline structure of your R Markdown document, but rather a more conventional scientific report structure.

Therefore, you typically write your actual report in a separate document, where you can include tables and figures generated by the R Markdown document. Again, you should strive to never manually have to alter the figures or edit the tables.

Report structure

The IMRaD (Introduction, Methods, Results and Discussion) structure is widely used for scientific reports and scientific papers, in particular. It's a standardized format for the organization of sections of the report

and their content, but there's also some flexibility depending on the scientific field, traditions, and the author's and publisher's preferences. It's designed for empirical sciences and not used for writing mathematical papers, say.

The IMRaD structure (or small variations of it) can work well for reporting the results of an applied statistics project. For large reports, the standardized sections may become chapters instead with appropriate sections and subsections. However, for large reports the separation of methods and results may become artificial and inappropriate, and the report may be in need of additional chapters containing a literature survey or theory, say.

One alternative structure could be dubbed IEMaD (Introduction, Exploratory data analysis, Modelling and Discussion). Some alternative sections that might be appropriate to consider, but are not naturally part of the IMRaD structure, are treated below.

In this section the IMRaD structure is described in some detail. The focus is on how it can be used for reporting applied statistics projects. Don't feel obliged to use the structure, if you don't think it fits your needs, but do reflect on why you want to deviate from it.

An example using (a variation of) the IMRaD structure is the paper Survival prognosis and variable selection: A case study for metastatic castrate resistant prostate cancer patients. For reproducibility the code is available as a Zenodo archive as well as on GitHub.

The IMRaD structure

The backbone of IMRaD consists of the four sections: Introduction, Methods, Results, and Discussion. There is also always a references section at the end of the report, and typically an abstract as well. What goes into the different sections and how they are organized are quite standardized. Each of the sections are covered below.

Abstract

A short summary stating the problem, your proposed solution and the main results and conclusions. It should be brief and non-technical. A good abstract makes the reader want to read on, but don't ever exaggerate.

Some scientific journals require standardized *structured abstracts*. See e.g. this IEEE guide, the guide from Medical Library Association, or this brief NIH note. It's a good idea to become familiar with structured abstracts. Even if you are not writing a paper to a particular journal or use labeled sections, ordering the content of the abstract as in a structured abstract can be beneficial for clear communication.

Introduction

Provides the background and framework. It should give the reader sufficient information on how the report fits into the bigger picture. The introduction often also gives a brief overview of how the report is organized and what the main results achieved are.

The introduction is more detailed than the abstract, but it should still be fairly non-technical. Introduce essential technical terms, but keep the focus on *purpose*, *objective*, *framework* and *results*.

Methods

Sometimes in experimental sciences this section is called "Materials and methods". It's used for reporting in detail on the methods used. It's a full description and documentation of your analysis pipeline and all the components in it.

The ideal is (or used to be) that the methods section could make others reproduce the findings presented in the report. This can rarely be achieved in practice, and real reproducibility of the data analysis requires access to the code, e.g. in the form of an R Markdown document. However, the section should provide the reader with enough detail on the methods, so that he or she can assess if the methods used are appropriate.

You should avoid discussions in this section. It's tempting to argue why one method might be good and another bad, but all discussions should be postponed to the discussion section.

Results

In this section you report the results of the data analysis. The convention is that results are reported in the past tense. The text should be neutral and summarize the facts that you found in the data analysis. You should again avoid discussing the results.

If the results show that one prognostic survival model is better than the others, then state that observation. If the results show that a mutation in the BRCA1 gene has a causal effect on breast cancer development, then state that and report the estimated effect size and uncertainty estimate.

Reporting the results correctly and in a comprehensible way is very important. Some time should be invested in the development of high quality graphs. You don't want to show all the graphs that you produced in the analysis, but you want to carefully develop a few graphs that show the key results.

Discussion

This is where you relate your findings to the literature, discuss why certain methods are preferable over others, why some models are better than others, whether your findings are based on certain untestable assumptions, whether these assumptions are actually plausible, etc.

Finally, you can draw conclusions suitably modified by the discussion of potential weaknesses and knowledge from the literature.

References

Needless to say, all material used should be appropriately cited in the main text, and the references should be collected in this section. Please be careful that references are correct in all details. Whenever you use an R package, you should also provide an appropriate citation, and R has the `citation` function for that purpose.

```
citation() ## Citation for the R language itself
```

```
##
## To cite R in publications use:
##
##   R Core Team (2017). R: A language and environment for
##   statistical computing. R Foundation for Statistical Computing,
##   Vienna, Austria. URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2017},
##     url = {https://www.R-project.org/},
```

```
##   }
##
## We have invested a lot of time and effort in creating R, please
## cite it when using it for data analysis. See also
## 'citation("pkgname")' for citing R packages.
```

The `citation` function provides by default a citation for the base R package, which implements "basic functions which let R function as a language".

Citations for a package are provided if the package developers have included an appropriate citation. The results can be formatted as a BibTeX entry.

```
print(citation("glmnet"), bibtex = TRUE) ## Citations for the glmnet package
```

```
##
## To cite glmnet in publications use:
##
##   Jerome Friedman, Trevor Hastie, Robert Tibshirani (2010).
##   Regularization Paths for Generalized Linear Models via
##   Coordinate Descent. Journal of Statistical Software, 33(1),
##   1-22. URL http://www.jstatsoft.org/v33/i01/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Regularization Paths for Generalized Linear Models via Coordinate Descent},
##     author = {Jerome Friedman and Trevor Hastie and Robert Tibshirani},
##     journal = {Journal of Statistical Software},
##     year = {2010},
##     volume = {33},
##     number = {1},
##     pages = {1--22},
##     url = {http://www.jstatsoft.org/v33/i01/},
##   }
##
## If coxnet is used, please also cite:
##
##   Noah Simon, Jerome Friedman, Trevor Hastie, Rob Tibshirani
##   (2011). Regularization Paths for Cox's Proportional Hazards
##   Model via Coordinate Descent. Journal of Statistical Software,
##   39(5), 1-13. URL http://www.jstatsoft.org/v39/i05/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent},
##     author = {Noah Simon and Jerome Friedman and Trevor Hastie and Rob Tibshirani},
##     journal = {Journal of Statistical Software},
##     year = {2011},
##     volume = {39},
##     number = {5},
##     pages = {1--13},
##     url = {http://www.jstatsoft.org/v39/i05/},
##   }
```

Additional sections

The IMRaD structure may be supplemented by additional sections such as:

- Conclusion
- Appendices (supplementary material)
- Acknowledgements

or other ad hoc sections.

Alternative structures

The IMRaD structure has its clear benefits. Because it's a standard most scientists use and know, it makes the reading easy and fast. As a reader you know where to look for the information you want. It can, however, also be quite restrictive, if the work you have done doesn't fit into the format that well.

Theory

The IMRaD structure doesn't have a theory section. One can try to squeeze theory into the methods section, but that isn't always a good idea. What people look for in the methods section is a concise description of what was done and not a theoretical justification. Thus they don't look for theoretical arguments, derivations or discussions related to the methods. This is one issue that makes it clear that IMRaD is designed for empirical and not theoretical sciences. If theory is an important part of the project and the report, include theory as a separate section or chapter.

Simulation studies

A simulation study can be split between the methods and results sections, so that the design of the simulation study is presented in the methods section, while the results are given in the results section. Alternatively, one section can be devoted to the simulation study only.

Merging methods and results

Instead of reporting all methods first and then all results afterwards, it's sometimes better to have several subsections dealing with methods and results together one method at a time. This can give a more natural flow in the reading, when you don't have to flip back and forth between methods and results. The downside is that the comparison of results across different methods is made more difficult, or that a coherent understanding of the results is made more difficult.

For longer reports it's often a good idea to merge methods and results, and have a section (or an entire chapter even) for each method with corresponding results. The main results can then be summarized and compared in a section or chapter just before the discussion.

Writing recommendations

Writing and editing

When writing it's a good idea to write a lot. You understand a problem, a method or a result much better after having written about it. It clarifies your thinking to write out specific details as well as to write out abstract and general ideas. Nothing is – in principle – too small or too big to be considered. There is, of course, a limit on your time budget, and you need to prioritize, but otherwise there's no reason to constrain

yourself. But that does *not* mean that you should report everything you write! Besides being a writer your other role is being an editor.

To act as your own editor is difficult. As an editor you have to be critical about the text from the point of view of a reader. Which sentences, paragraphs and sections work, which should be rewritten, and which should simply be deleted? As a writer you are creative and should be enthusiastic about your own text – it's your creation, it's alive and it's evolving. It's difficult to kill your own creations, but, as editor, you need to be able to delete.

When you write you should avoid value-laden or opinionated writing. All claims you make should be recognizable as either basic facts or justified by your results or by appriate arguments or references. Thus keep focus on the facts and the figures and what you know and can justify.

Level of details

When you write your report, you should first identify who you are writing to, and what they should be able to get out of your report. You should also write the report using a *need to know* instead of a *nice to know* approach. Most textbooks you have read during your studies were written from a *nice to know* perspective. To formulate a mathematical result, say, in abstract generality is often nice but not really necessary. When writing a report there are no extra points for introducing a nice abstraction or for presenting methods or theoretical results that are not needed.

So what you should write is precisely enough details so that the intended reader can understand exactly what they need to understand and no more. You need to include details about all decisions that you've made, which are not self-evident. Mathematical details are, on the other hand, often irrelevant for the presentation. You should, for instance, not include a mathematical formula, if it's not used explicitly for an explanation, a definition or a derivation. Nor should you include components of a derivation that the reader cannot follow anyway.

You might have written one full page of detailed explanations of a method that you implemented, and upon reading it, you realize that two precise sentences would be enough. What a waste to throw away the entire page! It is tempting to leave the page in the report *just in case* the reader is interested in these details. But that is often the wrong choice. If the short version contains what the reader *needs to know*, then it's enough.

To give a concrete example, suppose you are fitting regression models using the shrinkage estimator known as lasso. Two R packages implementing the computation of lasso estimates are `glmnet` and `lars`. When you started working on the project, it was not clear to you how the implemented algorithms work, and what they compute. Hence, you studied the algorithms in detail, and you wrote about the *coordinate descent algorithm*, as implemented in `glmnet`, and you wrote about the *least angle regression* algorithm, as implemented in `lars`. You also wrote about how the results of these algorithms are organized, and though they clearly return different objects, you realized that both algorithms effectively compute a representation of the same thing – what is known as the *solution path* for lasso. That is, the parameter estimates as a function of the tuning parameter. Though these details are, indeed, nice to know, the reader only needs to know something like:

> The solution path, $\lambda \mapsto \hat{\beta}(\lambda)$, for lasso can be computed in several ways. The R packages `glmnet` and `lars` provide implementations of two different algorithms, and for the data analysis both packages were used.

Some specific advice

- It goes without saying that spelling and grammar mistakes should be avoided. Note that there are differences between American and British English such as in *modeling* (BE) and *modelling* (AE). Try to be consistent. To minimize mistakes try to write complete and correct sentences the first time. Later, when editing the manuscript, you can focus on the content and structure.

- Avoid narrative writing. That is, avoid a (real or artificial) chronology such as *First I did, ..., and then I implemented.*
- Avoid any attempt to build up suspense or to write in a mystical or secretive voice that insinuates (unsupported) depth of the work. The reader is not looking for a thrill but for knowledge and understanding.
- Using an active voice is often advocated as good writing practice, but a passive voice can be appropriate for reporting methods and results. The passive voice emphasizes what was done and not who did it, as in *Lasso was used for ...* instead of *I used lasso for ....* Use of a passive voice can also prevent excessive use of *I*, *we* or *us* – and, in particular, the impersonal *one*, which should generally be avoided. But do use the personal pronouns whenever it helps clarify who did what. In the discussion it's, for instance, better to write *We/I found that ...* than *It was found that ....* It's also better to write actively that *Figure 3 shows ...* than passively that *It's shown on Figure 3 ....* For mathematical-style writing *we* is preferred over *I* as in *We see from formula ...* – even if the document has a single author. The use of *we* in this context means *the author(s) and the reader.*
- The word *data* can be used as a mass noun, which thus doesn't have a plural form and is used with a singular verb (*data shows that ...*). Since *data* is the Latin plural form of *datum*, it can also be used with a plural verb (*data show that ...*). Both usages are acceptable and common, but be consistent.
- Appropriate usage of past and present tense can be tricky. As mentioned above, results are conventionally reported in the past tense. Use past tense when writing about an activity that was done as part of the data collection or data analysis or as a preparation for either. Use present tense when writing about a procedure, an algorithm or a method that persists. Use present tense when writing about your interpretation, understanding, recommendation etc. Use present tense when referring to figures or tables in the document, but use past tense when referring to the results they show.
  - In the abstract: *We propose APipe; a pipeline for data analysis.*
  - In the introduction: *The data analysis pipelines BPipe and CPipe were introduced by B. P. (2015) and C. P. (2014), respectively. CPipe is very fast, as shown by C. P. (2014), but BPipe gives more accurate results as B. P. (2015) demonstrated. BPipe is, however, slow, because it uses an exhaustive search over all combinations of variables. This motivated our development of APipe, which uses lasso for fast variable selection.*
  - In the methods section: *We implemented an analysis pipeline that we called APipe. It consists of the three steps ...*.; or *We implemented a new analysis pipeline. It's called APipe and consists of the three steps ....*
  - In the methods section: *The second step in APipe consists of fitting a regression model. This is done by computing the lasso solution path using* `glmnet` *and choosing the tuning parameter by cross-validation.*
  - In the Results section: *Data XX was analyzed using APipe, BPipe and CPipe. Figure 4 shows that APipe and BPipe gave comparable results, while CPipe gave a different result. Figure 5 shows that the computation time for APipe scaled much better with the number of variables than for BPipe, and that it was generally faster.*
  - In the discussion: *We found APipe and BPipe to give comparable results but APipe to be much faster. Though CPipe was found to be even faster than APipe, CPipe is not recommended as it was demonstrated by B. P. (2014) to give inaccurate results.*
  - In the conclusion: *We recommend APipe as a fast but accurate alternative to BPipe.*

Figures, graphs and tables

All figures, graphs and tables should be readable in the printed version of the report, and not just when zoomed in on a computer screen. This applies to all elements of a figure. If some elements are not readable, they should either be made readable or excluded.

Figures should be equipped with suitable legends and a figure text that explains what is on the figure. A good figure and figure text is readable without reading the main text.

Graphs should generally be used instead of tables for reporting results. If the precise values in a table are very important, then include the table in an appendix. It's not always trivial to construct a good graph, but a good graph can be much easier to read and comprehend than a large table.

When you construct a graph in R and save it, you can control such things as size and resolution. When you include a figure in the report you should *never change the aspect ratio*! If you want a wide figure, construct it as a wide figure in R, don't stretch it later.

Further reading

Adrian Wallwork. English for Research: Usage, Style, and Grammar

Donald E. Knuth, Tracy Larrabee, and Paul M. Roberts. Mathematical Writing

Nicholas J. Higham. Handbook of Writing for the Mathematical Sciences, Second Edition

Jeff Leek. Writing your first academic paper

A. S. C. Ehrenberg. Writing Technical Papers and Reports

Garrett Grolemund and Hadley Wickham. R for Data Science

IMRAD on Wikipedia