The computer code of computational research is vital paradata, the academic paper optional metadata

Richèl J.C. Bilderbeek[1]

[1]Department of Immunology, Genetics and Pathology, Science for Life Laboratory, Uppsala University, Uppsala, Sweden

May 31, 2022

Contents

Abstract

Here we define paradata as the data that describes the generation of data. In genetic epidemiology, the data generated is mostly the results of an analysis (e.g. predicting a person having a disease), as done by computer code. In such context, paradata is usually the scientific paper that describes what the computer code does. However, this has the unrealtistic assumption that there is a perfect match between the paper and the code. In this chapter it is argued that the source code should should be supplied, as this is the true paradata: if the paper and code disagree, it is the code that has generated the results. The chapter concludes by some rules how to better code to serve as paradata, and hence allowing computational research to be truly reproducible.

Keywords: paradata, reproducible research, code, software, FAIR data, computational research, Open Science, best practices, genetic epidemiology

Definitions

Term	Definition	Example
Code	Body of text to be run by a computer	The scripts run by a computational experiment
Data	Individual facts, statistics, or items of information	A SNP that has a significant association
Genotype	The DNA allele at a certain location	AA, AC, CG, GT, ...
Paradata	Data that describes the generation of data	The code to conclude that a SNP has a significant association
Phenotype	How an organism looks like in the broadest sense	The concentration of IL6RA in the blood
Metadata	Data that provides information about other data	The article that describes an experiment
Trait	A phenotype	The concentration of IL6RA in the blood

1 Introduction

The currency that universities generate is knowledge, which in turn is distributed in mostly lectures and academic papers. Academic papers contain knowledge, either as the report of measurements or the outcome of an experiment. These reports are described in English, the current lingua franca of science.

A paper contains and/or describes data, where we define data as 'individual facts, statistics, or items of information, often numeric' [1]. To ensure unhampered knowledge growth, this data should [TODO: Why? Reference!] follow the FAIR principles. The FAIR principles [2] are that data should be [TODO: there are probably better examples] Findable (e.g. found by web), Accessible (e.g. DataDryad and or Zenoda), Interoperable (e.g. comma-separated files) and Reusable (e.g. work on all operating systems)

A paper in itself is data, as it is a collection of text and figures that we itself can do measurements upon. Also here, a paper should be FAIR, [TODO: there are probably better examples] i.e. Findable (e.g. Google Scholar) Accessible (e.g. view it online), Interoperable (e.g. written in English) and Reusable (e.g. it is valid to cite and build upon earlier academic papers). This paper itself will later specify the standpoint that a paper is metadata (i.e. data about data), instead of paradata (i.e. data that describes the collection of data)

This paper focusses on research fields that uses computation to do experiments and uses genetic epidemiology as an example. Genetic epidemiology is a field within biology that, among others, measures the spread (hence 'epidemiology') of heritable (i.e. genetic) traits, as well as the relation between having a certain genetic makeup and a certain trait, where the trait can be any human property, such as weight, height, the amount of metabolites and having a disease yes/no.

2 Example study

Figure 1: Picture of Karesuando's church, the village where the Northern Swedish Population Health Study started. From [3]

As an example, we'll use a pseudorandomly selected paper from [4]. The data used by that paper is from a population study called the Northern Swedish Population Health Study (NSPHS) that started in 2010 [5]. The approximately 1000 participants were initially mostly surveyed about lifestyle [5] and follow-up studies provided the type of data relevant for this paper, which are (1) the genetic data [6], (2) the concentration of certain proteins in the blood [7, 8].

The first type of data, the (final form of the) genetic data, consists out of single nucleotide polymorphisms (SNPs). SNPs consist out of a name, a position and a nucleotide. DNA (organized into chromosomes and present in every (nucleated) cell, see figure 2) consists out of many nucleotides, of which there are four types, called adenosine, cytosine, guanine and thyrosine, all commonly abreviated as A, C, G and T respectively. One SNP example is `rs12133641`, which is a SNP located at position 154,428,283, where some people have a certain nucleotide. In this case, 67 percent of the people within this study have an A, and 33 percent have a G (also from [4], Table S3).

The second type of data are concentrations of certain proteins in the blood.

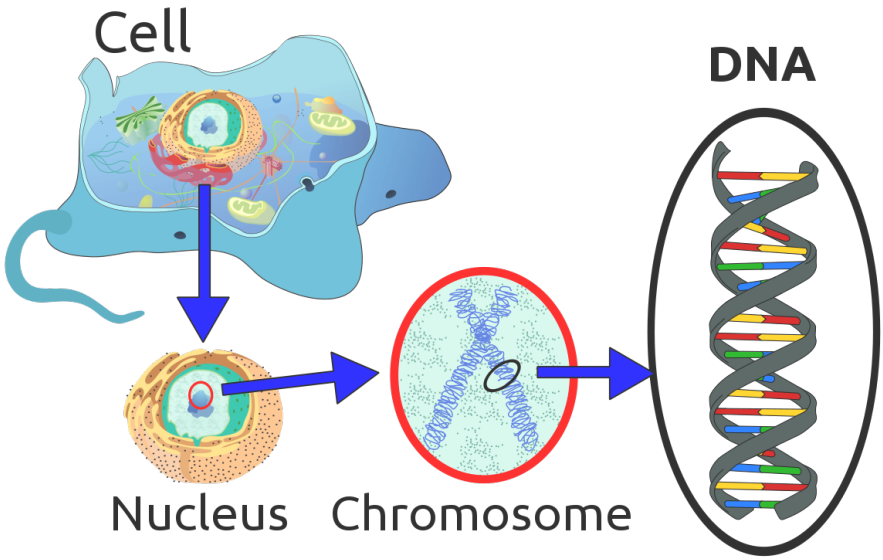

Figure 2: A cell has a nucleus that contains chromosomes. Each of these chromosomes (46 in humans) consist out of DNA. DNA itself conists out of 4 nucleotides, as depicted by the horizontal sticks with the colors red, yellow, green and blue. From [9]

DNA contains the code for building proteins (see Figure 3), as well as the rate at which a protein is created. Some proteins end up in the blood and their presence can be used to assess the health of an individual. IL6RA is a protein [TODO: more info]

The field of genetic epidemiology looks -among others- for correlations between genetic data and biological traits. For example, figure 4 (from [4]) shows that SNP `rs12133641` is highly correlated (p value is 3.0 * 10e-73, 961 individuals) with protein IL6RA. What this results does not yet teach us, is how this correlation works, yet figure 5 shows us the direction of the association: the X axis shows the possible genetic make-ups (aka the genotype) of the individuals, where 0 denotes the individuals with genotype AA (the individual inherited one adenosine from his/here mother and one adenosine from his/her father), 1.0 denotes AG (one A is inherited from one parent, where the G is inherited

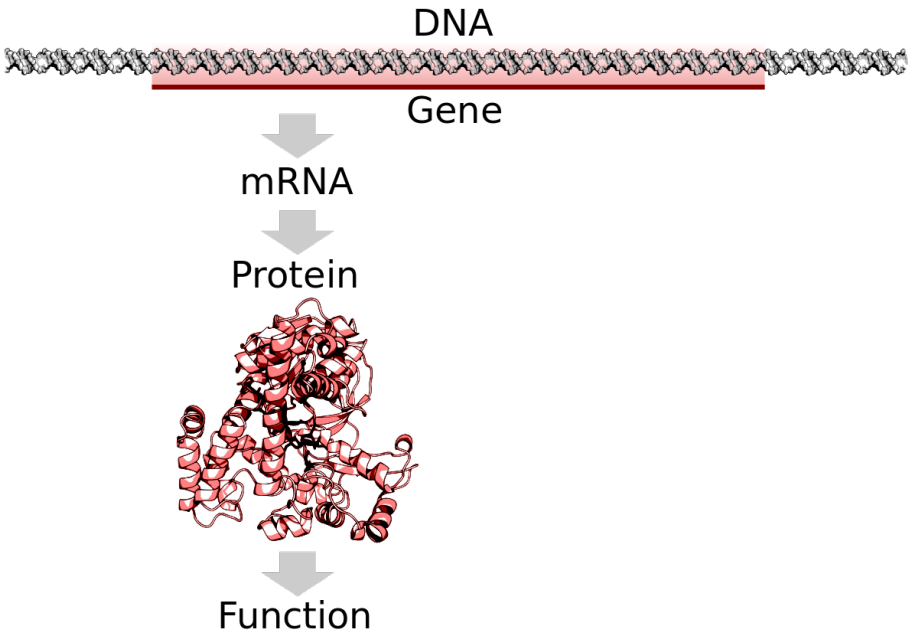

Figure 3: Parts of DNA (so-called 'genes') code for proteins. The DNA, that always stays put in the cell's nucleus, is transcripted to messenger RNA (mRNA). mRNA leaves the nucleus and its code gets translated to a protein sequence. Near the start of a gene are regions that determine the amount of proteins produced (not shown in figure). Adapted from [10]

from the other parent) and 2.0 denotes GG. From Figure 5 we can conclude that, on average, the more guanines are inherited at that SNPs location, the higher concentration of IL6RA can be found in a human's blood.

The amount of variance that can be explained by an association (i.e. the R squared value) is rarely 100 percent, which means that a trait (in this case, IL6RA concentrations) cannot be perfectly explained from the genotype alone. As we can see in figure 5, 43 percent of the variance can be attributed to an individuals' genotype and additional factors, such as the effect of the environment (e.g. geographic location, time of day), lifestyle (e.g. smoking yes/no) or having a disease (e.g. diabetes) are needed to explain the variation between individuals better.

Biomarker	Chr	SNP name	SNP position	*P* (GWAS) N = 961
IL6RA	1	rs12133641	154428283	3.0×10^{-73}

Figure 4: An example result of a genetic epideological research. It shows that the SNP named rs12133641 (located at position 154,428,283 of chromosome 1) is highly correlated (p value is 3.0 * 10e-73, 961 individuals) to the concentration of the protein IL6RA, as measured in blood. The table is a simplified result from [4].

3 Code is more important than the (English) paper

The experiment described above is run by code. There is no fieldwork, nor lab-work involved. It is the researcher that writes the code, after which the code does the work. A genetic epidemiologist that looks for associations between genotype and phenotype, one does not need a lab.

To *obtain* the genetic data, yes, there may be fieldwork and/or lab-work involved. For example, a researcher needs to go into the field to take samples (e.g. blood) of humans or other species. To analyse these samples, a researcher

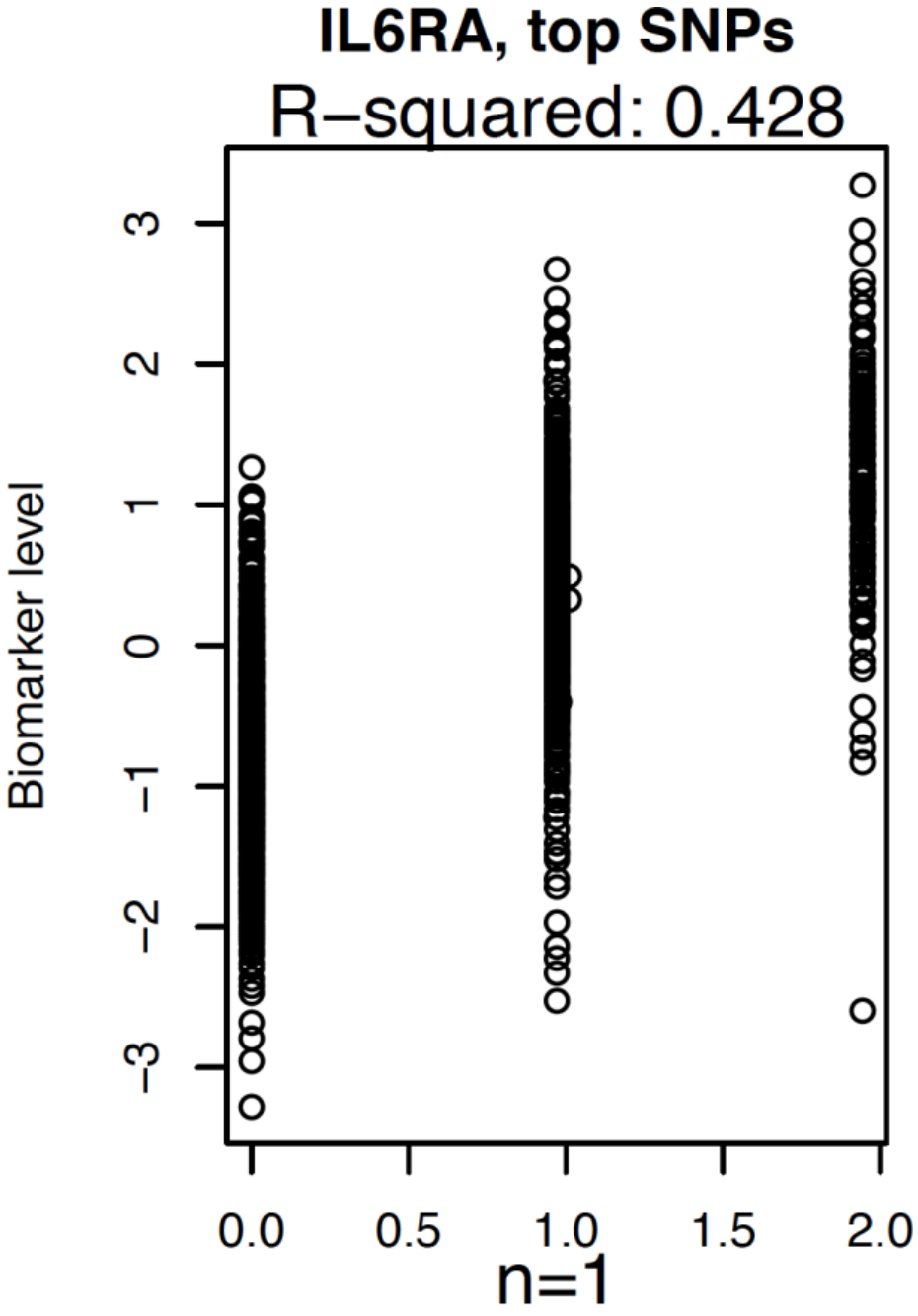

Figure 5: The relation between the genotype for SNP rs12133641 and the protein concentration of IL6RA is relatively strong. The X axis shows the the genotype of the individuals, where 0 denotes AA, 1.0 denotes AG and 2.0 denotes GG. The Y axis shows the concentration of the protein IL6RA as found in the participants' blood. n = 1 denotes the number of SNPs that were determined to be involved.

needs a lab. Depending on the technique being used on how to analyse the samples, there may be a big bioinformatics step to aggregate the measurements into useful genetic data. Although this paper does not focus on the code run to aggragate the raw data into useful genetic data, the same arguments can be made for that code as in the example: that code is the ground truth of how the raw data is collected into useful genetic data.

Programming code is paradata, as it is data that describes how data is collected, which is the definition of paradata [TODO: REF]. Programming code is data that is usually in the form of text, spread over one or more files, that describes the experiment. The experiment collects the data. The data collected is the data we call the results of an experiment.

The article is metadata, as it is data that describes other data, which is the definition of metadata [TODO: REF]: an article describes the experiment and hence the programming code in English. However, it is not the best candidate to describe how the data is collected, as it has a loose connection with collecting the data. Again, if the code and article of an experiment disagree, it is the code that actually let the data be collected. Instead, an article is metadata about a research.

A scientific claim must be reproducible before it is accepted by the scientific community. Reproducibility (i.e. to reproduce the same results) and replicability (i.e. to reconclude a conclusion) are fundamental characteristics of scientific studies ([11]). For computational science, it is relatively easy to reproduce an experiment, as all it takes is a computer, electricity, an optional internet connection, the code and the data. In computational science, however, it appears that culture to reproduce results has been lost and to counter this, it has been suggested to make reproducible research a minimal requirement for publication ([12]).

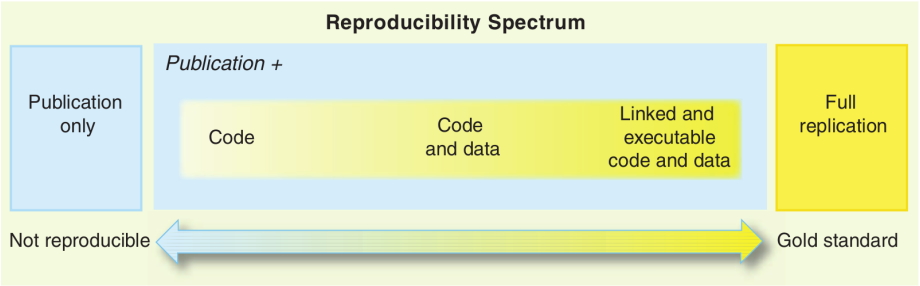

Figure 6: Levels of reproducibility, from [12]

The code of a computational experiment must be published for an experiment to reproducible. It is however (too) common that the original data and/or code are unavailable ([13]). One sad example is described in [14], where an algorithm that detects breast cancer better than a human expert, was irreproducible. In the case of genetic epidemiology, it is a given that the original data cannot be published as this data is sensitive. The code of a computational experiment, however, can be distributed without such problems. It is, however, not code that is published, but an English description of what that code does instead. From a knowledge management's perspective, to conserve an experiment, it is the code that must be archived and not the English description of what it does: it is the code does the actual work. Hence, from a knowledge management perspective, emphasis should be put more on the preservation of code, as it is the most important actor in an experiment.

Code should be published, as it holds the ground truth of an experiment; it does the actual work. The more complex the computation pipeline is, the easier it is to have a mismatch between the article (that describes what the code does) and the code (that actually does the work). It is easy (tempting?) to overlook how easy it is to have a mismatch between paper and code.

To illustrate how easy it is to get a mismatch between a paper and the code, consider this fictional example of a text in a paper:

```
We compared the values of x and y using a one-tailed Student's T-test,
as we expect the average value of x to be less than the average value of y.
```

We ignore the choices of words and style of this sentence: what is important is the content: a one-tailed T test is performed. Taking a look at the (in this case, the programming language R) code, we find the following line:

```
t.test(x, y)
```

The line of code is simplified (as the result of the T test is never stored), but the code is correct: `t.test` is indeed the name of an R function to do a T-test and it is reasonable to assume that this code, when found in the code of an article, is assumed to be correct.

Here, however, we see a mismatch between the English description and the code: by default, the R function `t.test` does a **two**-tailed T-test. A consequence of this fictional example is that the published p-values are higher than needed, resulting in less significant findings, which results in less conclusions drawn from a paper.

It is the paper that accompanies the code, as it is the code that generates the results. When humans are fallible and code gets bigger, the likelihood of a mismatch between the English paper and the code increases. We know that the most common errors are simple (and may legitimately fear that those simple errors are common) ([15]), and this example is not too far off from the examples given in that paper.

But regardless of the size of the code, it is the code that is the ground truth.

4 There are multiple way to publish code

There are multiple ways of publishing code, that are increasingly hard to do, yet result in more reproducible experiments: (1) publishing the code as text

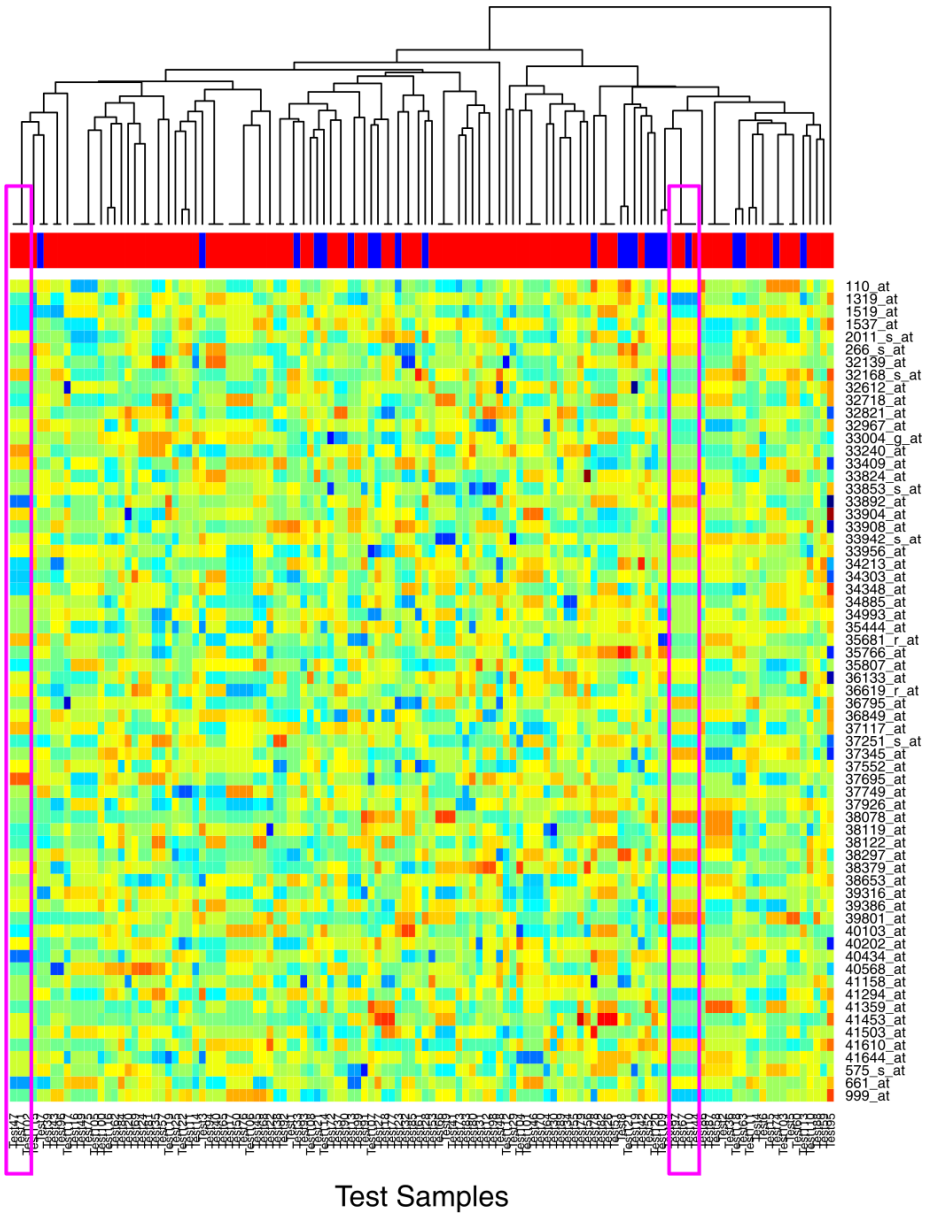

Figure 7: Gene expression result due to an off-by-one error. The cells in the main table show how much mRNA is produced per cell line (the columns) for different proteins (the rows). The colors above the cells, with the dendrogram, shows red for cells that do not respond (red) and do respond (blue) to an antibiotic. The purple rectangles show samples that are duplicated. Note that, due to this, some samples that behave identically are both non-responders and responders. From [15]

only, e.g. by printing it in a Supplementary Materials, (2) publishing the code on a hosted website, e.g on GitHub, with added features (3) publishing the installed code within a virtual environment, for example, using a Singularity container. Here I will go into details of each of these ways.

4.1 Publishing raw code

Code is commonly published alongside of a paper, pasted as text in the Supplementary Materials (for example, in [16]). This is the rawest form of publishing code, as it is just text.

The raw code of a computational experiment is more important to preserve than the article describing the experiment, even when both describe the experiment identically, correctly and reproducibly, as it has/should been that code that generated the results of an experiment. Again: if the paper and code differ, it is the code that did the actual work with the actual/true results.

Raw code gives an indication of the quality of a computational experiment and several metrics are devised to get an idea of the quality of code.

The simplest metric to assess the quality of code is the amount of (single) lines of code (SLOC). Code that has a lower SLOC count, i.e. it is more concise, is usually regarded as having a lower chance of containing bugs. Additionally, following a style guide (e.g. [**?**]), improves software quality ([17]).

A more interesting metric is the cyclomatic complexity of the code. The cyclomatic complexity is defined as the number of ways that the code can be executed. For example, code that has only one `if` statement has a cyclomatic complexity of 2, as the condition within the `if` statement can be true or false. The cyclomatic complexity correlates with code complexity, where more complex code is likelier to contain bugs.

A feature that deserves more attention is that code can actively teach what it

does. A mechanism to do so, which is present in most programming languages, is error handling: errors are the mechanism by which a programmer can teach the user what to do/expect. One example is when calculating the variance of a distribution. To calculate the variance of a distribution, at least two values are required. When the variance of only one value is requested, an error message can helpfully indicate that one needs at least two values.

There are multiple problems with publishing raw code. First, there is no guarantee that that code *can* actually run, for example, due to a typo added after the code was pasted. Secondly, there is no way to determine if code that worked yesterday works today. These problems can be prevented by, instead of pasting raw code, using a website to host that code (see below).

Publishing raw code is already better than not publishing code at all, as that raw code can be assumed to have run successfully at least once. To reproduce an experiment from raw code can be painful, yet way better than rewriting the same code from scratch.

4.2 Publishing hosted code

Code is sometimes published alongside a paper, in the form of a hosted website. This way of publishing code gives additional options, both for the authors of that code, as well as its users.

There are multiple websites for hosting code, with BitBucket, GitHub, GitLab and SourceForge being the most well-known. Of these, GitHub is the most popular website to host code in general, with millions of users. GitHub users can create dedicated websites (called 'repositories') to upload code. GitHub hosts that uploaded by and provides a user interface for every visiter of such a website. The use of code hosting websites, such as GitHub, accommodates collaboration ([18]).

`https://github.com/richelbilderbeek/babette`,

and improves transparency ([19]). An example from the author is the website [20], that hosts part of the code for the paper [21], as shown in Figure 8.

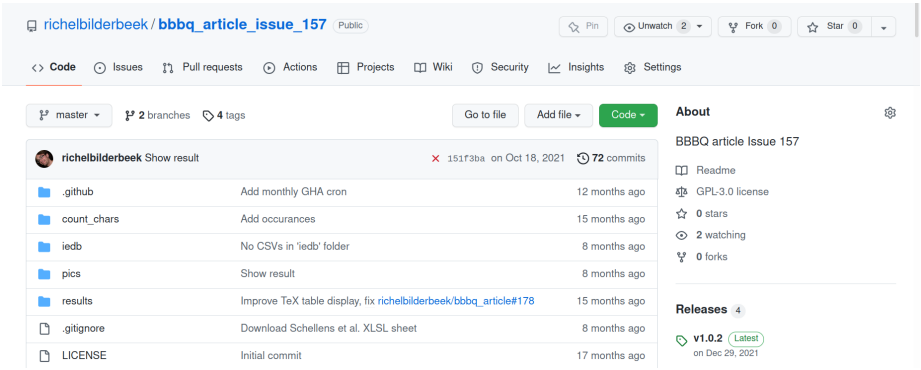

Figure 8: A typical GitHub repository ([20]), hosting the code for a computational experiment, as published with [21]. Note that this repository has 72 commits, where a commit is a change to the code.

Hosted code has many additional features, such as a version-controlled history, continuous integration scripts and automatically generated metrics.

Hosted code commonly has a commit history. For example, GitHub uses a version control system called `git` (hence its name). In practice, this means that when a change is made to the code, a new version is created. In the case that the change was harmful, one can go back to an earlier version and continue again from there.

Hosted code improves transparency ([19]). The history of changes, called the commit history, helps to assess the honesty in the reporting of the code. Ideally, the full version history of the code has been generated, by hosting the code directly at its inception. However, in some repositories, only the final version of the code has been posted, which means all historical information is lost. From a code history, it can be observed, for example, how much statistical tests have really/actually been done and see if all are reported: not reporting

a statistical tests is considered a Dubious Research Practice **[RB: REF]**, as this allows dishonest reporting of p-value hacking.

Some code hosts, among others, GitHub, allows the researcher to start software scripts when uploading a new version. To determine the reproducibility of the code, one such script should run the code (on trivial data) to verify if the code works at all. If the functionality of the code is lost, the researcher can immediatly see which small change was the cause of this. A formalized version of this workflow is called continuous integration (CI). Using a CI service (such as GitHub Actions), is known to significantly increase the number of bugs exposed ([22]) and increases the speed at which new features are added ([22]). Using CI opens up the possibility to informally annotate features of a repository. For example, Figure 9 shows a build badge that indicates that a LaTeX document (i.e. this paper) could be built.

Figure 9: A build badge for the GitHub repository of this article. The description indicates which process it is that the badge signals success (as in this case) or failure of.

A more sophisticated check for the code's correctness is to run units tests and determine the percentage of code that has been tested, which is called the code coverage. Code coverage correlates with code quality ([23, 24]). The non-profit organisation rOpenSci, which peer-reviews code, has made it a prerequisite to have a code coverage of 100%. Also here, CI opens up the possibility to informally annotate the code coverage. For example, Figure 10 shows a build badge that indicates that certain R package's code has a 100% code coverage.

Most code hosts, among others, GitHub and GitLab, automatically keep track of certain metrics. One such metric is the amount of code each contributor has made. In that way, authors involved in the writing of the software are

Figure 10: A build badge for the code coverage of an R package (`https://github.com/ropensci/beautier`). The umbrella signals that the code coverage is hosted by CodeCov (`https://codecov.io/`). The text indicates which process it is that the badge signals, which in this case, is that the code is fully covered by tests.

276 honestly acknowledged. Figure 11 shows the number of commits the authors
277 of DAISIE ([25]) have made to its code.

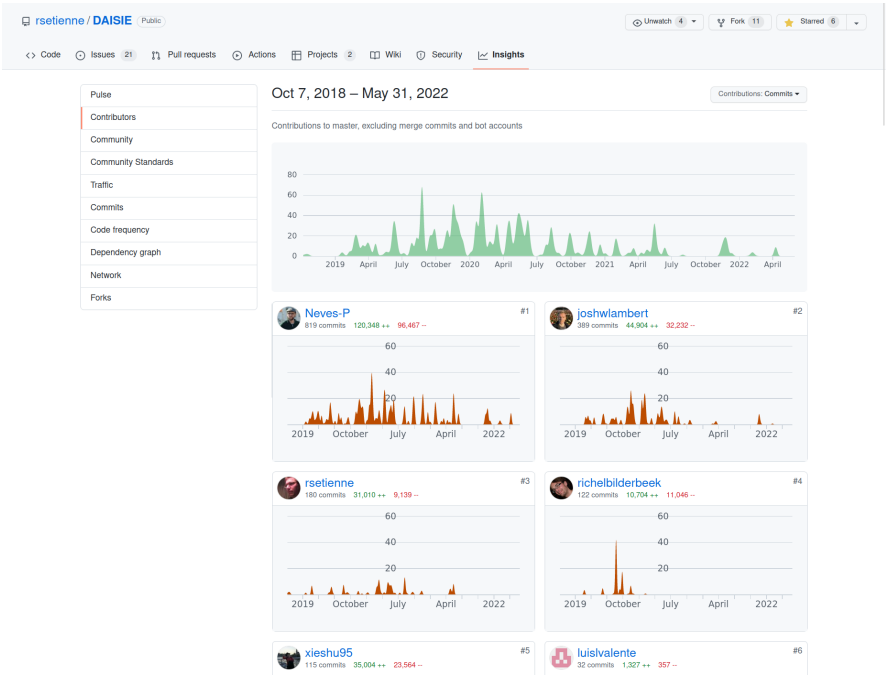

Figure 11: An overview of the commits that multiple contributors have made, in this case, to the code of DAISIE ([25])

278 A feature that gets too few attention is that code hosts allow users to post
279 bug reports and/or ask questions. This allows other researchers to assess the
280 health of the code.

281 Albeith hosted code is vastly superior over raw code in case of reproducib-

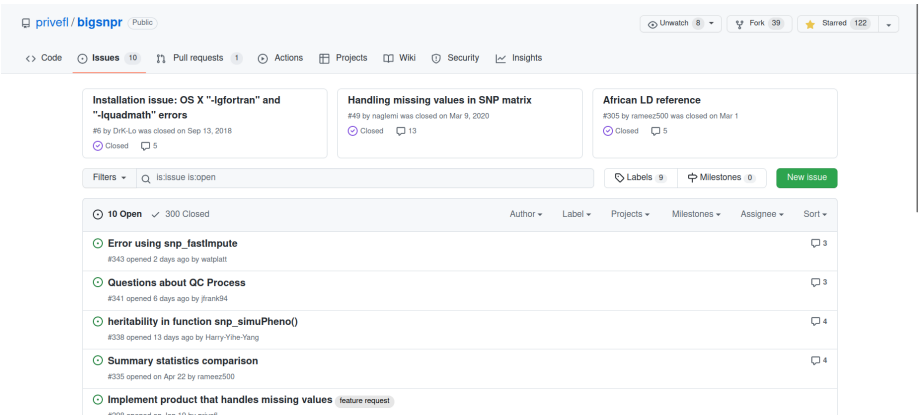

Figure 12: An overview of the issues for the code for LDpred2 ([26]). Of the 310 issues, 300 have been closed. The first four issues have been replied to, as indicated by the talk balloon. The fifth issues has a label with the text 'feature request' to signal the issue type.

lity and correctness, there are some problems with hosted code as well. The most important one is that hosted code needs maintanance: the libraries that are used by an experiment will change over time, inevitably causing the experiment to fail over time. This process is called bit rot and is a fundamental problem **[RB: ref]**. A solution that has more longevity is to publishing the code installed in a virtual environment, as described below.

A lesser problem, which is only a problem to some, is, that the author of the code can be contacted and that bugs can be reported. It can be argued that no energy should be wasted on published code. This is however, a rather selfish point: a bug should be reported and given proper attention, for the sake of the correctness of the results published.

Here are multiple suggestions to make hosted code FAIR. To make the hosted code findable, commonly a paper refers to its URL. At the homepage of hosted code, there is no standardized metadata, to, for example, link to the paper. Using a popular code host ensures the code is easily accessible. There is some interoperabily supplied by a version control manager, such as in obtain-

ing the history of the code. The reusability of code is reasonably easy to assess, by using build badges. However, there is no guarantee that a build badge that claimes success, actually is an honest signal (i.e. that code may not actually build), nor are there standards for what 'success' entails.

4.3 Publishing a virtual environment

The most reproducible way of submitting the code of an experiment, is by providing the code and all it needs installed in a virtual container. This is close to the golden standard suggested by [12] (see also Figure 6), however, that was before virtualization existed. Virtualization allows one to provide a standard environment, that can be used on any other computer. The best known examples of container software are Docker and Singularity, where the latter is the only viable option to be able to run an experiment on a high performance computing (HPC) environment, such as a computer cluster.

Unlike English, code is fragile in standing the test of time. Containers can alleviate this, as these can be created at a point in time and then run later. Because the container is stand-alone, all libraries the code depends on remain in their current/old state. As with hosted code, an experiment may not run with newer versions of libraries anymore, but a containerized experiment can run regardless, if one refrains from updating.

Ideally, to re-do an experiment, one would need a one-liner to a container, for example, `singularity run my_experiment.sif run`.

Containers allow a computation experiment to be highly reproducible: given the same data, an experiment put into a container will give the same results on different platforms, at least in theory. In practice, differences may be observed when peripheral factors are different, such as the random numbers as generated by an operating system, or data that is downloaded from additional online

sources.

Containers allow a computation experiment to be highly re-usable, as any scientist can work with it and tweak code.

There are some problems with providing a container with a runnable container. First, a container is several gigabytes and need to be stored somewhere online. Ideally, this would be at Syslabs, as that one is free, yet there is a limit of 15 Gb. Also, Sylabs does not allow much metadata to be entered, such as the URL of the article, nor the URL of hosted code. Also, journals do not request hosted code, nor runnable code.

Here are multiple suggestions to make containers with runnable code FAIR. To make such containers findable, the most common environments (Syslabs and DockerHub) to upload containers must be indexed. Uploading a container to those environments is, however, limited by a restricted amount of free storage, with the possible consequence that those containers are uploaded somewhere where they escape proper indexing. Having containers uploaded at the existing websites ensures they are accessible, interoperable and reusable. One could argue however, that the scientific community need its own, similar website to upload containers, as the metadata for these sites are limited to free-text comments and tags (DockerHub) or not metadata at all (Syslabs).

When research truly needs to be reproducible, putting the code of an experiment into a container is a good solution, as it can stand the test of time well. Rewarding this, however, is not done yet by journals.

5 Conclusions

The code that does the work in a computational experiment should always be published together with a scholarly article. For research to be reproducible, one ideally has access to both the data used and the code. In some fields, such as

genetic epidemiology, the data is sensitive, hence cannot be released, yet there are methods being devised to run code on sensitive data with assured privacy [27, 28].

Code has additional useful information, similar to confidence intervals, that allow a reader to gauge how much he/she trusts the results. The most important way to determine the quality of code is the amount of unit tests. When following a set of best practices, such as DevOps, TDD, Agile, writing unit tests is an essential part in writing code. The amount of unit tests is an honest signal for code correctness (i.e. it does what it is supposed to do, as opposed to 'it does something'). In academia, to uncover the truth, code correctness is essential. To make a comparison with cell biologists, where working sterile is essential to have the correct results, unit tests allow a computational biologist to have the same.

Code is harder to preserve than an English text.

Although code is the primary actor in computational experiments, there is no incentive to submit code alongside a publication. Most academic journal do not require authors to submit their code, nor it the submitted code peer reviewed.

Although the code of computation experiments can be archived well, there is no incentive to do so.

Although a runnable version of a computation experiment can be archived well, there is no incentive to do so.

6 Discussion

Code may or may not be paradata, depending on how the definition is interpreted. Here we repeat the definition of 'paradata' and discuss its (numbered) constituents. This paper defined paradata as 'data (1) about the collecting (2)

of the data (3)', where (1) code must be seen as data, (2) downloading raw data and doing calculations must be seen as collecting, and (3) the results of an experiment must be seen as data. This paper argues, that (1) code is data in the form of text spread over one or more files, that has useful measurable properties, (2) downloading raw datas and doing calculations, such as a T-test, does describe how the bits and pieces of an end result is collected, and (3) an experimental results is data, as it can be measured and used as the raw data of a next experiment.

Publishing code may be disadvantageous for an author. For science, yes, as this allows reproducible research. For an author, publishing code alongside an experiment opens up the possibily to receive questions regarding that code. Note, however, that not publishing code will always thwart the reproduction of incorrect code, at the cost of a scientific career [15].

Is it worth it to publish version-controlled code? For an author, there is additional training involved, and also here, publishing code alongside an experiment opens up the possibily to receive questions regarding that code.

Is it worth it to archive running versions of code? For an author, there is additional training involved. There is no FAIR infrastructure for Singularity containers.

Being able to re-do an experiment is a core principle of the scientific method. Publishing only a paper about a computational experiment is not enough, as the results are too likely to mismatch that English description. Journal should make it mandatory for authors to publish their code alongside a computational experiment. Authors should follow the FAIR principes for their code as well, as can be done using the infrastructure as supplied by, for example, GitHub and GitLab, opening up new angles in metascience regarding computational experiments. Knowledge managers should create the infrastructure for the

preservation of runnable experiments, to allow scientist to upload a Singularity container, so these are as well following the FAIR principles.

Type of code	Recommendation
No code	Disallow papers that do not supply code publicly
Raw code	Don't publish raw code as text, host it on a code hosting website
Hosted code	Use a standarized badge for build status
	Use a standarized badge for code coverage
	Standardize to link between paper and code host
Runnable code	Allow to upload containers without limits
	Website to host containers must allow for annotation
	Use a standarized badge for the container to be functioning
	Standarize to link between container, paper and code repository

Table 1: Recommendations described in this paper

The world of science would be a more open, humble, trustworthy, truthful and helpful would the code that accompanies a scientific paper be treated like a first class citizen. As doing so in an exemplary way in yet to be rewarded, hence it has to be the idealististic scientists to wage this battle. I feel the truth and science are worth fighting for and I hope this paper helps others to join.

7 Data Accessibility

This article and its metadata can be found at

`https://github.com/richelbilderbeek/chapter_paradata`

.

References

[1] OECD Glossary. The oecd glossary of statistical terms, 2003.

[2] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-

Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

[3] Nicolas Höpfner. Wooden church of Karesuando, Sweden, 2005. URL `https://commons.wikimedia.org/wiki/File:Karesuando_church.jpg`.

[4] Muhammad Ahsan, Weronica E Ek, Mathias Rask-Andersen, Torgny Karlsson, Allan Lind-Thomsen, Stefan Enroth, Ulf Gyllensten, and Åsa Johansson. The relative contribution of dna methylation and genetic variants on protein biomarkers for human diseases. *PLoS genetics*, 13(9): e1007005, 2017.

[5] Wilmar Igl, Åsa Johansson, and Ulf Gyllensten. The northern swedish population health study (nsphs)–a paradigmatic study in a rural population combining community health and basic research. *Rural and remote health*, 10(2):198–215, 2010.

[6] Åsa Johansson, Stefan Enroth, Magnus Palmblad, André M Deelder, Jonas Bergquist, and Ulf Gyllensten. Identification of genetic variants influencing the human plasma proteome. *Proceedings of the National Academy of Sciences*, 110(12):4673–4678, 2013.

[7] Stefan Enroth, Åsa Johansson, Sofia Bosdotter Enroth, and Ulf Gyllensten. Strong effects of genetic and lifestyle factors on biomarker variation and use of personalized cutoffs. *Nature communications*, 5(1):1–11, 2014.

[8] Stefan Enroth, Sofia Bosdotter Enroth, Åsa Johansson, and Ulf Gyllensten. Effect of genetic and environmental factors on protein biomarkers for common non-communicable disease and use of personally normalized plasma protein profiles (pnppp). *Biomarkers*, 20(6-7):355–364, 2015.

[9] Sponk, Tryphon, Magnus Manske, User:Dietzel65, LadyofHats (Mariana Ruiz), and Radio89. Diagram of DNA in a eukaryotic cell, 2012. URL https://commons.wikimedia.org/wiki/File:Eukaryote_DNA-en.svg.

[10] Thomas Shafee. Protein coding genes are transcribed to an mRNA intermediate, then translated to a functional protein. RNA-coding genes are transcribed to a functional non-coding RNA, 2005. URL https://en.wikipedia.org/wiki/File:DNA_to_protein_or_ncRNA.svg.

[11] Prasad Patil, Roger D Peng, and Jeffrey T Leek. A visual tool for defining reproducibility and replicability. *Nature human behaviour*, 3(7):650–652, 2019.

[12] Roger D Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[13] Roger D Peng and Stephanie C Hicks. Reproducible research: A retrospective. *Annual review of public health*, 42:79–93, 2021.

[14] Benjamin Haibe-Kains, GA Adam, Ahmed Hosny, Farnoosh Khodakarami, MS Board, Levi Waldron, Bo Wang, Chris Mcintosh, Anshul Kundaje, Casey Greene, et al. The importance of transparency and reproducibility in artificial intelligence research. 2020, 2020.

[15] Keith A Baggerly and Kevin R Coombes. Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *The Annals of Applied Statistics*, pages 1309–1334, 2009.

[16] Jeremy A Labrecque and Sonja A Swanson. Interpretation and potential biases of mendelian randomization estimates with time-varying exposures. *American journal of epidemiology*, 188(1):231–238, 2019.

[17] Xuefen Fang. Using a coding standard to improve program quality. In *Quality Software, 2001. Proceedings. Second Asia-Pacific Conference on*, pages 73–78. IEEE, 2001.

[18] Yasset Perez-Riverol, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe Leprevost, Christian Fufezan, Tobias Ternent, Stephen J Eglen, Daniel SS Katz, et al. Ten simple rules for taking advantage of git and github. *bioRxiv*, page 048744, 2016.

[19] Krzysztof J Gorgolewski and Russell Poldrack. A practical guide for improving transparency and reproducibility in neuroimaging research. *bioRxiv*, page 039354, 2016.

[20] Richèl J. C. Bilderbeek. BBBQ article issue 157. `https://github.com/richelbilderbeek/bbbq_article_issue_157`, 2021. [Accessed 29-May-2022].

[21] Richèl J. C. Bilderbeek, Maksim V. Baranov, Geert van den Bogaart, and Frans Bianchi. Transmembrane helices are an over-presented and evolutionarily conserved source of major histocompatibility complex class i and ii epitopes. *Frontiers in Immunology*, 12, 2022. ISSN 1664-3224. doi: 10.3389/fimmu.2021.763044. URL `https://www.frontiersin.org/article/10.3389/fimmu.2021.763044`.

[22] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 805–816. ACM, 2015.

[23] Joseph R. Horgan, Saul London, and Michael R Lyu. Achieving software quality with testing coverage measures. *Computer*, 27(9):60–69, 1994.

[24] Fabio Del Frate, Praerit Garg, Aditya P Mathur, and Alberto Pasquini. On the correlation between code coverage and software reliability. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pages 124–132. IEEE, 1995.

[25] Rampal Etienne, Luis Lima Valente, Albert B Phillimore, Bart Haegeman, Joshua Lambert, Pedro Neves, Shu Xie, Richèl Bilderbeek, Torsten Hauffe, Giovanni Laudanno, et al. Daisie: Dynamical assembly of islands by speciation, immigration and extinction. 2020.

[26] Florian Privé, Julyan Arbel, and Bjarni J Vilhjálmsson. Ldpred2: better, faster, stronger. *Bioinformatics*, 36(22-23):5424–5431, 2020.

[27] Lifang Zhang, Yan Zheng, and Raimo Kantoa. A review of homomorphic encryption and its applications. In *Proceedings of the 9th EAI International Conference on Mobile Multimedia Communications*, pages 97–106, 2016.

[28] C-A Azencott. Machine learning and genomics: precision medicine versus patient privacy. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2128):20170350, 2018.

A Supplementary materials

A.1 Funding

This chapter has not been supported by funding.