Reproducibility in a Safe Haven

Toronto Data Workshop on Reproducibility
25–26 February 2021

Nick Radcliffe, Ph.D.
Global Open Finance Centre of Excellence
& Department of Maths, University of Edinburgh
& Stochastic Solutions Limited
njr@StochasticSolutions.com

Pei Shan Yu, Ph.D.
Global Open Finance Centre of Excellence,
University of Edinburgh
Pei-Shan.Yu@ed.ac.uk

Introduction: Reproducibility, Reproduction, Correctness, and Testing

Today I'm going to talk about the challenges of reproducibility in a Safe Haven—a highly locked down research environment for working with sensitive data. I'm Nick Radcliffe, giving the talk, and the opinions expressed here are mine, but all of the actual hard work on reproducing results that I discuss were carried out with Pei Shan Yu, who works with me at the Global Open Finance Centre of Excellence in Edinburgh.

R/Academia vs. Python/programming
I'm guessing that a lot of the people here today come from academic and statistical backgrounds where maybe R is the tool of choice, and where the ideas of reproducible research, publishing data and code, occasionally even doing studies that try to reproduce previously published results are quite popular and well developed. I come more from the other, slightly less reputable side of the current "data science" world—I have a physics background; I think about machine learning and AI more than "statistics"; I have run software development teams; I work in Python rather than R; and when I think about quality I come at it more from a software testing perspective, than an academic publishing perspective (though I am also a part time academic the Maths Department at Edinburgh, so I have some experience of publishing).

I think it might be useful to start with a little framing. I presume, ultimately, all of us are interested in reproducibility because we care about the correctness and validity and meaningfulness of our analyses. There are a couple of distinctions I'd like to draw out here:

Reproducibility (in principle) vs. actually reproducing results.
First, there's the distinction between the practice and notion of reproducible research and actually reproduced research, i.e. between making data and code available so that it can in principle be inspected and verified, and that actually happening. I might be wrong about this, but I see a lot more discussion of making analysis reproducible than about actually reproducing analyses, and though I think there are clear benefits even in the practices of reproducibility for correctness, and in the potential for results to be reproduced, it seems to me that the goal should be actual reproduction, rather than mere facilitation of reproducibility.

Test-Driven Development
There's a related concept in free and open-source software like Linux, where there's a saying—sometime's known as Linus's Law, but normally attributed to Eric Raymond[1]

> *"Given enough eyeballs, all bugs are shallow"* [Linus's Law]

which is to say, that bugs won't survive very long or be very hard to fix in open source software. I would argue that there's quite a lot of evidence, particularly from long-standing bugs and security holes in widely used software whose source has been open to many eyes, that this assertion is overly optimistic. It's a kind-of less whimsical version of the philosophical question "if tree falls in a wood and there's no one around to hear it, does it make a sound?"

In software development, one of the biggest changes over the last few decades has been the rise of what's known as *test-driven development*,[2] (TDD), which is really the intersection of all the various so-called "Agile" methodologies. With test-driven development, the broad idea is that before you write any actual code, you first write tests that should pass only if the code is correct. So if you're writing a function to add numbers, you might write tests that check that

$$\begin{aligned} 2 + 2 &= 4 \\ \text{and} \quad -1 + 1 &= 0 \\ \text{and} \quad 10^{100} + 10^{100} &= 2 \times 10^{100} \end{aligned}$$

and so forth. Testing frameworks (often known as "xUnit" testing frameworks) exist that allow such tests to be run before you've even implemented the function, and these initially fail with an error (because you haven't written the function), and then when you write the function they either begin to pass (as you implement the correct functionality) or fail, if you either get the implementation code wrong, or got the expected test result wrong.

And the idea with test-driven development is you write so many tests, particularly focusing on edge cases, that once all the tests pass, you should stop because you've finished. So the tests act as a kind of executable functional specification. And then continuing on, the idea is that if you find a bug, you first write one or more tests that say what should have happened, then you fix the bug, then you stop. And the same if you want to add functionality.

There are two more key parts to the TDD idea.

The first is that after you've implemented something that works *correctly*, you can then safely "refactor" your code to make it more elegant, logical, performant, extensible etc., secure in the knowledge that as long as the tests continue to pass, you're unlikely to be breaking anything.

The second is that you should run your tests often—at least when you change anything, but ideally in a continuous integration system that runs them periodically and whenever any change is made to the code. This way, as soon as any kind of failure appears, you learn about it, which is helpful, because generally the closer you are in time to when a breakage occurs, the easier it is to fix.

I should also mention that a distinction is sometimes made between so-called *unit tests*, that test small, isolated components of a system, such as a single function, and *system* or *integration* tests, that test the function of a larger software system working as a whole. Both are useful and important.

Test-Driven Data Analysis

I've been an advocate of test-driven development for a number of years, and have used it extensively in the development of the Python-based Miró data analysis software that my company produces. A number of years ago, a long-time collaborator of mine, Patrick Surry said something to the effect of "*we believe in test-driven development, so shouldn't we also do test-driven data analysis?*" and it seemed immediately clear that the answer was "yes", even though we didn't really know what that meant. We wrote an article[3] outlining the broad idea, and since then I've worked, with various others, on developing that into a methodology[4] and an open-source Python library—the `tdda`[5] library—that encapsulates ideas for how to bring the idea of test-driven development to data science. Broadly, those ideas come down to two things:

- *constraint-based data validation* (a.k.a. unit tests for data). Here the idea is that for data at every stage of the analysis pipeline—input data, output results and intermediates—we use constraints to describe everything we sensibly can about things that should be true of the data, and use those constraints to validate the data. The TDDA library includes not only code for verifying data from constraints, but also for automatically *generating* constraints by learning from example data.
- *reference tests*, which are the equivalent of system tests for data analysis processes. With data science, the outputs are often to complex to create by hand, and the output artefacts are inconsistent even when the semantic results are identical (because outputs include version numbers, random IDs, environment information, date stamps etc.). The TDDA library provides support for comparing the parts of outputs that must remain identical while allowing some parts to vary either freely or in well-defined ways. It also provides support for cleanly and painlessly updating reference outputs when a bug is found or target behaviour changes.

I'll describe how we use these at GOFCoE in a moment.

GOFCoE and the Safe Haven

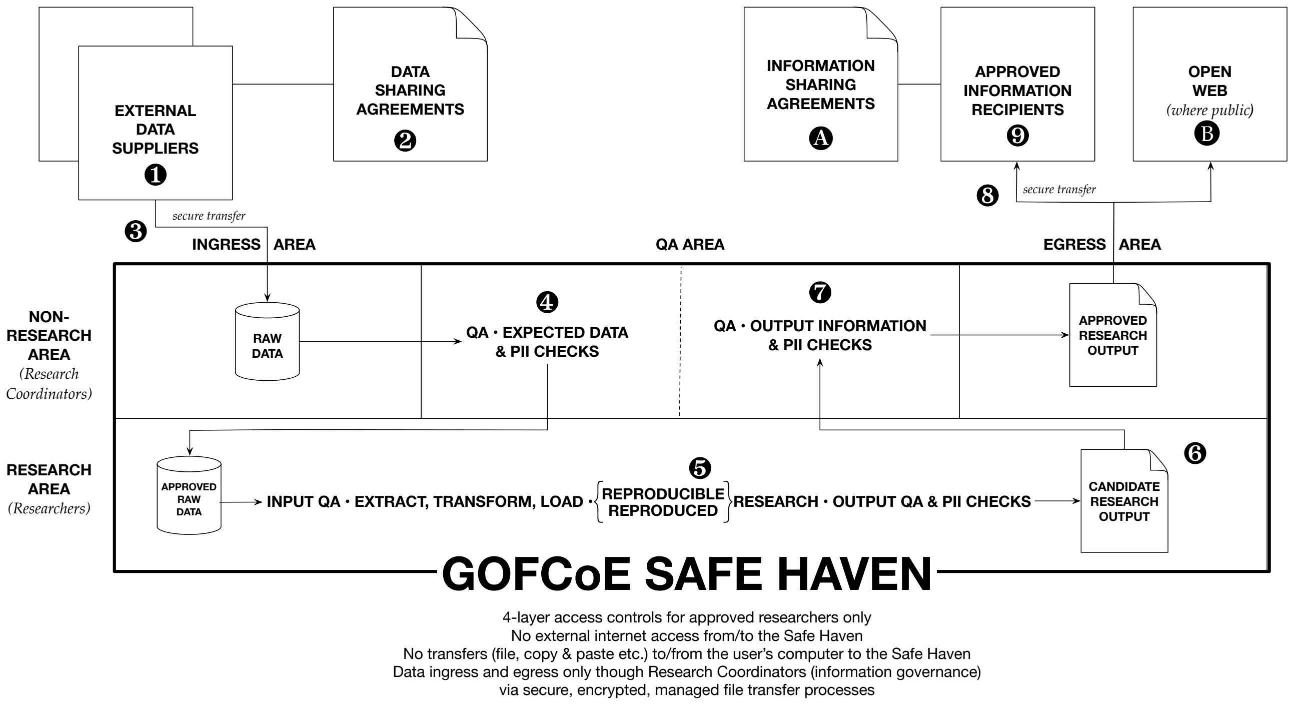
The Global Open Finance Centre of Excellence is an embryonic research institute currently being incubated within the University of Edinburgh, but which we expect to spin out as a non-profit later this year. It seeks to harness the potential of financial data, in Britain and around the world, to improve lives and livelihoods by allowing safe, ethical research on financial data of all kinds in the safe, secure and highly governed research environment of a data Safe Haven. GOFCoE's Safe Haven for Financial Data is run by EPCC, and inherits much of its architecture and governance from the Public Health Scotland Safe Haven, in which approved health researchers access data from NHS patients in Scotland for medical research. GOFCoE is currently engaged in two projects using financial data to analyse the impact of COVID-19 on individuals, households and businesses across Britain, using controlled, de-identified feeds of data from, *inter alia*, banks and online platforms. Our results—but not the underlying data—are provided to various branches of central, national and local government under appropriate information sharing agreements, for the purposes of informing policy development and assessment in the context of the measured financial impact of the pandemic.

Correctness of Analysis.
Naturally, given the importance of this work, it is important that the results we generate are accurate and seen to be accurate—both in the narrow sense of correctly calculating and reporting the things we are trying to calculate and report, and in the large sense of being meaningful and, so far as possible, presented in a way that makes misinterpretation hard or unlikely. In other circumstances, it might seem that a reproducible research methodology would be an ideal way to approach this work, but we are operating in a context where the data—and even the outputs, at this stage—cannot be publicly shared, and for less fundamental reasons, even the code is not yet publicly sharable. Those are the challenging circumstances that form the background for this talk.

To date, every output shared with government has been verified by having different researchers using independent analytical software pipelines to produce results, which have then be verified as identical or equivalent by two separate verification processes.

The GOFCoE Safe Haven

The GOFCoE Safe Haven for Financial Data is shown schematically here.


EXTERNAL DATA SUPPLIERS ❶
DATA SHARING AGREEMENTS ❷
INFORMATION SHARING AGREEMENTS Ⓐ
APPROVED INFORMATION RECIPIENTS ❾
OPEN WEB (where public) Ⓑ
secure transfer
❸
❽
secure transfer
INGRESS AREA
QA AREA
EGRESS AREA
NON-RESEARCH AREA (Research Coordinators)
RAW DATA
❹ QA · EXPECTED DATA & PII CHECKS
❼ QA · OUTPUT INFORMATION & PII CHECKS
APPROVED RESEARCH OUTPUT
RESEARCH AREA (Researchers)
APPROVED RAW DATA
INPUT QA · EXTRACT, TRANSFORM, LOAD ·
❺ REPRODUCIBLE REPRODUCED
RESEARCH · OUTPUT QA & PII CHECKS
❻ CANDIDATE RESEARCH OUTPUT
GOFCoE SAFE HAVEN
4-layer access controls for approved researchers only
No external internet access from/to the Safe Haven
No transfers (file, copy & paste etc.) to/from the user's computer to the Safe Haven
Data ingress and egress only though Research Coordinators (information governance)
via secure, encrypted, managed file transfer processes

In some sense, every part of this picture has something to with reproducibility and testing, but there are three main areas I want to focus on that are central:

1. *Data ingestion and verification*, when we check that data we receive conforms exactly to specification and expectation, and in particular does not contain personal data, anomalies etc.;
2. *Testing analytical code and processes and replicating outputs*;
3. *Output disclosure control*—vetting of outputs, audit trails etc.

Verifying Input Data

When we take (non-public) data into the Safe Haven there is generally an initial upload followed by some kind of regular feed, e.g. weekly. In every case, there is a tight specification of exactly what we will receive, specified in a data sharing agreement, and the data is currently always de-identified, i.e. only pseudo-identifiers exist on the data—there are no names, addresses, real customer numbers etc. Our governance requires that we verify that what we are receiving is exactly what we expected to receive and—most importantly—that there are no unexpected identifiers or other data items that might assist re-identification. Data is only made available to researchers once these checks have been carried out by our research coordinators.

We use the Python `tdda` library[5] (though Miró) for input data verification. The way this works is as follows:

- *Discovery*. On first receipt of the data we use the *discover* functionality within TDDA to generate a set of constraints that are true for some or all of the data received. The result is a JSON file containing constraints on each field and on the set of fields. These generally cover field names and types, minimum and maximum values, uniqueness, presence of missing values and (in the case of text fields), either a list of allowed values (in the case of categorical data) or regular expressions characterizing the patterns in the sample data, if they are not. In the normal case, the constraints generated are satisfied by the training data, though Miró does have options to find constraints that are nearly satisfied by the training data, which can be useful if it is not expected to consist only of valid data. An extract from a constraint file might look like this:

```
{
  "creation_metadata": {
    "as_at": "2021-01-20 13:51:08",
    "local_time": "2021-01-20 13:51:08",
    "utc_time": "2021-01-20 13:51:08",
    "creator": "Miro 3.1.18",
    "creation_command": "discover -o /tmp/tbank",
    “creation_session_log":
        “~/miro/log/2021/01/20/session073.html",
    "source": "banksc-raw",
    "host": "gofcoe01",
    "user": "njr",
    "dataset": banksc-raw",
    "n_records": 205723986,
    "n_selected": 205723986,
    "tddafile": "~/citizendata/qa/banksc.tdda",
    "last_edit": "Nick Radcliffe",
    "version": "1.1",
    "last_edit_date": "2021-02-25 16:00:00"
  },
  "fields": {
    "pseudoid": {
      "type": "string",
      "min_length": 40,
      "max_length": 40,
      "max_nulls": 0,
      "rex": [
        "^([0-9a-z]{40})$",
      ]
    },
    "end_of_previous_period": {
      "type": "date",
      "min": "2018-12-30",
      "max": "2022-01-01",
      "max_nulls": 0
    },
    "end_of_this_period": {
      "type": "date",
      "min": "2019-01-07",
      "max": "2022-01-01",
      "max_nulls": 0
    },
```

```
"end_of_this_period:time-before-now": {
  "min": "6 days",
   "transform": "time-before-now"
},
"end_of_this_period:time-before-now": {
  "min": "0 days",
  "transform": "time-before-now"
},
"postal_district": {
  "type": "string",
  "min_length": 0,
  "max_length": 5,
  "max_nulls": 0,
  "rex": [
     "^([A-Z]{1,2})([0-9]{1,2})([A-Z]?)$"
  ]
},
"sex": {
  "type": "string",
  "min_length": 1,
  "max_length": 1,
  "max_nulls": 0,
  "allowed_values": [
    "F",
    "M"
  ]
},
"total_credits": {
  "type": "real",
  "min": 0,
  "max": 10000000,
  "sign": "non-negative",
  "max_nulls": 0
},
"total_debits": {
  "type": "real",
  "min": -10000000,
  "max": 0,
  "sign": "non-positive",
  "max_nulls": 0
},
```

```
    "final_balance": {
      "type": "real",
      "min": -10000000,
      "max": 10000000,
      "max_nulls": 0
    }
  },
  "dataset": {
    "required_fields": [
      "pseudoid",
      "end_of_previous_period",
      "end_of_this_period",
      "postal_district",
      "sex",
      "total_credits",
      "total_debits",
      "final_balance"
    ],
    "allowed_fields": [
      "pseudoid",
      "end_of_previous_period",
      "end_of_this_period",
      "postal_district",
      "sex",
      "total_credits",
      "total_debits",
      "final_balance"
    ],
    "allowed_all_null_fields": []
  },
  "field_groups": {
    “end_of_previous_period,end_of_this_period": {
      "lt": true
    }
  }
}
```

- *Adaptation*. The generated constraints are then checked against the data specification and sanity and adapted as appropriate. Typical changes include
  - *relaxing constraints* where they have overfitted the training data (e.g. a maximum transaction size of £900,421.22 might be increased to some reasonable upper limit, like £5 million, or £1 billion, or removed altogether); similarly, a maximum age might be increased from 104 to (say) 120, but probably wouldn't be set much higher than that.
  - *tightening or adding constraints* if (bad) outliers have caused them to be too loose or omitted entirely (e.g. adding a uniqueness constraint to an PseudoID column in a master table if a duplicates appeared in the training data, causing such a constraint not to be generated)
  - *tidying up or tightening regular expressions generated for string fields*. Rexpy—the part of TDDA that generates regular expressions from example data—"always" generates correct regular expressions (unless it contains bugs!), but sometimes they are ugly or inefficient, or more permissive than is ideal.
- *Validation*. The full data is then verified using the adapted constraints. If any serious violations of the constraints exist, the data is rejected and re-sourced; if there are only unimportant violations (typically ones that suggest bad data, but not anything that represents a privacy risk or large-scale systematic error that would fundamentally undermine the analysis), these are logged.

Once the constraints have been validated and the data verified, the data is transferred to the research area and made available to the relevant researchers. It should be noted that some kinds of constraints that cannot easily be constructed from the raw data can trivially be so constructed from datasets derived by taking measurements (aggregations, projections etc.) from that data, and the same process can be used on such derived datasets.

As new data arrives periodically, the same constraints are used to verify that, with similar procedures if and when there are violations, including potentially updating constraints if the specification changes or they are found to be too tight, or too loose, or otherwise unhelpful.

Testing analytical pipelines and replicating outputs

The primary approach we have used thus far for engendering confidence that our outputs are correct is to generate them all twice, using entirely independent analysis pipelines, and verifying that the results are either identical or equal to within a small defined tolerances (e.g. 10^{-8}). In the same spirit, we have two independent implementations of the code that verifies results are the same from the two processes, to reduce the non-trivial possibility that a bug in the verification code could fool us into thinking we had reproduced results when we had, in fact, failed to do so.

A second component of our validation involves using test data, where the volumes or nature of the patterns allow us to calculate the expected results by hand, and using reference *testing to* check that our code continues to produce correct results on the test data as that code is developed and enhanced. This aspect is still in development at the time of writing.

The scale of data we are currently working with is large, but not truly massive. For example, in one of the projects we currently have between 100 million and billion data rows, and between a billion and a ten billion data elements in the inputs, meaning that we're at a scale where decomposing the data and processing it in batches is often necessary.

The analysis we are currently doing is mostly measurement, reporting and quantification, rather than modelling, and construction of graphs, tables, maps etc., but does require various careful adjustments, weightings and so forth, so there's plenty of scope for error.

Here is a selection of the problems and discrepancies between the implementations that this replication revealed.

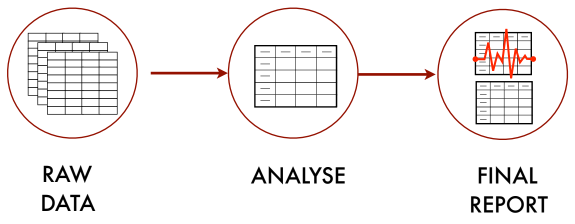
- *Rolling values missing some individuals*. A number of values we calculate involve rolling windows, but some individuals have gaps in their data. The exact strategy for handling these gaps affects the results, and there are several reasonable approaches as well as some less defensible (but tempting) approaches.
- *Discrepancies in handling of null (missing) values.* Such discrepancies can affect both in the missing values in the raw data and nulls generated during intermediate steps in the processing. Again, there are multiple possible approaches, some more appropriate than others.
- *Discrepancies in the handling of invalid values*. Much of the data we are working with is financial data that is machine generated and closely tied to well established mission-critical processing in banks, meaning that errors are relatively unlikely (though of course, errors can still occur, and custom extraction processes are to create our feeds, which are inevitably less well established and testing than core financial processing). Other data, however, is human generated, and here the scope for incorrect and potentially invalid data is always larger. We collect partial postcodes to allow some level of geographic analysis, and we have found a number of non-existent postal areas (the letters at the start of a UK postcode—EH for Edinburgh, WC for West Central London etc.). We have also found invalid postal districts/outcodes that are invalid, even within postal areas that do exist. (Outcodes are the first chunk of the postcode, before the space, e.g. EH11, WC1A etc.) There are obviously many reasonable strategies for handling invalid data such as this, and to achieve the same results, not only does the same strategy have to be applied, but it often has to be applied at the same (logical) processing stage to avoid discrepancies.

- *Weird bug in R's* `write.csv` *function*, which changed the number of decimal places it was writing part way through a file. (We eventually worked around this by switching to a different CSV writing library.)
- *Numerical accuracy*. When aggregating over large populations, numerical errors can accumulate. Much of our data consistent of monetary values consisting of whole penny amounts expressed as floating point values to 2 decimal places. Unfortunately, as is well known, while some decimal values (e.g. 0.50) can be expressed exactly in binary (as 0.1), others such as 0.2 cannot. There a number of ways of dealing with this including (1) using a database or library that has a money or fixed-precision type that handles this automatically, (2) re-expressing everything in pennies and using integer values (being careful, where necessary, about overflow) (3) leaving the values as floating point values but being careful to round values to 2 decimal places are various point in the computation before errors can accumulate.
- *Incorrectly "unrolling" a loop / Copy & paste error*. Some of our analysis involved partitioning the population was divided into "wealth" quintiles, each was to be processed. Rather than looping over the quintiles 1 to 5, five calls to the relevant routine were made, passing quintile numbers 1, 2, 2, 4, 5 (i.e., repeating 2 and omitting 3).
- *Threshold/boundary values*. We used various calculations, including partitioning (binning) and subset membership based on aggregate values. Where boundaries were left on exact penny values (e.g. x ≥ £100.00), a value of £99.999 998 72 would fail to match the criterion, whereas £100.00 or £100.000 001 88 would meet it. In some cases we resolved these discrepancies by using integer values, and in others by careful rounding to prevent propagation of errors.
- *Sort ordering with ties*. Any time there is sorting—implicit or explicit—there is potential for ties, and different systems resolve ties different ways. We were caught by this in some places and ended up having to specify more sort columns for the express purpose of breaking ties consistently. This is also a reason to be careful with over statistics like `mode`, where different systems might break ties in different ways.

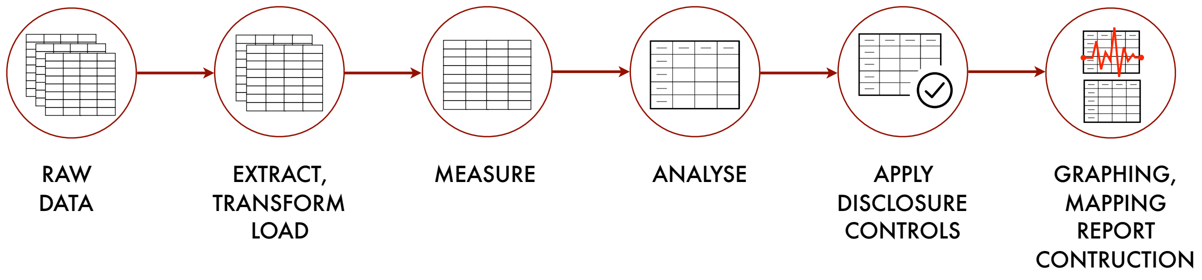
It's worth noting that among the eight discrepancies between implementations, only two were issues that would have led to materially incorrect results (the loop-unrolling/copy and paste error, and the bug in the CSV writer). The other cases were all examples of differences that one would normally not regard as significant, with either approach being likely to produce reasonable answers. Nevertheless, in pretty much every case, the act of attempting to reproduce the calculations in two different systems led us to think more carefully about edge cases and to standardise and specify more completely our analytical approach. While in our *particular* case, only the loop unrolling/copy and paste error would actually have caused misleading results, all have potential to create non-trivial inaccuracies.

Decomposing and Productionizing the Analytical Pipelines

Another important aspect of reproducing the analysis was to break the processing into phases that allow verification at different points. Whereas during prototyping, the analysis was often rather *ad hoc*, with potentially everything happening in a single script (or worse, notebook), like this:



we have moved to a more standardized, scripted process with well-defined stages and intermediate outputs suitable for verification, ideally operated using a `Makefile` with `clean` and `all` targets and proper dependency identification to allow processes to be re-executed reliably.



RAW DATA
EXTRACT, TRANSFORM LOAD
MEASURE
ANALYSE
APPLY DISCLOSURE CONTROLS
GRAPHING, MAPPING REPORT CONTRUCTION

By doing as much of the processing as possible ahead of the final reporting stage (during which we draw graphs and maps and format tables) we maximise the degree to which we can verifiably reproduce results in different systems. We build confidence that the reports, including graphical outputs, are correct by using the reference testing features of the TDDA library to validate that known reference inputs produce hand-validated graphs, and by running these tests continuously as code is committed, we maintain confidence that final outputs are correct.

Output Verification

It is the responsibility of the research team to validate the outputs for correctness, but before outputs are released to government, our research coordinators, who have responsibility for information governance, check that the outputs are safe. There are two primary consideration here.

1. As a further protection against invasions of privacy, we apply minimum cell sizes and censor outputs from populations below those sizes.
2. Notwithstanding the vetting of data as it comes into the Safe Haven, we further vet output for possible identifying data on the way out.

In order to do this, we again use the TDDA functionality to check all of the data that feeds into the final reporting stage, including checking particularly all textual data with categories and regular expression constraints. We also check cell sizes for every cell used in output, and ensure that the cell sizes are included in the final data even in cases where they are not used in the final reports themselves. The reports themselves are also checked by eye, and the scripts used to produce the outputs are also vetted by the research coordinators.

We also keep an audit trail that tracks data provenance by logging every data file used in constructing the final data, complete with their hashes. Alongside these, we record the versions and git commit hashes of the software systems that have been used to produce both the principal output and its reproduced counterpart. The logs have entries something like this:

```
/…/audit/miro/xxxxxx-by-pa-enhanced.csv (hash 2a4cf239fed057281c3a513888492496)
compared to
/…/audit/r/xxxxxx-by-pa-enhanced.csv (hash 59e5540d687daff81f066cc7884efb9c)
validated as equivalent on 2021-01-05T10:34:51Z.
   ⋮
All files produce for report validated successfully.
```

and the files validated in there are those written by audit trails that record the chain of datasets used through the pipeline that generates the reports, and also captured in the `Makefiles` used to run the build process. (`xxxxxx` is an (expurgated!) code prefix that tells us which suppliers' raw data this derives from.)

Although we have not yet achieved this in every case, we are also working towards a situation in which rather than the research team actually producing the output artefacts, the research team hands over the code to the research coordinators who then run it over the appropriate data to produce the outputs, verifying the final data tables against those produced by the research team.

Conclusion

I'll leave you with two thoughts.

The first is about mindset. Louis Heren, a former journalist on *The Times* newspaper of London, said in his memoirs[6] that whenever he was interviewing a politician, the question that went through his mind repeatedly, was "W*hy is this lying bastard lying to me?"* (It has often misattributed —not least by me—to Jeremy Paxman,[7] who once quoted it.) In my view, anyone working with data should have a version of that question going around in his or her head constantly. My own constant thought is

> *"How is this misleading data misleading me?"*

The related thought comes from Blake (Alec Baldwin's character, the corporate trouble shooter and superstar salesman in the film *Glengarry Glen Ross;*[8] not William Blake):

> *"ABC: Always Be Closing".*

My message is really *"always be testing"*, but obviously that doesn't spell ABC, so I guess instead I'll leave you with

> *"ABC: Always Be Checking".*

References

[1] Eric S. Raymond. *The Cathedral and the Bazaar.* O'Reilly Media (Sebastapol, CA, USA) 1999.
[2] Kent Beck. *Test-driven Development: By Example.* Addison Wesley (Boston, MA, USA) 2002.
[3] Patrick Surry & Nick Radcliffe. *Four Ways Data Science Goes Wrong and How Test-Driven Data Analysis Can Help.* Machine Learning Times. December 8, 2015.
http://www.predictiveanalyticsworld.com/machinelearningtimes/four-ways-data-science-goes-wrong-and-how-test-driven-data-analysis-can-help/
[4] TDDA blog. http://tdda/info
[5] TDDA repo on Github, also available from the Python Package index, PyPI, with
`pip install tdda`.
[6] Louis Heren. *Growing Up on the Times,** p.124. Hamish Hamilton (London) 1978.
[7] Matt Wells. *Paxman answers the questions.* The Guardian, 31st January 2005.
https://www.theguardian.com/media/2005/jan/31/mondaymediasection.politicsandthemedia
[8] Glengarry Glen Ross, 1992. (Film, directed by James Foley; written by David Mamet.
https://www.imdb.com/title/tt0104348/ & https://en.wikipedia.org/wiki/Glengarry_Glen_Ross_(film)

* Corrected 26th February 2021. Original transcript referenced *Memories of Times Past*, rather than *Growing Up on the Times,* both by Louis Heren.

Acknowledgements

The work described herein would not have been possible without the support of a number of other people, including GOFCoE data scientists Ian Ozsvald, Giles Weaver and Frank Kelly, all of whom have also been involved in the reproduction and verification of outputs, Sorcha Lorimer and Anna Roubickova, who run our information governance and have been responsible for verifying and validating data as it comes into the Safe Haven, and information ahead of release from the Safe Haven. More broadly, everyone involved in GOFCoE has supported this work in different ways, as have EPCC—which runs both Safe Haven infrastructure both for GOFCoE and Public Health Scotland—the University of Edinburgh, and UKRI—which has provided the bulk of funding to GOFCoE to date.

Transcript and Slides Availability and Status

This transcript is what we *planned* to say at the Toronto Data Workshop on Reproducibility 25–26 February 2021, organised by Rohan Alexander. It is likely to be a little too long for the 25-minute slot, so some sections will probably be skipped or shortened in delivery.

It is available from:

- https://stochasticsolutions.com/pdf/toronto2021-reproducibility-in-safe-haven-transcript.pdf.

The associated slides are available from:

- https://stochasticsolutions.com/pdf/toronto2021-reproducibility-in-safe-haven-slides.pdf

History

26th February 2021. Talk given; initial transcript released 13:00 GMT.
26th February 2021. Louis Heren reference corrected, 15:45 GMT.