



Best Practice in Reproducible Data Science

Matt Forshaw

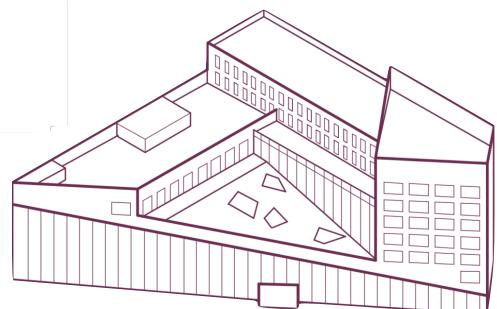
Lecturer, Newcastle University.

Data Skills Lead, The Alan Turing Institute

@mattforshaw

Data Science North East
21st January 2020

**The
Alan Turing
Institute**



*What do you understand about
the meaning of “Reproducibility” or “Open Science”?*

Discussion

“Open science is at a stage where no-one is quite sure what it is, but they think it’s a good idea.”

–Martyn Rittman, Editor at MDPI



Anette Unser, 2016

Attempt to replicate major social scientific findings of past decade fails

https://www.theguardian.com/science/2018/aug/27/attempt-to-replicate-major-social-scientific-findings-of-past-decade-fa...

Support The Guardian Available for everyone, funded by readers

Contribute → Subscribe →

News Opinion Sport Culture Lifestyle More ▾

Education Schools Teachers Universities Students

Science This article is more than 10 months old

Attempt to replicate major social scientific findings of past decade fails

Scientists and the design of experiments under scrutiny after a major project fails to reproduce results of high profile studies

Hannah Devlin
Science correspondent

@hannahdev Mon 27 Aug 2018 16.00 BST

f t e 290



▲ One finding which this study was unable to replicate was that people who viewed a picture of Rodin's sculpture The Thinker subsequently reported weaker religious beliefs. Photograph: Alamy

Some of the most high profile findings in social sciences of the past decade do not stand up to replication, a major investigation has found.

The project, which aimed to repeat 21 experiments that had been published in Science or Nature - science's two preeminent journals - found that only 13 of the original findings could be reproduced.

The research which follows similar efforts in psychology and biomedical

Search jobs Dating Sign in Search UK edition

The Guardian

Editorially independent, open to everyone

We chose a different approach – will you support it?

Support The Guardian →

The screenshot shows a web browser window for the website www.wired.co.uk. The title of the article is "Bug in fMRI software calls 15 years of research into question". Below the title, a sub-headline states: "Popular pieces of software for fMRI were found to have false positive rates up to 70%". The author is listed as "By EMILY REYNOLDS 06 Jul 2016". Below the author information are social media sharing icons for Twitter, Facebook, and Email. The main image of the article is a grid of 12 brain scan slices, each showing a different cross-section of a brain with various colored regions indicating activity or data analysis results.

*"A bug in the software used by researchers to interpret fMRI data could invalidate **fifteen years** worth of neuroscientific research, a paper claims.*

*Three of the most popular pieces of software for fMRI – SPM, FSL and AFNI – were all found to have **false positive rates of up to 70 per cent**. These findings could invalidate "up to **40,000 papers**", researchers claim.*

*"Though fMRI is 25 years old, surprisingly its most common statistical methods have **not been validated** using real data," said Anders Eklund.*

Eklund, A., Nichols, T. E., & Knutsson, H. (2016). Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates. *Proceedings of the National Academy of Sciences*, 201602413. <http://www.pnas.org/content/113/28/7900.abstract>

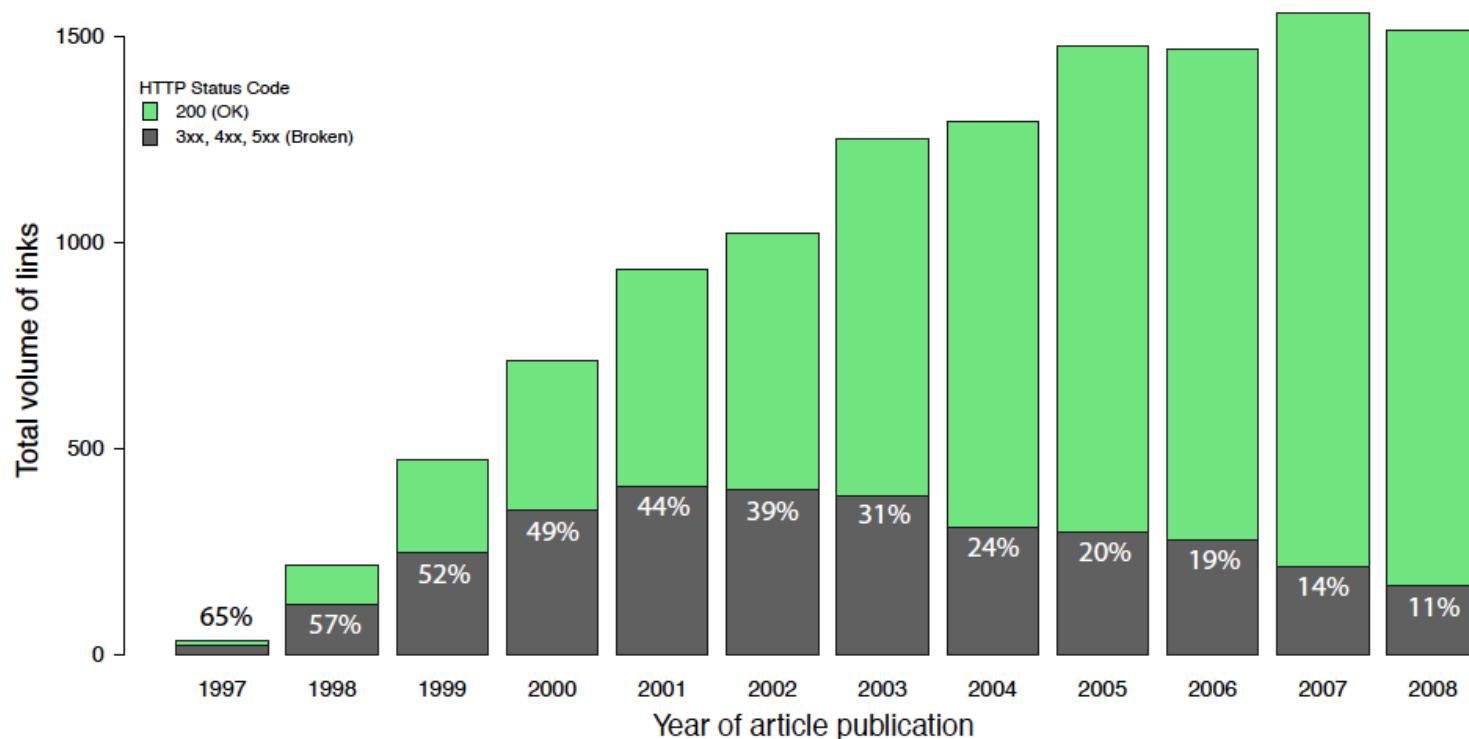


Figure 1. Volume of potential data links in astronomy publications. Total volume of external links in all articles published between 1997 and 2008 in the four main astronomy journals, color coded by HTTP status code. Green bars represent accessible links (200), grey bars represent broken links. .

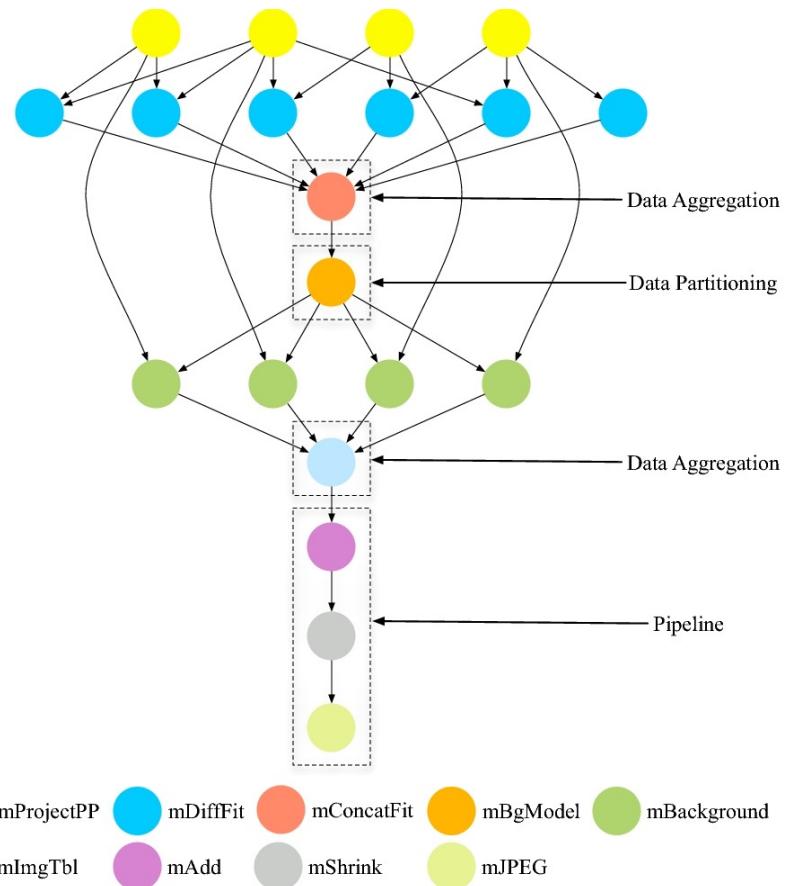
Pepe A, Goodman A, Muench A, Crosas M, Erdmann C (2014) How Do Astronomers Share Data? Reliability and Persistence of Datasets Linked in AAS Publications and a Qualitative Study of Data Practices among US Astronomers. PLoS ONE 9(8): e104798.
<http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0104798>

Zhao, et al:

18 of 92 (19.57%)

Mayer, et al:

341 of 1443 (23.63%)



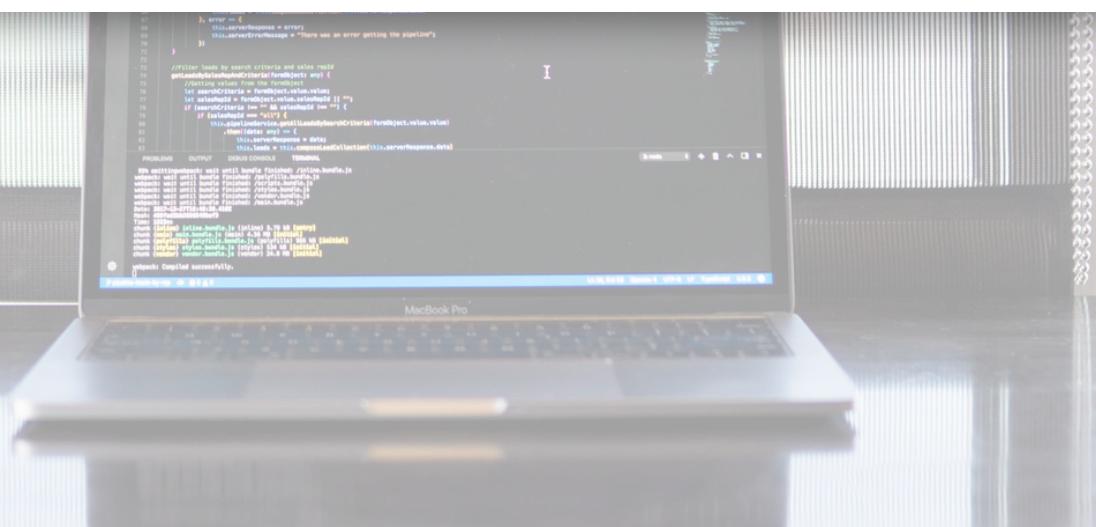
Zhao, Jun, et al. "Why workflows break—Understanding and combating decay in Taverna workflows." E-Science (e-Science), 2012 IEEE 8th International Conference on. IEEE, 2012.

<http://ieeexplore.ieee.org/document/6404482/>

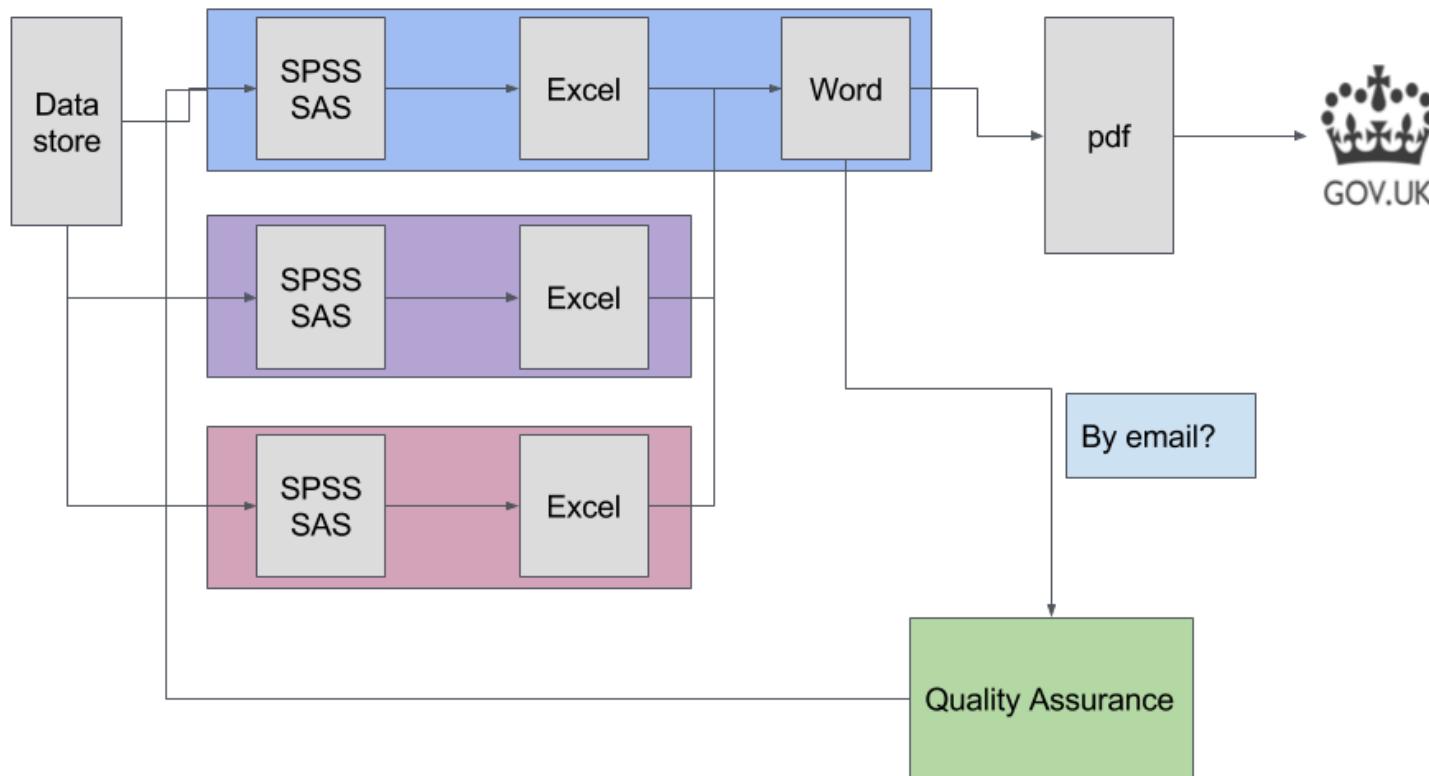
Mayer, Rudolf, and Andreas Rauber. "A quantitative study on the re-executability of publicly shared scientific workflows." e-Science (e-Science), 2015 IEEE 11th International Conference on. IEEE, 2015.

<http://ieeexplore.ieee.org/abstract/document/7304314/>

Review code snippet (3 mins)

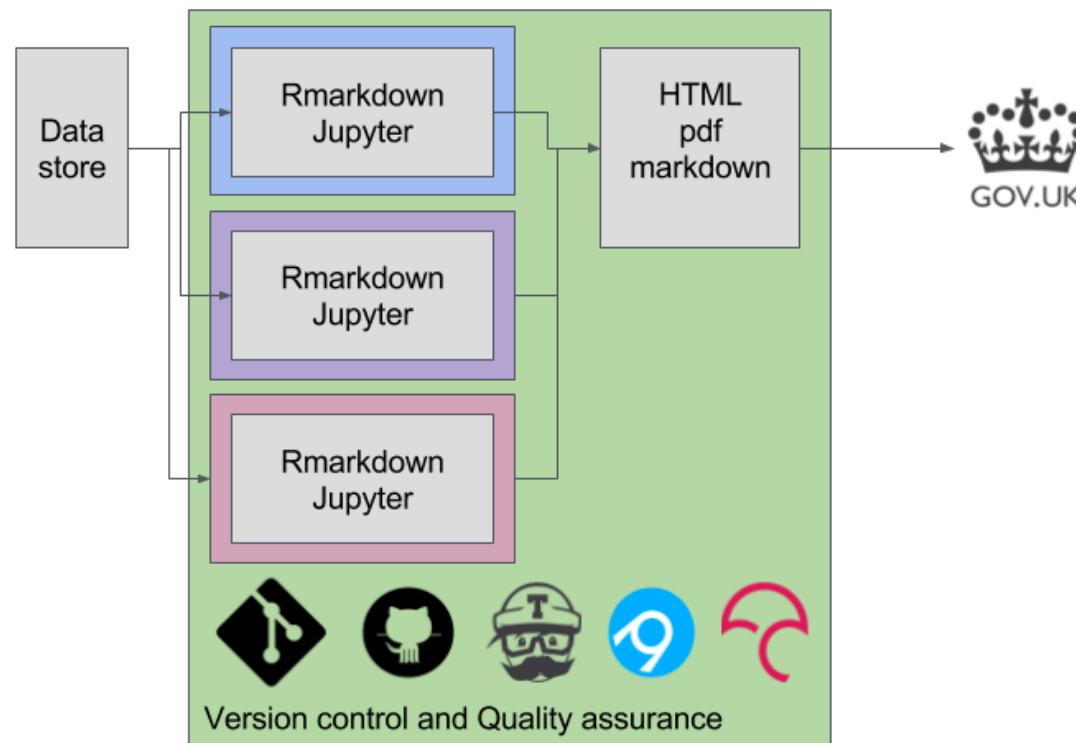


A case study of gov.uk



https://ukgovdatascience.github.io/rap_companion/why.html#the-current-statistics-production-process

A case study of gov.uk



https://ukgovdatascience.github.io/rap_companion/why.html#the-current-statistics-production-process

GitHub, Inc. [US] | https://github.com/ukgovdatascience/rap_companion/blob/master/README.md

Search or jump to... Pull requests Issues Marketplace Explore

ukgovdatascience / rap_companion Watch 6 Star 19 Fork 1

Code Issues 14 Pull requests 1 Projects 0 Wiki Insights

Branch: master rap_companion / README.md Find file Copy path

mammykins Add MOOC link to close issue 70 (#71) 2b6a8c9 on 19 Feb

2 contributors

33 lines (19 sloc) 2.66 KB Raw Blame History

build passing

Reproducible Analytical Pipeline Companion

This is a prototype and subject to constant development

A technical communication document written using bookdown intended to give assistance to people developing a Reproducible Analytical Pipeline.

This document is intended to be RAP community maintained. If you discover any good resources associated with RAP, please contribute to the development of this book by forking and pulling on Github.

See the [eesectorsmarkdown](#) repository for an example of implementing RAP as an R package in the context of a Statistical First Release (SFR).

Learning to RAP

To complement this book, one of our RAP Champions has developed a [Massive Online Open Course](#) to share an approach to learning this technical skill-set. This course is an informal introduction and describes the best practices through the use of screencasts and assignments. It is currently available on [Udemy](#) and takes you through the RAP journey using a simple RAP example.

Collaboration

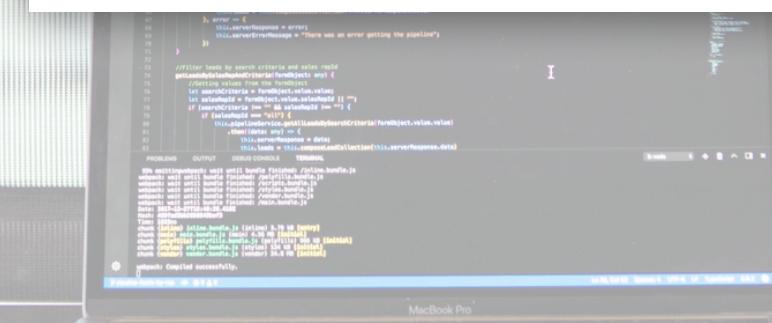
This is a community effort produced by those interested in automating the production of statistical reports in government. If you wish to contribute refer to the [CONTRIBUTING.md](#).



https://github.com/ukgovdatascience/rap_companion/pulls

https://github.com/ukgovdatascience/rap_companion

Live Demo

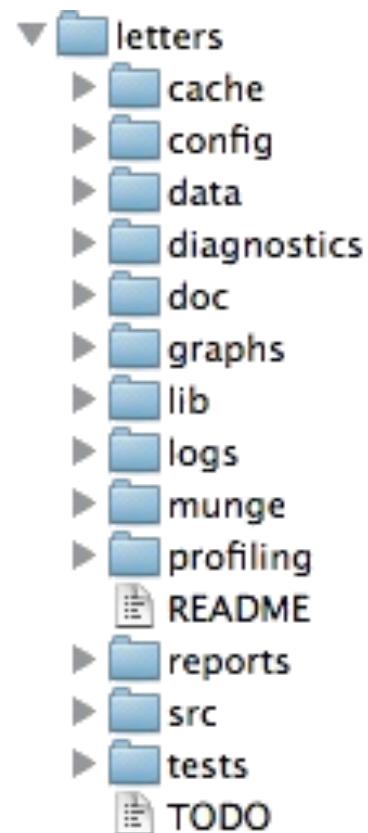


A MacBook Pro laptop is shown from a slightly elevated angle, displaying a Java code editor and a terminal window. The code editor shows a Java file with annotations like @Entity and @ManyToOne. The terminal window shows the output of a command-line interface, likely related to the code being demonstrated.

```
31     } catch (Exception e) {
32         this.serverResponse = error;
33         this.serverErrorMessage = "There was an error getting the object!";
34     }
35 }
36
37 //filter: looks for search criteria and sets mode
38 protected void filterSearchCriteriaAndSetMode(
39     List<Object> filteredObjects) {
40     //Getting values from the formobject
41     String searchCriteria = formobject.getCriteria();
42     List<Object> selected = formobject.getValues();
43     for (Object o : selected) {
44         if (o instanceof Criterion) {
45             if (((Criterion)o).getMode() == null) {
46                 ((Criterion)o).setMode("EQ");
47             }
48         }
49     }
50 }
51
52 //filter: looks for search criteria and sets mode
53 protected void filterSearchCriteriaAndSetMode(
54     List<Object> filteredObjects, List<Object> values) {
55     //Getting values from the formobject
56     String searchCriteria = formobject.getCriteria();
57     List<Object> selected = formobject.getValues();
58     for (Object o : selected) {
59         if (o instanceof Criterion) {
60             if (((Criterion)o).getMode() == null) {
61                 ((Criterion)o).setMode("EQ");
62             }
63         }
64     }
65 }
66
67 /**
68 * @param id
69 * @return
70 */
71 @Override
72 public Object getOneObject(Object id) {
73     EntityManager em = getEntityManager();
74     CriteriaBuilder cb = em.getCriteriaBuilder();
75     CriteriaQuery<Object> cq = cb.createQuery();
76     Root<Object> root = cq.from(em.getMetamodel().entity(id));
77     cq.select(root);
78     cq.where(cb.equal(root.get("id"), id));
79     return em.createQuery(cq).getSingleResult();
80 }
81
82 /**
83 * @param id
84 * @param value
85 * @return
86 */
87 @Override
88 public Object updateObject(Object id, Object value) {
89     EntityManager em = getEntityManager();
90     Object object = em.find(em.getMetamodel().entity(id));
91     if (object != null) {
92         em.merge(object);
93         em.flush();
94     }
95     return object;
96 }
97
98 /**
99 * @param id
100 * @param value
101 * @return
102 */
103 @Override
104 public Object deleteObject(Object id, Object value) {
105     EntityManager em = getEntityManager();
106     Object object = em.find(em.getMetamodel().entity(id));
107     if (object != null) {
108         em.remove(em.merge(object));
109         em.flush();
110     }
111     return object;
112 }
113
114 /**
115 * @param id
116 * @param value
117 * @return
118 */
119 @Override
120 public Object addObject(Object id, Object value) {
121     EntityManager em = getEntityManager();
122     Object object = em.find(em.getMetamodel().entity(id));
123     if (object != null) {
124         em.persist(em.merge(object));
125         em.flush();
126     }
127     return object;
128 }
129
130 /**
131 * @param id
132 * @param value
133 * @return
134 */
135 @Override
136 public Object addObjectList(Object id, List<Object> value) {
137     EntityManager em = getEntityManager();
138     Object object = em.find(em.getMetamodel().entity(id));
139     if (object != null) {
140         em.persist(em.merge(object));
141         em.flush();
142     }
143     return object;
144 }
145
146 /**
147 * @param id
148 * @param value
149 * @return
150 */
151 @Override
152 public Object removeObjectList(Object id, List<Object> value) {
153     EntityManager em = getEntityManager();
154     Object object = em.find(em.getMetamodel().entity(id));
155     if (object != null) {
156         em.remove(em.merge(object));
157         em.flush();
158     }
159     return object;
160 }
161
162 /**
163 * @param id
164 * @param value
165 * @return
166 */
167 @Override
168 public Object removeObject(Object id, Object value) {
169     EntityManager em = getEntityManager();
170     Object object = em.find(em.getMetamodel().entity(id));
171     if (object != null) {
172         em.remove(em.merge(object));
173         em.flush();
174     }
175     return object;
176 }
177
178 /**
179 * @param id
180 * @param value
181 * @return
182 */
183 @Override
184 public Object removeObjectList(Object id, List<Object> value) {
185     EntityManager em = getEntityManager();
186     Object object = em.find(em.getMetamodel().entity(id));
187     if (object != null) {
188         em.remove(em.merge(object));
189         em.flush();
190     }
191     return object;
192 }
193
194 /**
195 * @param id
196 * @param value
197 * @return
198 */
199 @Override
200 public Object removeObject(Object id, Object value) {
201     EntityManager em = getEntityManager();
202     Object object = em.find(em.getMetamodel().entity(id));
203     if (object != null) {
204         em.remove(em.merge(object));
205         em.flush();
206     }
207     return object;
208 }
209
210 /**
211 * @param id
212 * @param value
213 * @return
214 */
215 @Override
216 public Object addObjectList(Object id, List<Object> value) {
217     EntityManager em = getEntityManager();
218     Object object = em.find(em.getMetamodel().entity(id));
219     if (object != null) {
220         em.persist(em.merge(object));
221         em.flush();
222     }
223     return object;
224 }
225
226 /**
227 * @param id
228 * @param value
229 * @return
230 */
231 @Override
232 public Object removeObjectList(Object id, List<Object> value) {
233     EntityManager em = getEntityManager();
234     Object object = em.find(em.getMetamodel().entity(id));
235     if (object != null) {
236         em.remove(em.merge(object));
237         em.flush();
238     }
239     return object;
240 }
241
242 /**
243 * @param id
244 * @param value
245 * @return
246 */
247 @Override
248 public Object removeObjectList(Object id, List<Object> value) {
249     EntityManager em = getEntityManager();
250     Object object = em.find(em.getMetamodel().entity(id));
251     if (object != null) {
252         em.remove(em.merge(object));
253         em.flush();
254     }
255     return object;
256 }
257
258 /**
259 * @param id
260 * @param value
261 * @return
262 */
263 @Override
264 public Object removeObjectList(Object id, List<Object> value) {
265     EntityManager em = getEntityManager();
266     Object object = em.find(em.getMetamodel().entity(id));
267     if (object != null) {
268         em.remove(em.merge(object));
269         em.flush();
270     }
271     return object;
272 }
273
274 /**
275 * @param id
276 * @param value
277 * @return
278 */
279 @Override
280 public Object removeObjectList(Object id, List<Object> value) {
281     EntityManager em = getEntityManager();
282     Object object = em.find(em.getMetamodel().entity(id));
283     if (object != null) {
284         em.remove(em.merge(object));
285         em.flush();
286     }
287     return object;
288 }
289
290 /**
291 * @param id
292 * @param value
293 * @return
294 */
295 @Override
296 public Object removeObjectList(Object id, List<Object> value) {
297     EntityManager em = getEntityManager();
298     Object object = em.find(em.getMetamodel().entity(id));
299     if (object != null) {
300         em.remove(em.merge(object));
301         em.flush();
302     }
303     return object;
304 }
305
306 /**
307 * @param id
308 * @param value
309 * @return
310 */
311 @Override
312 public Object removeObjectList(Object id, List<Object> value) {
313     EntityManager em = getEntityManager();
314     Object object = em.find(em.getMetamodel().entity(id));
315     if (object != null) {
316         em.remove(em.merge(object));
317         em.flush();
318     }
319     return object;
320 }
321
322 /**
323 * @param id
324 * @param value
325 * @return
326 */
327 @Override
328 public Object removeObjectList(Object id, List<Object> value) {
329     EntityManager em = getEntityManager();
330     Object object = em.find(em.getMetamodel().entity(id));
331     if (object != null) {
332         em.remove(em.merge(object));
333         em.flush();
334     }
335     return object;
336 }
337
338 /**
339 * @param id
340 * @param value
341 * @return
342 */
343 @Override
344 public Object removeObjectList(Object id, List<Object> value) {
345     EntityManager em = getEntityManager();
346     Object object = em.find(em.getMetamodel().entity(id));
347     if (object != null) {
348         em.remove(em.merge(object));
349         em.flush();
350     }
351     return object;
352 }
353
354 /**
355 * @param id
356 * @param value
357 * @return
358 */
359 @Override
360 public Object removeObjectList(Object id, List<Object> value) {
361     EntityManager em = getEntityManager();
362     Object object = em.find(em.getMetamodel().entity(id));
363     if (object != null) {
364         em.remove(em.merge(object));
365         em.flush();
366     }
367     return object;
368 }
369
370 /**
371 * @param id
372 * @param value
373 * @return
374 */
375 @Override
376 public Object removeObjectList(Object id, List<Object> value) {
377     EntityManager em = getEntityManager();
378     Object object = em.find(em.getMetamodel().entity(id));
379     if (object != null) {
380         em.remove(em.merge(object));
381         em.flush();
382     }
383     return object;
384 }
385
386 /**
387 * @param id
388 * @param value
389 * @return
390 */
391 @Override
392 public Object removeObjectList(Object id, List<Object> value) {
393     EntityManager em = getEntityManager();
394     Object object = em.find(em.getMetamodel().entity(id));
395     if (object != null) {
396         em.remove(em.merge(object));
397         em.flush();
398     }
399     return object;
400 }
401
402 /**
403 * @param id
404 * @param value
405 * @return
406 */
407 @Override
408 public Object removeObjectList(Object id, List<Object> value) {
409     EntityManager em = getEntityManager();
410     Object object = em.find(em.getMetamodel().entity(id));
411     if (object != null) {
412         em.remove(em.merge(object));
413         em.flush();
414     }
415     return object;
416 }
417
418 /**
419 * @param id
420 * @param value
421 * @return
422 */
423 @Override
424 public Object removeObjectList(Object id, List<Object> value) {
425     EntityManager em = getEntityManager();
426     Object object = em.find(em.getMetamodel().entity(id));
427     if (object != null) {
428         em.remove(em.merge(object));
429         em.flush();
430     }
431     return object;
432 }
433
434 /**
435 * @param id
436 * @param value
437 * @return
438 */
439 @Override
440 public Object removeObjectList(Object id, List<Object> value) {
441     EntityManager em = getEntityManager();
442     Object object = em.find(em.getMetamodel().entity(id));
443     if (object != null) {
444         em.remove(em.merge(object));
445         em.flush();
446     }
447     return object;
448 }
449
450 /**
451 * @param id
452 * @param value
453 * @return
454 */
455 @Override
456 public Object removeObjectList(Object id, List<Object> value) {
457     EntityManager em = getEntityManager();
458     Object object = em.find(em.getMetamodel().entity(id));
459     if (object != null) {
460         em.remove(em.merge(object));
461         em.flush();
462     }
463     return object;
464 }
465
466 /**
467 * @param id
468 * @param value
469 * @return
470 */
471 @Override
472 public Object removeObjectList(Object id, List<Object> value) {
473     EntityManager em = getEntityManager();
474     Object object = em.find(em.getMetamodel().entity(id));
475     if (object != null) {
476         em.remove(em.merge(object));
477         em.flush();
478     }
479     return object;
480 }
481
482 /**
483 * @param id
484 * @param value
485 * @return
486 */
487 @Override
488 public Object removeObjectList(Object id, List<Object> value) {
489     EntityManager em = getEntityManager();
490     Object object = em.find(em.getMetamodel().entity(id));
491     if (object != null) {
492         em.remove(em.merge(object));
493         em.flush();
494     }
495     return object;
496 }
497
498 /**
499 * @param id
500 * @param value
501 * @return
502 */
503 @Override
504 public Object removeObjectList(Object id, List<Object> value) {
505     EntityManager em = getEntityManager();
506     Object object = em.find(em.getMetamodel().entity(id));
507     if (object != null) {
508         em.remove(em.merge(object));
509         em.flush();
510     }
511     return object;
512 }
513
514 /**
515 * @param id
516 * @param value
517 * @return
518 */
519 @Override
520 public Object removeObjectList(Object id, List<Object> value) {
521     EntityManager em = getEntityManager();
522     Object object = em.find(em.getMetamodel().entity(id));
523     if (object != null) {
524         em.remove(em.merge(object));
525         em.flush();
526     }
527     return object;
528 }
529
530 /**
531 * @param id
532 * @param value
533 * @return
534 */
535 @Override
536 public Object removeObjectList(Object id, List<Object> value) {
537     EntityManager em = getEntityManager();
538     Object object = em.find(em.getMetamodel().entity(id));
539     if (object != null) {
540         em.remove(em.merge(object));
541         em.flush();
542     }
543     return object;
544 }
545
546 /**
547 * @param id
548 * @param value
549 * @return
550 */
551 @Override
552 public Object removeObjectList(Object id, List<Object> value) {
553     EntityManager em = getEntityManager();
554     Object object = em.find(em.getMetamodel().entity(id));
555     if (object != null) {
556         em.remove(em.merge(object));
557         em.flush();
558     }
559     return object;
560 }
561
562 /**
563 * @param id
564 * @param value
565 * @return
566 */
567 @Override
568 public Object removeObjectList(Object id, List<Object> value) {
569     EntityManager em = getEntityManager();
570     Object object = em.find(em.getMetamodel().entity(id));
571     if (object != null) {
572         em.remove(em.merge(object));
573         em.flush();
574     }
575     return object;
576 }
577
578 /**
579 * @param id
580 * @param value
581 * @return
582 */
583 @Override
584 public Object removeObjectList(Object id, List<Object> value) {
585     EntityManager em = getEntityManager();
586     Object object = em.find(em.getMetamodel().entity(id));
587     if (object != null) {
588         em.remove(em.merge(object));
589         em.flush();
590     }
591     return object;
592 }
593
594 /**
595 * @param id
596 * @param value
597 * @return
598 */
599 @Override
600 public Object removeObjectList(Object id, List<Object> value) {
601     EntityManager em = getEntityManager();
602     Object object = em.find(em.getMetamodel().entity(id));
603     if (object != null) {
604         em.remove(em.merge(object));
605         em.flush();
606     }
607     return object;
608 }
609
610 /**
611 * @param id
612 * @param value
613 * @return
614 */
615 @Override
616 public Object removeObjectList(Object id, List<Object> value) {
617     EntityManager em = getEntityManager();
618     Object object = em.find(em.getMetamodel().entity(id));
619     if (object != null) {
620         em.remove(em.merge(object));
621         em.flush();
622     }
623     return object;
624 }
625
626 /**
627 * @param id
628 * @param value
629 * @return
630 */
631 @Override
632 public Object removeObjectList(Object id, List<Object> value) {
633     EntityManager em = getEntityManager();
634     Object object = em.find(em.getMetamodel().entity(id));
635     if (object != null) {
636         em.remove(em.merge(object));
637         em.flush();
638     }
639     return object;
640 }
641
642 /**
643 * @param id
644 * @param value
645 * @return
646 */
647 @Override
648 public Object removeObjectList(Object id, List<Object> value) {
649     EntityManager em = getEntityManager();
650     Object object = em.find(em.getMetamodel().entity(id));
651     if (object != null) {
652         em.remove(em.merge(object));
653         em.flush();
654     }
655     return object;
656 }
657
658 /**
659 * @param id
660 * @param value
661 * @return
662 */
663 @Override
664 public Object removeObjectList(Object id, List<Object> value) {
665     EntityManager em = getEntityManager();
666     Object object = em.find(em.getMetamodel().entity(id));
667     if (object != null) {
668         em.remove(em.merge(object));
669         em.flush();
670     }
671     return object;
672 }
673
674 /**
675 * @param id
676 * @param value
677 * @return
678 */
679 @Override
680 public Object removeObjectList(Object id, List<Object> value) {
681     EntityManager em = getEntityManager();
682     Object object = em.find(em.getMetamodel().entity(id));
683     if (object != null) {
684         em.remove(em.merge(object));
685         em.flush();
686     }
687     return object;
688 }
689
690 /**
691 * @param id
692 * @param value
693 * @return
694 */
695 @Override
696 public Object removeObjectList(Object id, List<Object> value) {
697     EntityManager em = getEntityManager();
698     Object object = em.find(em.getMetamodel().entity(id));
699     if (object != null) {
700         em.remove(em.merge(object));
701         em.flush();
702     }
703     return object;
704 }
705
706 /**
707 * @param id
708 * @param value
709 * @return
710 */
711 @Override
712 public Object removeObjectList(Object id, List<Object> value) {
713     EntityManager em = getEntityManager();
714     Object object = em.find(em.getMetamodel().entity(id));
715     if (object != null) {
716         em.remove(em.merge(object));
717         em.flush();
718     }
719     return object;
720 }
721
722 /**
723 * @param id
724 * @param value
725 * @return
726 */
727 @Override
728 public Object removeObjectList(Object id, List<Object> value) {
729     EntityManager em = getEntityManager();
730     Object object = em.find(em.getMetamodel().entity(id));
731     if (object != null) {
732         em.remove(em.merge(object));
733         em.flush();
734     }
735     return object;
736 }
737
738 /**
739 * @param id
740 * @param value
741 * @return
742 */
743 @Override
744 public Object removeObjectList(Object id, List<Object> value) {
745     EntityManager em = getEntityManager();
746     Object object = em.find(em.getMetamodel().entity(id));
747     if (object != null) {
748         em.remove(em.merge(object));
749         em.flush();
750     }
751     return object;
752 }
753
754 /**
755 * @param id
756 * @param value
757 * @return
758 */
759 @Override
760 public Object removeObjectList(Object id, List<Object> value) {
761     EntityManager em = getEntityManager();
762     Object object = em.find(em.getMetamodel().entity(id));
763     if (object != null) {
764         em.remove(em.merge(object));
765         em.flush();
766     }
767     return object;
768 }
769
770 /**
771 * @param id
772 * @param value
773 * @return
774 */
775 @Override
776 public Object removeObjectList(Object id, List<Object> value) {
777     EntityManager em = getEntityManager();
778     Object object = em.find(em.getMetamodel().entity(id));
779     if (object != null) {
780         em.remove(em.merge(object));
781         em.flush();
782     }
783     return object;
784 }
```

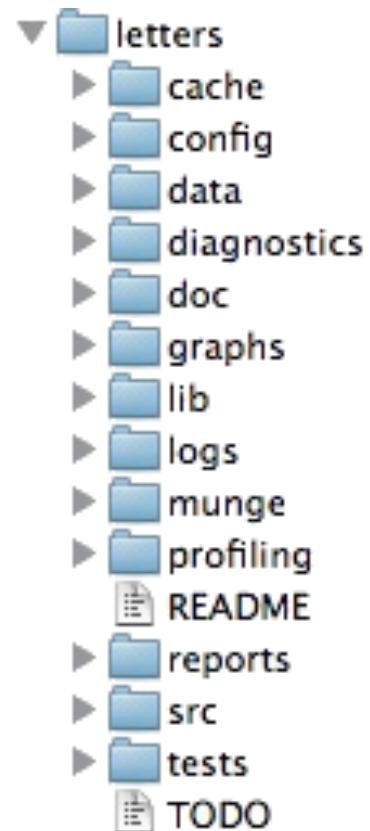
What is ProjectTemplate?

- ProjectTemplate is a system for automating the thoughtless parts of a data analysis project:
 - Organising the files in your project.
 - Loading all the R packages you'll use.
 - Loading all of your data sets into memory.
 - Munging and preprocessing your data into a form that's suitable for analysis.



Benefits

1. Documents and automatically install any dependencies used by your analysis.
2. Caches intermediate results
3. Gives you a common structure to locate your files.
4. Avoids drift between code and workspace contents.



Creating a Project

```
library('ProjectTemplate')
create.project('new_project')
```

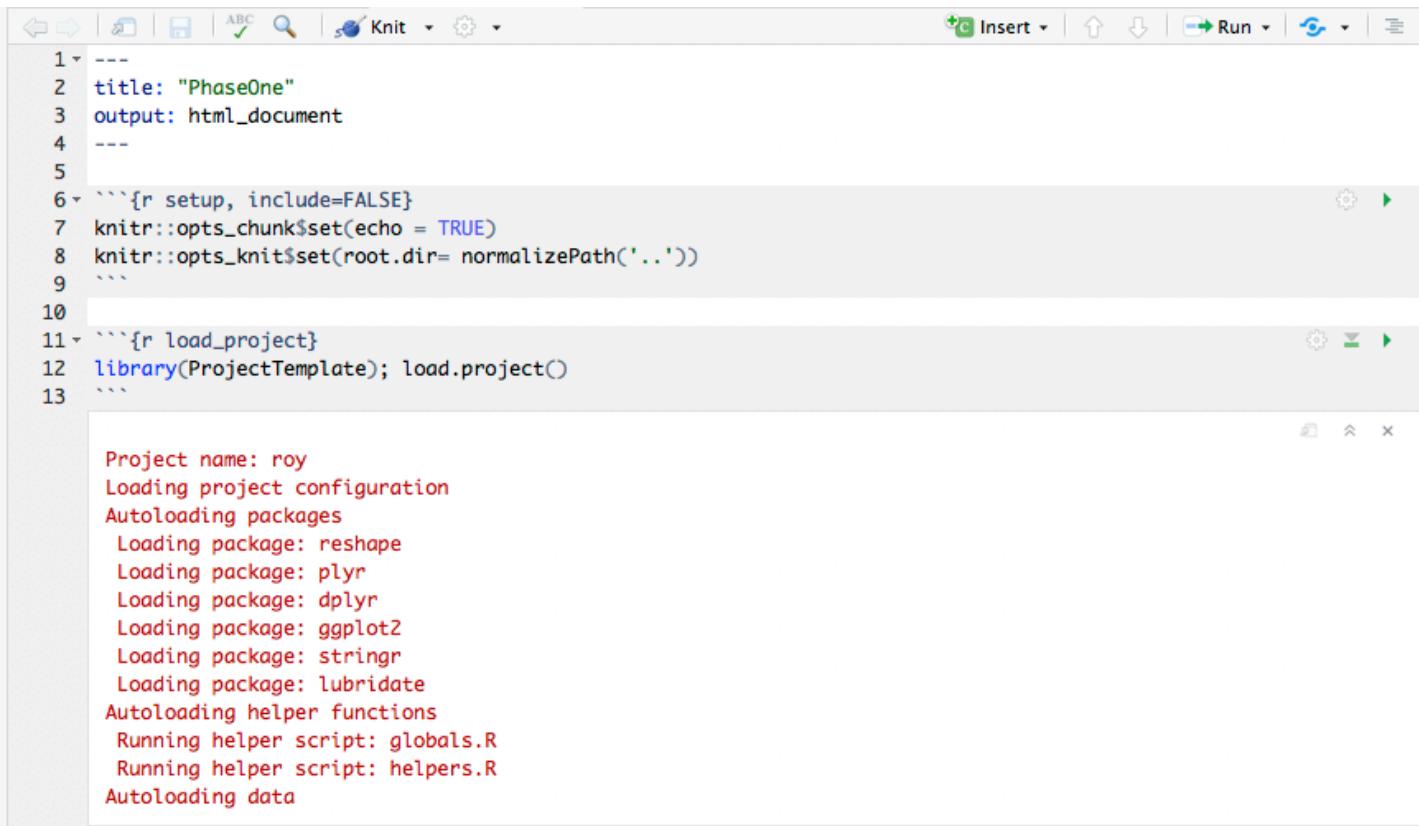
Loading a Project

```
library('ProjectTemplate')
setwd('~/projects/new_project')
load.project()
> library('ProjectTemplate')
Loading required package: testthat
> load.project()
Loading project configuration
Autoloading utility functions
Autoloading data
  Loading data set: letters
Munging data
  Running preprocessing script: 01-A.R
> █
```

Hints and Tips

- Append your ‘munge’ script names with a number, so they run in a predictable order.
- Do not edit your raw data!
- Specify all dependencies within config/global.dcf

```
knitr::opts_knit$set(root.dir= normalizePath('..'))
```



The screenshot shows the RStudio interface. The top bar includes standard icons for file operations, search, and a 'Knit' button. Below the bar is a code editor pane containing R code. The code starts with a YAML header block, followed by several code chunks. The first code chunk (line 6) contains an R command to set the root directory. The second code chunk (line 11) loads a project named 'ProjectTemplate'. The terminal pane at the bottom displays the output of the R code, which includes the project loading process and the auto-loading of various packages and helper functions.

```
1 ---  
2 title: "PhaseOne"  
3 output: html_document  
4 ---  
5  
6 `r setup, include=FALSE}  
7 knitr::opts_chunk$set(echo = TRUE)  
8 knitr::opts_knit$set(root.dir= normalizePath('..'))  
9 ```  
10  
11 `r load_project}  
12 library(ProjectTemplate); load.project()  
13 ```  
  
Project name: roy  
Loading project configuration  
Autoloading packages  
Loading package: reshape  
Loading package: plyr  
Loading package: dplyr  
Loading package: ggplot2  
Loading package: stringr  
Loading package: lubridate  
Autoloading helper functions  
Running helper script: globals.R  
Running helper script: helpers.R  
Autoloading data
```

▶ **examples**
▶ **flow**
▶ **packages**
▶ **scripts**
▶ **src**
▶ **test**
▶ **types**

.babelrc.js

.editorconfig

.eslintignore

.eslintrc.js

.flowconfig

.gitignore

BACKERS.md

LICENSE

▶ **build: build 2.6.0-beta.2** **v2.6.0-beta.2 build: release 2.6.0-beta.2**

build: build 2.6.0-beta.2

feat: dynamic directive arguments for v-on, v-bind and custom directives (#9370)

feat: dynamic args for custom directives (#9370)

perf: improve scoped slots change detection accuracy (#9371)

test: test cases for v-on/v-bind dynamic arguments

refactor: v-bind dynamic arguments use bind helper

test: fix tests, resolve helper conflict

fix: fix middle modifier

feat: handle dynamic argument for v-bind.sync

feat: origin/slot-optimization

perf: improve scoped slots change detection accuracy (#9371)

feat: dynamic directive arguments for v-bind and v-on (#9370)

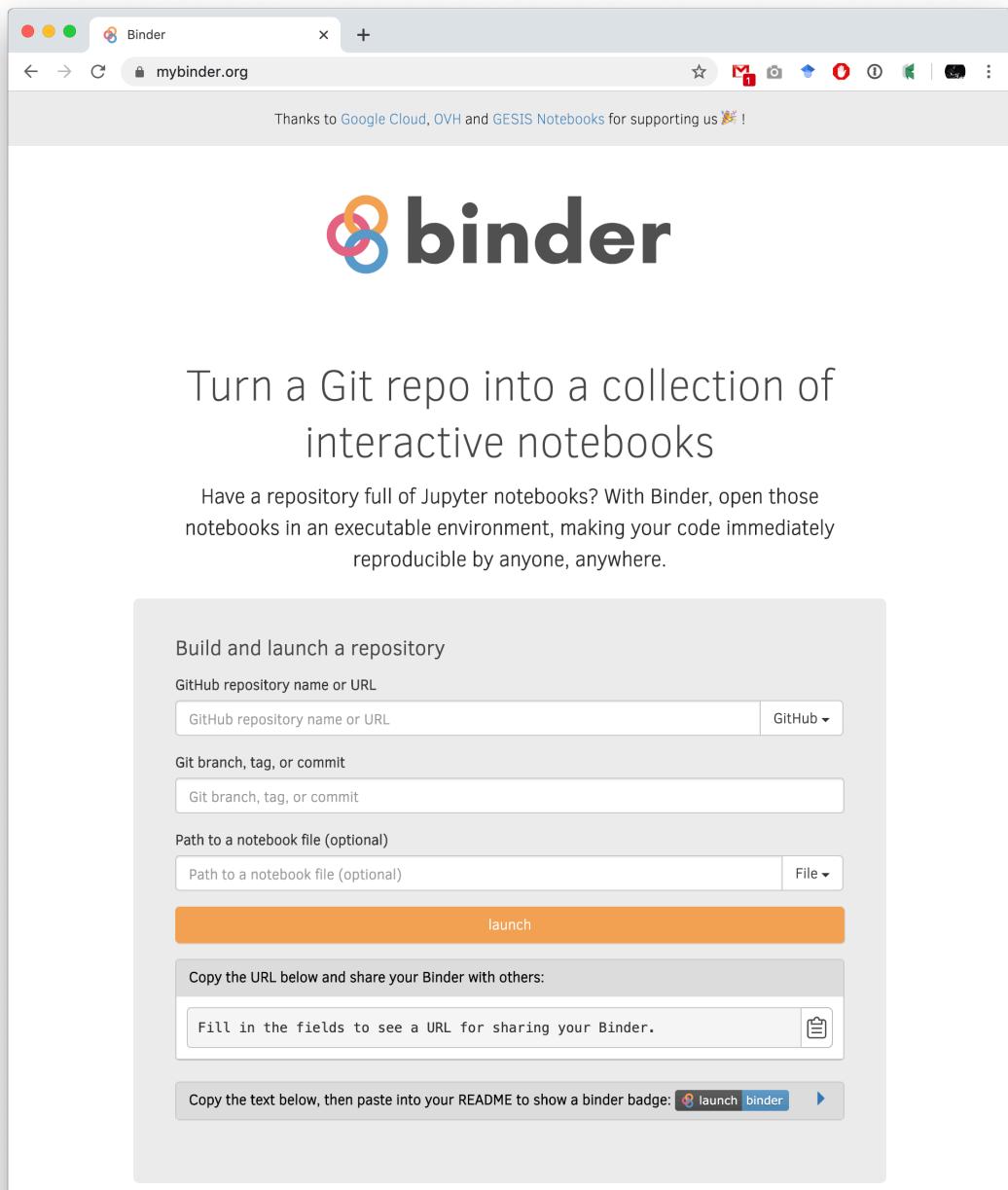
refactor: extend dom-props update skip to more all keys except value

fix: fix checkbox event edge case in Firefox

test: fix tests in IE/Edge

refactor: simplify timestamp parsing

above: update commit message



The Alan Turing Institute

Home + Research + Research projects

'The Turing Way' - A handbook for reproducible data science

Developing a handbook for best practice data science

Learn more ↓

T 'The Turing Way' - A handbook

https://www.turing.ac.uk/research/research-projects/turing-way-handbook-reproducible-data-science

The Turing Way

- 1. Introduction
- 2. Reproducibility
- 3. Open Research
- 4. Version Control
- 5. Collaborating on GitHub/GitLab
- 6. Credit for reproducible research
- 7. Research Data Management
- 8. Reproducible Environments
- 9. Testing
- 10. Reviewing
- 11. Continuous Integration
- 12. Reproducible Research with Make
- 13. Risk Assessment
- 14. BinderHub
- 15. Glossary

Powered by Jupyter Book

Introduction

https://the-turing-way.netlify.com/introduction/introduction.html

Welcome to the Turing Way

The Turing Way is a lightly opinionated guide to reproducible data science.

Our goal is to provide all the information that researchers need at the start of their projects to ensure that they are easy to reproduce at the end.

This also means making sure PhD students, postdocs, PIs, and funding teams know which parts of the "responsibility of reproducibility" they can affect, and what they should do to nudge data science to being more efficient, effective, and understandable.

A bit more background

Reproducible research is necessary to ensure that scientific work can be trusted. Funders and publishers are beginning to require that publications include access to the underlying data and the analysis code. The goal is to ensure that all results can be independently verified and built upon in future work. This is sometimes easier said than done. Sharing these research outputs means understanding data management, library sciences, software development, and continuous integration techniques: skills that are not widely taught or expected of academic researchers and data scientists.

The Turing Way is a handbook to support students, their supervisors, funders, and journal editors in ensuring that reproducible data science is "too easy not to do". It will include training material on version control, analysis testing, open and transparent communication with future users, and build on Turing Institute case studies and workshops. This project is openly developed and any and all questions, comments and recommendations are welcome at our GitHub repository: <https://github.com/alan-turing-institute/the-turing-way>.

The book itself

The book that you are reading is a [jupyter book](#). Jupyter books render markdown documents and jupyter notebooks as static html web pages. They are easy to read and navigate...but also easy to edit and extend!

Under construction

Watch this space for a little more information on how to contribute to the Turing Way!

Under construction

ON THIS PAGE

A BIT MORE BACKGROUND

THE BOOK ITSELF

THE TURING WAY COMMUNITY



JD Long

@CMastication

Reproducible actuarial workflows with **#rstats** ... I'm glad to see actuarial science adopting some good data science workflow methods.

<https://philipdarke.com/reproducible-actuarial-work/>

10:21 AM · Mar 2, 2019 · Twitter for iPad

15 Retweets 64 Likes

Reproducible data science techniques in actuarial work

What can actuaries learn from open science and other professions?

How can actuaries ensure that workflows are efficient, minimise the risk of errors and allow complex work to be reproduced by others in their organisation?

Actuaries exploring the use of data science have the opportunity to revisit existing ways of working and consider whether they remain appropriate. These challenges are also being faced in science and by other professions.

This presentation looks at why the concept of *reproducible work* is key and how it can help address the challenges of working in data intensive fields.



Institute
and Faculty
of Actuaries

Reproducible data science techniques in actuarial work

What can actuaries learn from open science and other professions?

O'REILLY®

R Cookbook

Proven Recipes for Data Analysis,
Statistics & Graphics



J.D. Long &
Paul Teator

Second
Edition

“ Reproducibility is like brushing your teeth. It is good for you, but it takes time and effort. Once you learn it, it becomes a habit. ”

Irakli Loladze, mathematical biologist

