Pensieve

Maximilian Held

2017-08-11

Contents

About

This is a very early draft, and is not ready for prime-time, let alone citation.

This also serves as the *documentation* of the `pensieve` package (in lieu of vignettes). To find out more, go to http://pensieve.maxheld.de.

Chapter 1

Introduction

`pensieve` is an open-source extension to the R project for statistical computing, distributed via the Comprehensive R Archive Network (CRAN) as one of thousands of packages.

It is accompanied by `accio`, a web-based, closed-source front-end, for which `pensieve` serves as the backend. If you do not want to use hosted or closed-source software, not to worry: `accio` is meant as a convenient way of accessing `pensieve`, for users who may not want to install and learn R. Almost all features accessible in `accio` are also available in `pensieve`, with the exception (for now) of deploying a web-based Q-Sort.

Though formally a package like many others, `pensieve` is in some ways *less and more* than mainstream packages that you may be familiar with and on which it is largely based. For now, at least, `pensieve` offers little advanced algorithms by itself, but merely wrangles data and interface other, powerful packages. This makes it somewhat less than a "real" package, and more of a wrapping layer. At the same time, `pensieve` is vastly more specialized than many other packages, tailored to the needs of *one* domain: the scientific study of human subjectivity. This makes it akin to a Domain Specific Language or a mini-language.

`pensieve`, is, in other words, a very nich enterprise, much like Q Methodology itself: endowed with great hopes by its creator, and perhaps some yet-unrealised potential.

1.1 The OO System

One of the dirty secrets of quantitative work is that a good 80% of the time is often spent on "cleaning and preparing" messy data [Dasu and Johnson (2003); as cited in @Wickham-2014: 1]. This tedium affects the quantitative stages of Q as much

as other analyses, as is best addressed head-on and as early as possible, to avoid downstream complications.

To that end, `pensieve` prescribes a large number of standardized data formats, in which otherwise messy data can be tidily stored.[1] These standardized data formats come in the form of S3 classes, which is the simplest (and oldest) system for object-oriented (OO) programming in R. There is no need to know anything about S3 or OO in general to use and benefit from this system in `pensieve`: Suffice it to say that S3 is a clever way for data to *know* what *kind* of data *class* they are, and what kind of *methods* can be applied to them. For example, `QLoadings` "know" that they are loadings, and that they can be `rotate()`d.

It is easy to recognize theses classes in `pensieve`: they are all in CamelCase and start with a capital Q, as, for instance `QSorts`. They all come with a *constructor* function, always by the same name (`QSorts()`), which allows you to create such an object from appropriate inputs. These constructor functions do not do very much; they simply validate the data and assign the class.

Each class (such as `QSorts`) is also accompanied by a method for the `check()` generic, which means that you can, well, `check()` any object to make sure that it conforms to the standardized data format. For example, `check()`ing a `QLoadings` object will assert that all values are between `-1` and `1`, as loadings must be by definition, as well as a number of other criteria. To find out about these criteria for valid data, consult the documentation for each of the classes, by entering, say `help("QSorts")`. The arguments section will list all of the criteria that are also tested. If your data is partly invalid, the `check()`s will give you (hopefully) precise and instructive error messages for you to fix.

These `check()`s are run on whenever `pensieve` touches *any* of its known classes, which ensures that data are always in their proper form. This is a simple way to shoehorn type validation into S3, which ordinarily does not have such a facility.

In addition to the `check()` functions (which return `TRUE` or the error message), you can also use `assert()` (which only returns a message in case of an error), `test()` (which only returns `TRUE` or `FALSE`) or `expect()` (for internal usage in testing via Wickham (2011)'s `testthat`). This follows the framework set by Lang (2017)'s `checkmate`, which `pensieve` extends.

If you only use the functions provided by `pensieve`, you will probably never need `check()` or any of its siblings: `pensieve` will do all necessary checking for you. If, however, you add some custom code of your own, you may want to throw in an occasional `assert()`ion into your script to ensure that everything is still kosher.

[1] Wickham (2014a) has suggested a formal definition of *tidy* datasets where each row is an observation, each column a variable and each table an observational unit. While much of `pensieve` follows this philosophy, not all Q objects are best represented as tidy `data.frame`s or `tibble`s. Where rows and columns can be meaningfully transposed or where linear algebra operations are readily applicable, as is the case for Q-sorts and downstream results, data is instead stored in matrices and higher dimensional arrays. This is in line with recent recommendations for such non-tidy data by (Leek 2016).

Aside from `check()`s, there are some other methods defined for common generics, defined for some of the objects, such as customized `print()`ing or `plot()`ing functions. You can find out about all these special "abilities" of the classes by reading their help.

A word of warning: you *can* always assign or remove classes *by hand*, rather than using the provided constructor functions and their included `check()`s, but I strongly recommend against this. Because S3 is a relatively simple and ad-hoc system, it is easy to "shoot yourself in the foot" (Wickham 2014b). Downstream `pensieve` functions may fail, or worse, produce wrong results if data is not provided in the precise format specified in the included classes.

This design may strike users as involved, or overly restrictive, but there is little need to worry. Most average users of `pensieve` will never be exposed to OO system "behind the scenes". More advanced users with custom extensions should also not feel stifled; rather than restricting the extensibility of R, this validation infrastructure merely provides a well-defined interface *for* extending `pensieve`. With such a system in place, all users can rest assured that all reasonable precautions are taken to ensure the reliability and reproducibility of their results, as would be expected from any *scientific* study of human subjectivity.

1.2 The `QStudy` List

In addition to *atomic* (or vector) data formats, such as `QSort` which cannot be divided up further, `pensieve` also provides a lot of *composite* (or list) data formats, which comprise of atomic classes, or other list classes. This is easy to illustrate with `QItems`, which must always (at least) consist of a `QItemConcourse` and a `QItemSample`.

Higher-level classes such as `QItems` also come with their constructor (`QItems()`) and `check()` functions, but these expect the lower level, atomic classes as inputs. Instead of checking *their* validity, they check the *consistency* between them. For example, we know that `QItemSample` may only contain items for which there is also an entry in `QItemConcourse`; it would not make sense to have an item that is part of the sample, but not the concourse.

This structure of nested classes goes all the way up to a complete `QStudy` object, which includes *all* data relevant for, and produced in any given study.

The top-level list `QStudy` looks like this:

- `QStudy`
- `QItems`
- `QDesign`
- `QPeople`
- `QData`

- …

The order of this list approximately follows the progress of most Q studies, and is also reflected in the organisation of this book. Each of the high-level list objects, including their elements, functions and substantive considerations is covered in the following chapters.

However, there is no technical reason to follow this order or structure of classes; users can organise their data in whichever form they prefer. Some functions accept higher-level, nested lists, but they always also accept the atomic form.

In addition, not *all* Q studies will have use for all of these objects, or their elements: the nested data structures offered in `pensieve` are, for the most part, strictly optional. It is, in fact, possible to run `pensieve` with only a minimum of these objects, emphasized in the above.

Part I

Design

Chapter 2

Items

Good, evocative items lie at the heart of a Q study. In fact, akin to the old computer science adage of *"garbage-in / garbage-out"*, the quality of the *items* strictly limits the insights that may be gleaned from the entire study.

Generating and selecting good items is at least half science, half tradecraft. There are some standards and frameworks that can structure your item generation and sampling, but – for now at least – it still requires a lot of experience, intuition and knowledge of the domain in question.

`pensieve` supports this stage of a Q study in a number of ways:

1. Items **generation and selection** can be fully documented inside of `pensieve`, making the process transparent, reproducible and easy to iterate over.
2. Items can be **professionally typeset**, and stored in `pensieve`, so that Q sorters and researchers alike both see the items in the *same, definitive form*, be they text or images.
3. Arbitrary **additional information** can be stored alongside the items for later "R-way" analysis.

Items and related information are stored in a specific, domain-specific format in `pensieve`, a list of S3 class `QItems`. The `QItems` list *can* be quite extensive, covering all *possible* use cases. But there is no need to worry, if your study is simpler: most of its elements are optional, and it is easy to get your items in and out of it.

- `QItems`
- `QItemConcourse` (char or bin matrix)
- `QItemFeatures` (tibble)
- `QItemStrata` (array)

- `QItemSample` (char vector)
- `QItemFormat` (list)
- `QItemSet` (list)

References

References

Dasu, Tamraparni, and Theodore Johnson. 2003. *Exploratory Data Mining and Data Cleaning*. 1 edition. New York: John Wiley & Sons.

Lang, Michel. 2017. "Checkmate: Fast Argument Checks for Defensive R Programming." *The R Journal* 9 (1): 437–45. https://journal.r-project.org/archive/2017/RJ-2017-028/index.html.

Leek, Jeff. 2016. "Non-Tidy Data." Blog. *Simplystats*. February 17. https://simplystatistics.org/2016/02/17/non-tidy-data/.

Wickham, Hadley. 2011. "Testthat: Get Started with Testing." *The R Journal* 3 (1): 5–10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

———. 2014a. "Tidy Data." *Tidy Data* 59 (10).

———. 2014b. *Advanced R*. 1 edition. CRC. Boca Raton, FL: Chapman and Hall/CRC.