Stashing and playing with raw data locally from the web

It is getting easier to get data directly into R from the web. Often R packages that retrieve data from the web return useful R data structures like a data.frame or plot. This is a good thing of course to make things user friendly.

However, what if you want to drill down into the data that's returned from a query to a database in R? What if you want to get that nice data.frame in R, but you think you may want to look at the raw data later? The raw data from web queries are often JSON or XML data. This type of data, especially JSON, can be easily stored in schemaless so-called NoSQL databases, and queried later.

So here's the use case, or workflow:

- Query a database on the web, and choose to write the raw data to a local database.
- Do whatever you want with the output R object - analyze, visualize, etc.
- Now you want to go back and search through some of the raw data. But, that query took an hour. Since you wrote it to a local database, you can search the data.
- If you hadn't written it locally, you would have to make a new web call.

I've started an R package to interact with the NoSQL database CouchDB. This is a schemaless database that speaks JSON, so you can store JSON and get back JSON (no worries about XML, we can just wrap it in JSON). What's especially cool is you can interact with CouchDB via a RESTful API. CouchDB doesn't have full text search built in (though you can build map-reduce *Views*, basically preset queries on the database), so I added functions (and docs to help) to interact with the CouchDB River plugin for Elasticsearch, which provides powerful full text search via an API interface. But nevermind the tech details - all this just means you can search on the full text of your stored data.

There are plenty of databases you can interact with from R, so why CouchDB? For one, it makes a lot of sense to write to a NoSQL database since this blog post is dealing with a use case writing JSON, which isn't a good fit for databases like MySQL, SQLite, PostgreSQL, etc. (though postgres allows you to write JSON). It didn't have to be CouchDB, but at least to me it seems relatively easy to install, you can interact with it via an HTTP API (if you're into that, which I am), and it has a nice web interface (navigate to http://localhost:5984/_utils/ after starting `couchdb`).

Is this for the casual R user? Probably not. But, I imagine there are R users out there that want some more flexibility when it comes to interacting with web data in R. It is nice and tidy to get back an R data.frame from a web call, but having the raw data at your fingertips could be super powerful. I'll describe using an R package to interact with a web database with `sofa` baked in, and discuss a bit about the functions within `sofa`.

First start CouchDB in your terminal

You can do this from anywhere in your directory. See here for instructions on how to install CouchDB.

```
couchdb
```

Then start elasticsearch in your terminal

See here for instructions on how to install Elasticsearch and the River CouchDB plugin.

```
cd /usr/local/elasticsearch
bin/elasticsearch -f
```

Install sofa

```
# Uncomment these lines if you don't have these packages installed
# install.packages('devtools') library(devtools) install_github('sofa',
# 'ropensci') install_github('alm', 'ropensci', ref='couch')
library(sofa)
library(alm)
```

Simultaneously write data to CouchDB along with API calls using the alm package to get altmetrics data on PLoS papers. Ping to make sure CouchDB is on

```
sofa_ping()
```

```
  couchdb   version
"Welcome"   "1.2.1"
```

Create a new database

```
sofa_createdb(dbname = "alm_db")
```

```
  ok
TRUE
```

Write couchdb database name to options

```
options(couch_db_name = "alm_db")
```

List the databases

```
sofa_listdbs()
```

```
 [1] "_replicator"                 "_users"
 [3] "alm_couchdb"                 "alm_db"
 [5] "dudedb"                      "example"
 [7] "poop"                        "rplos_db"
 [9] "shit"                        "shitty"
[11] "shitty2"                     "test_suite_db"
[13] "test_suite_db/with_slashes" "test_suite_reports"
[15] "testr2couch"                 "twitter_db"
```

Search for altmetrics normally, w/o writing to a database

```
head(alm(doi = "10.1371/journal.pone.0029797"))
```

```
          .id pdf html shares groups comments likes citations total
1   bloglines  NA   NA     NA     NA       NA    NA         0     0
2   citeulike  NA   NA      1     NA       NA    NA        NA     1
3    connotea  NA   NA     NA     NA       NA    NA         0     0
4    crossref  NA   NA     NA     NA       NA    NA         4     4
5      nature  NA   NA     NA     NA       NA    NA         4     4
6 postgenomic  NA   NA     NA     NA       NA    NA         0     0
```

Search for altmetrics normally, while writing to a database

```
head(alm(doi = "10.1371/journal.pone.0029797", write2couch = TRUE))
```

```
          .id pdf html shares groups comments likes citations total
1   bloglines  NA   NA     NA     NA       NA    NA         0     0
2   citeulike  NA   NA      1     NA       NA    NA        NA     1
3    connotea  NA   NA     NA     NA       NA    NA         0     0
4    crossref  NA   NA     NA     NA       NA    NA         4     4
5      nature  NA   NA     NA     NA       NA    NA         4     4
6 postgenomic  NA   NA     NA     NA       NA    NA         0     0
```

Make lots of calls, and write them simultaneously

```
# install_github('rplos', 'ropensci')
library(rplos)
dois <- searchplos(terms = "evolution", fields = "id", limit = 100)
out <- alm(doi = as.character(dois[, 1]), write2couch = TRUE)
lapply(out[1:2], head)
```

```
list()
```

Writing data to CouchDB does take a bit longer

```
system.time(alm(doi = as.character(dois[, 1])[1:60], write2couch = FALSE))
```

```
   user  system elapsed
  2.168   0.022   5.844
```

```
system.time(alm(doi = as.character(dois[, 1])[1:60], write2couch = TRUE))
```

```
   user  system elapsed
  0.034   0.007   0.938
```

Search using elasticsearch

tell elasticsearch to start indexing your database

```
elastic_start(dbname = "alm_db")
```

```
$ok
[1] TRUE
```

Search your database

```
out <- elastic_search(dbname = "alm_db", q = "twitter", parse_ = TRUE)
out$hits$total
```

```
[1] 678
```

Using views

Write a view - here letting key be the default of null

```
sofa_view_put(dbname = "alm_db", design_name = "myview", value = "doc.baseurl")
```

```
$ok
[1] TRUE

$id
[1] "_design/myview"

$rev
[1] "1-e7c17cff1b96e4595c3781da53e16ad8"
```

Get info on your new view

```
sofa_view_get(dbname = "alm_db", design_name = "myview")
```

```
$`_id`
[1] "_design/myview"

$`_rev`
[1] "1-e7c17cff1b96e4595c3781da53e16ad8"

$views
$views$foo
                                    map
"function(doc){emit(null,doc.baseurl)}"
```

Get data using a view

```
out <- sofa_view_search(dbname = "alm_db", design_name = "myview")
length(out$rows)  # 160 results
```

```
[1] 1
```

```
sapply(out$rows, function(x) x$value)[1:5]  # the values, just the API call URLs
```

```
[1] "http://alm.plos.org/api/v3/articles"
[2] NA
[3] NA
[4] NA
[5] NA
```

Delete the view

```
sofa_view_del(dbname = "alm_db", design_name = "myview")
```

```
[1] ""
```

What happens now?

Well, if no one uses this, then probably nothing. Though, if people think this could be useful, then...

- It would be cool to make easy hooks into any package making web calls to allow users to write data to CouchDB if they choose to, sort of like the example above with rplos.
- Perhaps automate some of the setup for CouchDB for users, making system calls so they don't have to.
- ???