Lifehacking dla **R**

Przemyślenia i rozwiązania w temacie reprodukowalności analiz i organizacji pracy

Michał Bojanowski
m.bojanowski@uw.edu.pl
www.bojanorama.pl

ICM, Uniwersytet Warszawski

Spotkania Entuzjastów R
27 luty, 2014

Lifehacking dla **R**

Wikipedia głosi

Lifehack [ang. *life* "życie", *hack* orig. "ciąć", też "kombinować"] to ogólna nazwa sposobów i trików używanych do ułatwiania sobie życia.

Cel:

- Jak ułatwić sobie życie pracując z **R**?
  - ...w pewnych sytuacjach...

1 Kontekst i problem

2 Potrzeby

3 Istniejące rozwiązania

4 Alternatywne rozwiązanie

5 Podsumowanie

Kontekst

- Mamy różne źródła danych (pliki, bazy danych, *web scrapping*)
- Efektem końcowym ma być raport (tekst lub prezentacja)
- Analizy średniej skali
  - Bardziej złożona niż jeden skrypt
  - Być może nie "Big Data"
- *In statu nascendi*
  - Względnie dużo czyszczenia i sprawdzania spójności danych
  - Eksperymentowanie z analizami i wizualizacjami (EDA)

Problem/zadanie: reprodukowalne analizy

Reprodukowalne analizy (*reproducible research*), co to jest i po co?

- Reprodukowalne przejście od danych surowych do raportu
- Wykonywane bez konieczności ręcznych interwencji
- Umożliwiają generowanie raportu nawet gdy zmieniają się kod, dane lub założenia
- Ręczne przeklejanie wyników do raportu pracochłonne i błędorodne.
- Dokumentacja procesu analitycznego
- "*Trustworthy software*" (Chambers 2008). Analizowanie danych to dziś pisanie software'u.

Pożądane własności rozwiązania

Wersjonowanie plików Śledzenie zmian.

Nieinteraktywne wykonywanie skryptów Wygenerowanie raportu nie powinno wymagać za każdym razem jakiś ręcznych interwencji.

Caching rezultatów Nie chcemy wykonywać analiz, dla których kod źródłowy i dane się nie zmieniły.

Dokumentowanie analiz Dobrze by było trzymać dokumentację i kod razem.

Obliczenia równoległe Szybciej!

Prostota Nie potrzebujemy dodatkowego "systemu" do zarządzania i debugowania.

Istniejące rozwiązania

Wersjonowanie plików Git, Subversion, etc.

Nieinteraktywnie Tryb wsadowy: `R CMD BATCH`, `Rscript`, `r`

Dokumentowanie analiz *Literate programming* m.in. Sweave, knitr (`knitr::spin`!), brew

Caching rezultatów cacheSweave, stashR, knitr

Pakiet projectTemplate organizuje cały projekt w foldery (dane, skrypty, logi, etc.). Raczej sztywna konstrukcja. Klon GNU `make`?

A może GNU `make` po prostu?

GNU Make

Co to jest GNU `make`[1]:

- Narzędzie do kompilowania programów
- Automatycznie określa, które fragmenty większego programu powinny być skompilowane jeszcze raz
- Wymaga opisania zależności pomiędzy plikami (źródła i efekty kompilacji) oraz komand niezbędnych do kompilacji. Plik `Makefile`.
- Może wykonywać operacje równolegle (`make -j`).
- Można używać w innych kontekstach, w których np. przetwarzamy jakąś kolekcję plików źródłowych (dane, skrypty, etc.)

[1] www.gnu.org/software/make

Zawartość Makefile

`Makefile` przede wszystkim zawiera reguły postaci

```
target: prerequisite1 prerequistie2 ..
    recipe
```

Przykład dla **R**:

```
wykres.pdf: skrypt.R dane1.dat dane2.dat
    R CMD BATCH skrypt.R
```

"Nieoczywistości"

- Podzielenie analiz na moduły (podkatalogi)?
  - `make` może działać "rekursywnie", ale jest to nieefektywne (Miller 1997) pod względem obliczeń równoległych (`make -j`).
- Jednym skryptem **R** często generujemy wiele innych plików (np, wykresów)
  - `make` zakłada, że jednym "przepisem" generujemy jeden plik wynikowy ("target"). Bynajmniej sprawy nie załatwia reguła typu:

    ```
    target1 target2: skrypt dane
    ```

GNU make z R

Jak użyć GNU `make` efektywnie z **R**?

Istniejące rozwiązania/szablony

- Rob J. Hyndman (autor m.in. forecast)[2]
  - Bardzo proste
  - Nie śledzi wszystkich targetów
  - Bazuje na plikach `.Rout` – `make` może się nie zatrzymać w przypadku błędów w skryptach **R**.
- Pierre Lindenbaum[3]
  - **R** jako główna powłoka `make`
  - Wymaga pisania kodu **R** wewnątrz `Makefile` – niezbyt wygodne.
  - Raczej "proof of concept".

[2] http://robjhyndman.com/hyndsight/makefiles/

[3] http://plindenbaum.blogspot.com/2014/01/parallelizing-rstats-using-make.html

Triki GNU `make` z **R**: modułowość bez rekursywności

Podkatalogi zawierają `module.mk` czytany przez `Makefile`

```
.
|
+- Makefile
+- dir1
|   +- module.mk
|   +- skrypt1.R
|   +- dane.dat
+- dir2
    +- module.mk
```

Triki GNU make z R: modułowość bez rekursywności

```
### Makefile

# Puste!
FILES:=

include dir1/module.mk
include dir2/module.mk

.PHONY: default
default: $(FILES)
```

```
### dir1/module.mk

# katalog modulu
local_dir:=dir1

# lokalne targety
local_files:=wykres1.pdf wykres2.pdf

# tutaj pełne ścieżki wzg Makefile
dir1/wykres1.pdf: dir1/skrypt1.R
    (cd dir1 && RCMD BATCH skrypt1.R)

# dopisanie tgt do FILES
FILES+=$(addprefix $(local_dir)/,
                   $(local_files))
```

Triki GNU `make` z **R**: wiele targetów

Przykładowo w `module.mk` (nazwy plików powinny być ze ścieżkami relatywnie do `Makefile`):

```
skrypt.res: skrypt.R dane.dat
    R CMD BATCH skrypt.R &&   # generuje wykresy
    tar cf wykres.pdf

wykres1.pdf wykres2.pdf: skrypt.res
    tar xf skrypt.res -m
```

Podsumowanie

- **R** + knitr + GNU `make` jest super!
- Zapewnienie poprawnego działania równoległego (bez rekursywności) wymaga pilnowania ścieżek w regułach i przepisach.

Pomysły:

- Odczytywanie targetów ze skryptu (np. składnia Roxygen?)

Szersza wersja tej prezentacji niebawem na blogu "Brokering Closure" bc.bojanorama.pl!