Programering med R (2022-10-12)

Table of Contents

Inledning

För mindre enklare frågeställningar, särskilt om det är frågeställningar som bara kommer upp en gång, är det ofta enklare att skriva kommandon direkt i RStudio (eller motsvarande). Men för större projekt eller om det är något som återkommer regelbundet (t.ex. en rapport till en chef eller ledningsgrupp) blir det mycket enklare att skriva ett skript. För ett större projekt är det en fördel att ha ett skript även om det är en engångsföreteelse:

- Man kan tänka igenom sin logik och därmed undvika misstag
- Man kan återanvända delar i andra projekt.
- Det är en dokumentation på vad man gjorde, om man själv eller någon annan vill veta vid en senare tidpunkt.

Av dessa skäl skriver jag personligen nästan alltid skript även om de är korta och egentligen hade gått lika snabbt att utföra direkt vid prompten. Ibland när jag arbetat på kommandoraden, men vill kunna återskapa det jag gjort utnyttjar jag att allt jag gjorde finns kvar i .Rhistory: när jag är klar kopierar jag .Rhistory och redigerar fram ett enkelt skript.

Spara resultat från ett skript

Jag börjar i slutet: hur skall man spara sitt resultat?

- Låta det gå ut som text/tabell i kommandofönstret. Detta fungerar bra om man t.ex. skall rapportera in några siffror till Folkhälsomyndigheten varje vecka/månad.
- Spara en CSV Eller Excel-fil
  - CSV: universalformat, ingen risk att metadata som kan avslöja mer om en patient än man tänkt sig kommer med
  - Excel: det bekanta formatet, som "alla" kan öppna och är vana vid att hantera.
- Graf eller multiplot. Du kan skapa en plot och sedan inkorporera den i t.ex. ett Centuri-dokument, PowerPoint, poster, vetenskaplig artikel, etc.
- En färdig rapport med Rmarkdown. Du kan skapa ett Rmarkdown-dokument som inkluderar R-kod som exekveras och där resultatet syns som t.ex. en tabell eller graf i dokumentet. Från Rmarkdown kan du bland annat skapa Word-dokument, men även t.ex. HTML.

Allmänt om programmering

Detta är inte en heltäckande programmeringskurs, utan är bara en introduktion. Men några tips bör vi ha med oss i bagaget.

- Kommentera din kod! Skriv tydliga kommentarer som förklarar vad du gör i alla delarna i ditt skript. Börja en rad med ett "#", och R kommer inte att utföra eventuella kommandon på den raden
- Döp dina variabler efter ett mönster. Om det är "första datum som skall tas med" kalla det t.ex. "Datum_första_inkluderade" eller "DatumFörstaInkluderade". Inte datum1.
- Ha en ordentlig struktur på sitt skript. Själv följer jag nästan alltid mönstret
  1. libraries()
  2. Skapa namn från t.ex. dagens datum
  3. Ladda in data-filer
  4. Städa
  5. Utför analys, etc
  6. Spara ev filer
- Skriv en lite utförligare kommentar i början där du förklarar syftet med hela skriptet, Ange vem som skrivit det, och när. Lägg till under-sektioner som TODO (vad saknas) och alla uppdateringar.

R-skript – viktigt att veta

1. De har filändelsen .R
2. De är helt enkelt en serie R-kommandon, som utförs i ett svep istället för ett och ett.
3. Man kör ett skript med kommadot `source("SkriptetsNamn.R", encoding="UTF-8")`. Som vanligt: om du inte använder t.ex. ääö någonstans kan du ofta klara dit utan *encoding = "UTF-8"*: jag har i princip alltid med det eftersom jag ofta skriver på svenska, eller mina data innehåller kolumner med namn som "ÖSJ".
4. Kommentarer: allt från ett "#" till slutet av raden räknas som en kommentar, och tolkas inte av R. Detta är användbart på två sätt
   - För att skriva kommentarer om vad du vill åstadkomma just här i skriptet.
   - För att tillfälligt(?) låta bli att utföra en del av ditt skript. Till exempel för att du vill testa ett alterantiv men ha kvar möjligheten att återgå, för att hoppa över vissa delar för att testa snabbare, osv.
5. Om du skall göra större förändringar i ett skript: spara en backup-kopia. Själv försöker jag hålla mönstret att jag döper dem
   - Skript.R
   - Skript_20221011.R
   - Skript_20221011T0654.R (formatet är en hemsk ISO-standard)

När jag arbetar i en Unix-miljö har jag ett skript som gör detta automatisk, jag borde skriva en version av det i PowerShell och som ett R-kommando: Windows filhanterares "fil - kopia(42).txt" är inte lika användbart.

Ladda in dina data och ta ut en del av dem

Detta gick vi igenom förra gången, men en kort repetiton kan vara användbar.

```
acov <- read.csv2("acov_GDPR.csv", fileEncoding = "UTF-8-BOM")
acov$Ålder <- as.numeric(gsub("," , ".",  acov$Ålder..År.), na.rm=TRUE)

acov.ÖSJ <- acov[ acov$Beställare == "ÖSJ", ]
acov.65plus <- filter(acov, Ålder > 65)
```

If-satser

Ibland vill du att R skall agera olika beroende på något i dina data.

```
if (Sys.Date() > "2022-10-01"){
      print("Efter")
} else {
      print("Före")
}
```

Ett tips: undvik om möjligt att ha långa kedjor av if-else inuti varandra: det blir oftast mycket svårt att följa. Bättre är att skriva mer specifika villkor (exempel: kön == "K" & Ålder > 65 & Ort == "Glesbygd")

Funktioner

När man skall utföra samma sak flera gånger (t.ex. rita en graf eller utföra en lite mer komplicerad beräkning) kan man skapa en funktion för det.

```
> DagarTillJul <- function(datum){as.Date("2022-12-24") - as.Date(datum)}
> DagarTillJul("2022-10-12")
Time difference of 73 days
>
> HurMångaDagar <- function(datum1, datum2){as.Date(datum1) -
as.Date(datum2)}
> HurMångaDagar("2022-10-10", "1969-07-20")
Time difference of 19440 days
```

Upprepa för varje värde: for-loopar

När du har ett antal olika grupper du vill utföra samma sak för (t.ex. skapa en graf) är det smidigt att använda for-loopar. Ett trivialt exempel är

```
> for (i in 1:5){
+ print(i)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Ett lite mer utvecklat exempel vore en sammanfattande rapport direkt på skärmen:

```
# ladda in datafilen, och snabbstäda
acov <- read.csv2("acov_GDPR.csv", fileEncoding = "UTF-8-BOM")
acov$Ålder <- as.numeric(gsub("," , ".",  acov$Ålder..År.), na.rm=TRUE)

## Warning: NAs introduced by coercion

for ( kundgrupp in unique(acov$Beställare)) {
print(
      paste0(
            "För ", kundgrupp, " var medianåldern ",
            median(filter(acov, Beställare == kundgrupp)$Ålder, na.rm=TRUE),
            " och i genomsnitt analysvärdet ",
            round(mean(filter(acov, Beställare == kundgrupp)$Mätvärde,
na.rm=TRUE ),1))
```

```
      )
}
```

```
## [1] "För ÖSJ var medianåldern 60.28 och i genomsnitt analysvärdet 5.2"
## [1] "För VC.GLESBYGD var medianåldern 52.3 och i genomsnitt analysvärdet
10.1"
## [1] "För VC.Östersund var medianåldern 51.41 och i genomsnitt analysvärdet
6.1"
## [1] "För NA var medianåldern NA och i genomsnitt analysvärdet NaN"
```

Upprepa tills ett villkor uppfylls: while-loopar

ibland vill du utföra något tills ett villkor är uppfyllt

```
> räknare <- 0
> while(räknare < 5) {
+   print(räknare)
+   räknare <- räknare + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
```

En varning: det är fullt möjligt att skapa en loop som aldrig tar slut:

```
> räknare <- 1
> while(räknare > 0) {
+   print(räknare)
+   räknare <- räknare + 1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
...
[1] 33698
>
```

(Jag avbröt genom att trycka på “Esc”, annars hade den fortsatt räkna mot oändligheten)

Resultat från ditt skript

Som jag nämnde ovan finns det flera alternativ. Du kan spara en CSV- eller Excel-fil, skapa en R Markdown-rapport eller bara visa resultatet på skärmen.

Spara en fil

Spara en CSV med `write.csv2(objekt , file="filnamn.csv")` eller med `readr::write_csv2(x=acov.ÖSJ, file = "foobar.csv", append = TRUE, progress = TRUE)`

Spara en excel-fil med `xlsx::write.xlsx2(acov.ÖSJ, file = "foobar.xlsx", sheetName = "ÖSJ", append=TRUE)`.

Ett exempel på en R markdown-rapport

````
---
title: "Ledtid, sammafattning"
output:
  #html_document:
   # toc: true
  word_document:
    #toc: true
  #md_document: default
---


Sammanfattning av svarstider för analyser på CobasPro under perioden
2021-01-01 -- 2022-09-30. Q90, Q95 samt Q99 är respektive kvartiler.
Alla tider är i minuter.



```{r echo = FALSE, results='asis'}

library(knitr)
knitr::kable(head(TAT.summary.cobas[,  c(1,2,3, 4,9,10,11)]), "pipe")
```

Mitt förslag är därför att vi anger förväntat svarstid enligt följande:

* C-analyser
`r ceiling(TAT.summary.cobas[1,9][[1]]/60) `h
* E-analyser
`r ceiling(TAT.summary.cobas[2,9][[1]]/60)`h.
* För virusanalyserna
föreslår jag istället `r ceiling(TAT.summary.cobas[3,9][[1]]/(24*60))` dygn.

Vi kan ange att vi historisk lämnat ut över 90% av svaren innom dessa
tidsgränser.
~
````

Ledtid, sammafattning

Sammanfattning av svarstider för analyser på CobasPro under perioden 2021-01-01 – 2022-09-30. Q90, Q95 samt Q99 är respektive kvartiler. Alla tider är i minuter.

Typ	Antal	Medel	Median	Q90	Q95	Q99
C-analys	1821738	130	61	225	312	1489
E-analys	260794	166	94	281	366	1304
E-Virus	2276	1035	340	2770	4188	8364

Mitt förslag är därför att vi anger förväntat svarstid enligt följande:

- C-analyser 4h
- E-analyser 5h.
- För virusanalyserna föreslår jag istället 2 dygn.

Vi kan ange att vi historisk lämnat ut över 90% av svaren innom dessa tidsgränser.

Visa på skärmen

Detta är det enklaste:

```
> tab1 <- ftable(acov$Beställare, acov$Kön)
> print(tab1)
                  K    M

VC.GLESBYGD    3201 1368
VC.Östersund   2881 1185
ÖSJ            5308 3993
```

Givetvis kan man göra tabellen mycker mer avancerad, eller skriva text med hjälp av `cat()` eller `paste0()`.

Debugging

Du kommer garanterat att stöta på buggar i din kod. Det finns inbyggda verktyg för att spåra vad som händer och varifrån felet uppkommer, men man kan även arbeta enklare, utan verktygen

Klipp och klistra

Det här är helt enkelt när du kör din kod direkt på kommandoraden, bit för bit. Du kan kolla att det blev rätt:

- Inga felmeddelanden? Bra
- Hände vad du förväntade dig med t.ex. en inladdad fil? Kanske tog du bort alla rader i stället för bara några?
- När allt funkar: fortsätt med nästa del i ditt skript

Lägg in debugg-kommentarer

Om du t.ex. undrar om en loop faktiskt gör vad du tror: lägg in en print-sats som talar om vad den jobbar med:

```
> räknare <- 0
> while (räknare < 12) {
+   cat("DEBUG: räknare =  ", räknare, "\n")
+   räknare <- räknare+1
+   print(Sys.Date() + räknare)
+ }
```

Om du undrar var ett fel uppstår kan du till och med bara lägga till `print("Här")` strax efter ett misstänkt stycke kod: ser du din notis vet du att den delen passerades utan att skriptet kraschade.

Avslutande ord!

Alla gör misstag, det krävs ett geni för att göra de stora spännande misstagen. När du börjar skapa program och grafer kommer du att åstadkomma saker som är mer komiska än användbara. Alla gör det, det är bara att gå vidare och lista ut vad som gick fel.

Övningar

Facit

1. Använd några data du har och skapa en for-loop för någon kategori. Det räcker t.ex. med att räkna antalet.
2. Skriv en funktion som räknar ut antalet dagar mellan dagens datum och ett datum du ger det (t.ex. antal dager sedan du föddes). Sedan kan du expandera det till att (förslagsvis)
   - ta två datum och räkna dagarna mellan dessa
   - att kontrollera vilket av de två som är äldst, och justera resultatet efter det.
3. Skriv ett skript som skapar en tabell (data frame) över Beställare och kön i acovid, med antal, medelålder och medel-resultat för varje kategori. Spara resultatet i en Excel-fil.