R for Byplankontoret

Håvard Karlsen

Invalid Date

Table of contents

Forord

Dette er en pamflet skrevet for Byplankontorets statistikere, for å gi en kort introduksjon til R. Teksten er skrevet i Quarto.

1 Introduksjon

Dette er en omfattende dokumentasjon av det jeg har gjort i R på Byplankontoret. I tillegg er det et forsøk på å vise de funksjonene man trenger for å arbeide i R. Det er ufattelig mange muligheter når det kommer til R. Dette er bare et lite utvalg, basert på det jeg har brukt mest.

2 Om R

Noen forskjeller på det å jobbe i R vs. Excel og SPSS.

R et et kodespråk. Det vil si at vi arbeider gjennom skript fulle av kode. Slik som dette:

```
2 + 2
```

```
[1] 4
```

```
print("Hello world")
```

```
[1] "Hello world"
```

```
sqrt(64)
```

```
[1] 8
```

```
# Dette er en kommentar og vil ikke leses av kodeleseren.
```

2.1 SPSS

R er mest likt SPSS, og spesielt SPSS' syntaks. Til forskjell fra SPSS er ikke det grafiske brukergrensesnittet (GUI) noe særlig nyttig i R.

Man kan kjøre R i et GUI som følger med R når man laster programmet, som heter R *hva enn versjonsnummeret er*, f.eks. `R 4.2.2`. Men det er bedre å bruke Rstudio til å arbeide med R i. Her får du et bra GUI som blant annet fullfører kodeforslag og har mange andre støttende funksjoner.

Vi arbeider vanligvis i skript, som har forkortelsen `.R`. Dette er likt SPSS' syntaksfiler (`.sps`). Du kan kjøre hele skriptet, eller kun deler av skriptet av gangen. Kjør deler av skriptet ved å enten ha markøren i den linja eller marker flere linjer og trykk `ctrl + enter`.

Til motsetning fra SPSS er Rs kodespråk lettere å lese og forstå (personlig mening). Man vil så klart aldri huske alle koder i R, men etter hvert vil en del av dem sitte fordi man bruker dem så ofte. Typiske eksempler på dette er `%>%`, `filter()` og `mutate()`. Se mer om disse i seinere kapitler.

2.2 Excel

Det er større forskjell på R og Excel. Excel er bygd rundt det grafiske grensesnittet. Det du ser er det du får (WYSIWYG). Dette har sine fordeler og bakdeler. Den største bakdelen, slik jeg ser det, er at Excel lar deg gjøre dumme ting. F.eks. hoppe over rader, forflytte en kolonne uten å mene det, glemme å markere alle felter, og det verste av alt: slå sammen celler.

Likevel, det er mange ganger det er bedre å bruke Excel.

Vi kan importere excel-filer til R, hvilket er veldig nyttig. Den største utfordringa med dette er at vi må kjempe mot de bakdelene jeg nevnte over.

I motsetning til både Excel og SPSS så lagrer R dataene bare i internt minne mens du arbeider med dem. Dvs. at du ikke er avhengig av å mellomlagre alt som en `.sav`, `.xlsx.`, eller `.csv`-fil. Dette kan bidra til å redusere behovet for mange versjoner av samme fil på ulike tidspunkter.

2.3 Hvorfor skal jeg bruke R?

En typisk tilbakemelding:

> Det tar tid å lære, det er en bratt læringskurve, og jeg får feilmeldinger hele tida.

Det er noen fordeler med R som er attraktive for oss:

- Når du har laga et skript kan du, uten særlig mange endringer, kjøre skriptet på nytt gang etter gang. Dette sparer deg for mye tid istedenfor å måtte starte på nytt hver gang.
  - Dette er delvis mulig i SPSS-syntaks alt. R oppfordrer i større grad til dette via funksjonene sine, og måten den håndterer data på.
- Man kan bruke R til alt. Fra før av kan vi spleise data i SPSS, lage tabeller i Excel, gjøre dem interaktive i Infogram, dele dem via Google sheets, etc. R kan gjøre alt dette i samme programvare/GUI.
- R lese og skrive til de fleste vanlige programmer. Dvs. at vi kan starte en prosess i Excel og så fortsette den i R. Eller vi kan importere en Stata-fil til R, gjøre noen pivots og lagre den som en SPSS-fil. Dermed kan R relativt sømløst puttes inn i arbeidsprosessen. (Enklest blir det så klart å gjøre alt i R.)

2.4 Versjonering

R, Rstudio, og alle pakkene til `R` kommer i ulike versjoner, f.eks. `R v.4.2.2`, `Rstudio 2023.03.0`, etc. Når man installerer en pakke vil den nyeste versjonen som er kompatibel med din versjon av `R` installeres. Her er noen ting å være oppmerksom på:

- Noen nye pakker funker ikke på gamle versjoner av `R`.
- Noen gamle pakker funker ikke eller litt annerledes på nye versjoner av `R`.
- Når pakker oppdateres vil noen ganger funksjonene deres endres.
  - Dette er en av bakdelene i `tidyverse`. De har endra på syntaksen sin slik at `tidyverse`-syntaks fra 2018 ikke gjelder i 2023. F.eks. pleide man å bruke `mutate_at()` før i tida for å mutere kun visse rader. Nå bruker man derimot en kombinasjon av `mutate()` og `across()` for å oppnå det samme. Dette er irriterende hvis du var vant til den gamle metoden.
- Du har **alltid tilgang til eldre versjoner av `R` og `R`s pakker**. Dette er et viktig kjennetegn ved FOSS (free, open-source software). Hvis du trenger en funksjon fra en gammel versjon av en pakke, kan du alltids nedgradere R-versjonen og laste inn den versjonen av pakka. Jeg nevner det her, men det er mer for viderekommende, og for Linux-fantaster.

Per nå er siste versjon vi har tilgang til på byplankontoret `4.2.3`. Hvilken versjon har jeg?

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 19045)

Matrix products: default

locale:
[1] LC_COLLATE=nb-NO.UTF-8  LC_CTYPE=nb-NO.UTF-8    LC_MONETARY=nb-NO.UTF-8
[4] LC_NUMERIC=C            LC_TIME=nb-NO.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.3  magrittr_2.0.3  fastmap_1.1.0   cli_3.4.1
 [5] tools_4.2.3     htmltools_0.5.4 rstudioapi_0.14 stringi_1.7.8
 [9] rmarkdown_2.17  knitr_1.40      stringr_1.4.1   xfun_0.34
[13] digest_0.6.30   jsonlite_1.8.3  rlang_1.0.6     evaluate_0.17
```

Jeg har forrige versjon, `4.2.2`. Grunnen er at IT installerte den nye `R`-versjonen i dag, og jeg ikke vil ta sjansen på at det er små endringer i koden som ødelegger noe jeg har gjort før. Mest sannsynlig vil det går bra. Små endringer, som å gå fra `x.x.2` til `x.x.3` vil nok ikke ha noen merkbare endringer.

3 Pakker og funksjoner

En av de store trekkplastra til R er det enorme biblioteket med utvidelser de har. Disse kommer i form av pakker (*packages* på engelsk). Egentlig er alt i R pakker. Når vi starter R får vi noen få pakker, hvor den viktigste er `base`. Vi kan se hvilke pakker vi har lasta inn slik

```
sessionInfo()[6]
```

```
$basePkgs
[1] "stats"     "graphics"  "grDevices" "utils"     "datasets"  "methods"
[7] "base"
```

Legg merke til at `base` er inkludert i denne lista. Dette er kjernen av `R`. Vi kommer til å supplere med masse andre pakker etter hvert som vi arbeider med ting. Pakkene er stort sett sentrert rundt å løse et eller annet problem. Her er noen eksempler:

- `haven`: importere og eksportere til andre statistikkprogrammer som SPSS, Stata, SAS
- `openxlsx`: lese og skrive excelfiler.
- `lubridate`: håndtere datovariabler på en bedre måte

Noen pakker bruker vi mer en andre. Et eksempel er `tidyverse`, men den diskuterer vi under.

Første gang du bruker en pakke må den installeres.

```
install.packages("RColorBrewer")
```

Vi laster vanligvis inn alle pakkene våre i toppen av skriptet.

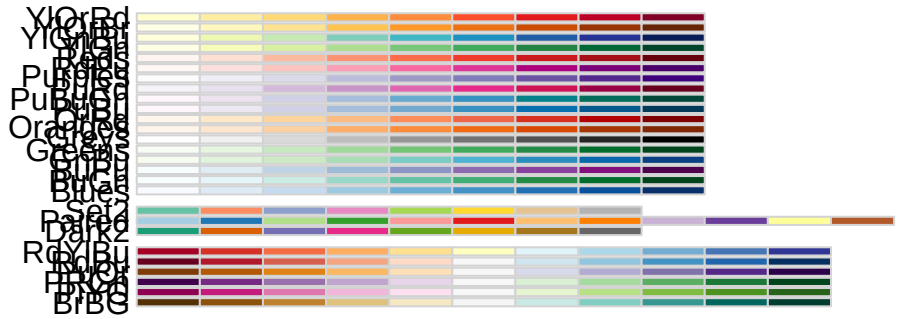
```
library(RColorBrewer)
```

Legg merke til at vi angir pakka som en streng (ved å bruke ") når vi installerer, men som et object (ved å ikke bruke ") når vi laster inn pakka. Det er en god grunn til det, men ikke en vi har tid å gå inn på nå.

En pakke trengs bare å *installeres* én gang, men den må *lastes inn* på nytt hver gang du starter en ny **session** i R. Du starter en ny session hver du starter programmet på nytt.

Du må laste inn pakka for å kunne ta i bruk funksjonene fra den. Her viser vi funksjonene til pakka `RColorBrewer` som kan brukes for finne komplementære farger.

```
display.brewer.all(colorblindFriendly = TRUE)
```



Det siste jeg sa er forresten ikke helt sant. Du kan kjøre en funksjon fra en pakke uten å ha lasta den inn. Da skriver du navnet på pakka, etterfult av to kolon og så navnet på funksjonen.

```
RColorBrewer::display.brewer.pal(n = 8, name = 'Dark2')
```

Dark2 (qualitative)

Så lenge pakka er lasta inn kan jeg bruke alle funksjonene fra pakka. Noen ganger trenger jeg bare én funksjon fra en pakke, og da benytter jeg meg av det over istedenfor å laste inn hele pakka.

3.1 Tidyverse

Tidyverse refererer til

- en designfilosofi
- en stor gruppe med pakker
- en spesifikk pakke som grupperer et lite antall pakker

Du kan lese mer om Tidyverse på nettsida deres. Det er også en lærebok som går grundigere gjennom alle funksjonene deres, R for Data Science.

Når man kjører `library(tidyverse)` vil den laste inn alle pakkene nevnt her. Blant annet `dplyr`, `ggplot2`, etc. I tillegg laster den inn enkeltfunksjoner fra andre pakker. F.eks. laster den inn pipe operatoren (`%>%`) fra `magrittr`. Mer om den seinere. Dermed er dette egentlig en snarvei for å slippe å laste inn flere pakker.

Tidyverse-pakkene er designa for å harmonisere med hverandre, og det gjør dem veldig sterke. Den underliggende filosofien gir også et bra rammeverk for andre pakker. Vinn-vinn.

Reint praktisk er det sånn at mange av funksjonene i `tidyverse` allerede eksisterer i `base R`. F.eks. filtrering, mutering, og etter `R v.4.1.`, pipe-funksjonen. Jeg bruker likevel `tidyverse`-variantene fordi disse er så mye lettere å forstå, skrive, og lese. De er utvikla for folk som jobber som oss, med tabeller og datasett. Som nybegynner er det ikke bare bare å forstå forskjellen mellom `base R` og `tidyverse`, så her er det viktigste:

- Når dere søker opp løsninger vil det ofte presenteres løsninger både i `base R` og i `tidyverse`. Dette skjer ofte på StackOverflow.
- De fleste `tidyverse`-funksjoner har et datasett som første argument i funksjonen. Dette gjør at vi lett kan *pipe* funksjoner etter hverandre.

3.2 Piper

Hvorfor er piper så nyttig? De lar oss flette sammen en serie operasjoner uten å måtte mellomlagre objekter. La oss si at vi har et datasett med biler og deres egenskaper. Vi vil

- filtrere ut dem som har under seks sylindre
- gjøre om vekta fra lbs. til kg.
- gruppere etter antall gir
- vise snitt av miles/gallon (mpg).

3.2.1 Uten pipa

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.3.6      v purrr   0.3.5
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.4.1
v readr   2.1.3      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```
mtcars <- mtcars
cars_filtered <- filter(mtcars, cyl >= 6)
cars_filtered_kg <- mutate(cars_filtered, wt = wt * 0.45359237)
cars_filtered_kg_grouped <- group_by(cars_filtered_kg, gear)
cars_filtered_kg_grouped_mean <- summarise(cars_filtered_kg_grouped, snitt = mean(mpg))
```

```
cars_filtered_kg_grouped_mean
```

```
# A tibble: 3 x 2
   gear snitt
  <dbl> <dbl>
1     3  15.7
2     4  19.8
3     5  16.8
```

3.2.2 Med pipa

```
library(tidyverse)
cars_filtered_kg_group_mean <- mtcars %>%
  filter(cyl >= 6) %>%
  mutate(wt = wt * 0.45359237) %>%
  group_by(gear) %>%
  summarise(snitt = mean(mpg))

cars_filtered_kg_grouped_mean
```

```
# A tibble: 3 x 2
   gear snitt
  <dbl> <dbl>
1     3  15.7
2     4  19.8
3     5  16.8
```

Det andre eksemplet er

1. mer lesbart
2. mindre stappfult av midlertidige objekter som vi seinere må slette

Jeg kommer til å bruke piper en god del både her og i alle skriptene mine. Så det er greit å vite hva det går ut på. Syntaksen `x %>% y` kan leses som `y får x`. Vi tar `x` og sender det til `y` som tar det inn som sitt første *argument*. Tidyverse-funksjonene er bygd rundt ideen om at det første argumentet til funksjonene er et datasett. Legg merke til at det er et datasett som er det første objektet i alle funksjonen jeg bruker i [med-piper].

Noen funksjoner, som `base::sum()` har ikke data som sitt første argument, men en vektor. Hvis man sender et datasett til `sum()` vil man få en feilmelding.

```
mtcars %>% sum(wt)
```

```
Error in mtcars %>% sum(wt): object 'wt' not found
```

For å få slike funksjoner til å funkere med ei pipe, kan man ofte bruke en funksjon fra `magrittr`:

```
mtcars %>% sum(.$wt)
```

```
[1] 14045.15
```

`.` blir her et alias for det aktuelle datasett, og dette er det samme som å skrive:

```
sum(mtcars$wt)
```

```
[1] 102.952
```

Da jeg lærte R var det `%>%` fra `magrittr` som var den gjeldende pipa. Den var så nyttig at ei pipe til slutt blei inkorporert i `base R`. Dette skjedde i `R 4.1.0`. `Base R`s pipe ser slik ut: `|>`. Den fungerer i hovedsak lik `%>%`. Når jeg fortsetter å bruke den gamle `magrittr`-pipa er det bare fordi jeg er gammel og ikke liker å endre på ting som funker. Dessuten har Rstudio en flott snarvei til `%>%` via `ctrl + shift + M`.

Dere velger altså sjøl om dere går for `%>%` eller `|>`. Husk bare at for å bruke `%>%` så må `tidyverse` eller `magrittr` lastes inn først. (`tidyverse` låner noen av funksjonene fra `magrittr`, men laster ikke inn *alle* funksjonene fra den pakka).

Forresten, noen ganger vil dere kanskje se pakker omtalt som online bibliotek (*library*). Og vi bruker jo funksjonen `library()` for å laste inn en pakke. Hva er forskjellen på en pakke og et bibliotek? I R er det den mappa hvor alle pakkene som er installert blir lagra som kalles **bibliotek**. Sjølve funksjonssamlinga kalles en **pakke**.

Referanser

En serie med nyttig ressurser og sider å drive produktiv prokrastinering.

Ressurser for å lære R

R for data science

Advanced R

Swirl

Produktiv prokrastinering

Brodrigues

Rweekly

Rbloggers