kurs-r-viderekomne

Innhold

1 R for viderekomne **3**

2 Indeksering **4**
2.1 Indeksering 4

3 Datatyper **6**
3.1 Datasett 7
3.2 Forskjellige datasetttyper 9

4 Kontroll **11**
4.1 Store prosesser 12

5 Løkker **13**
5.1 While-løkker 15

6 Funksjoner **17**
6.1 Lage en enkel funksjon 18
6.2 Lage en funksjon for fylke 19
6.3 Flere parameter 19
6.4 Standard/default parameter 20
6.5 Globalt vs, Lokalt-miljø 20
6.6 Varsling i funksjoner 21

7 R feilsøking **23**
7.1 Tips til feilsøking av kode 23
7.1.1 Generelle 23
7.1.2 Spesifikk 23
7.2 Tips til feilsøking av tekniske problemer 24
7.2.1 Generelle 24
7.2.2 Spesifikk 24

8 Videre bruk av R **26**
8.1 Metodebibliotek 26
8.2 fellesR 26
8.3 kurs: R i produksjon 26

1 R for viderekomne

Velkommen til kurset! Dette er et kurs for de som har litt R programmering fra før.

- Indeksering av vektorer
- Forklaring av forskjellige datasetttyper
- Kontroll setninger med if og else
- For- og whileløkker
- Hvordan skrive funksjoner
- Tips til feilsøking
- Andre ting og veien videre

2 Indeksering

Vektorer samler flere verdier til et objekt. De må være samme type (numeriske, logiske, karakterer).

Vi kan beregne direkte på alle elementer i en vektor:

```
alder <- c(49, 39, 51, 73, 41)
alder * 2
```

```
[1]  98  78 102 146  82
```

Vi kan kjøre tester på alle elementer i en vektor:

```
alder == 39
```

```
[1] FALSE  TRUE FALSE FALSE FALSE
```

2.1 Indeksering

Ved bruk av `[ ]` kan vi hente ut elementene i en vektor (eller et datasett). NB! I R (i motsetting til Python) starter indeksering fra 1.

For å hente ut første element:

```
alder[1]
```

```
[1] 49
```

For å ekskludere et element kan vi bruke -indeks:

```
alder[-1]
```

```
[1] 39 51 73 41
```

For å hente ut flere elementer kan vi spesifisere en sekvens:

```
alder[1:4]
```

```
[1] 49 39 51 73
```

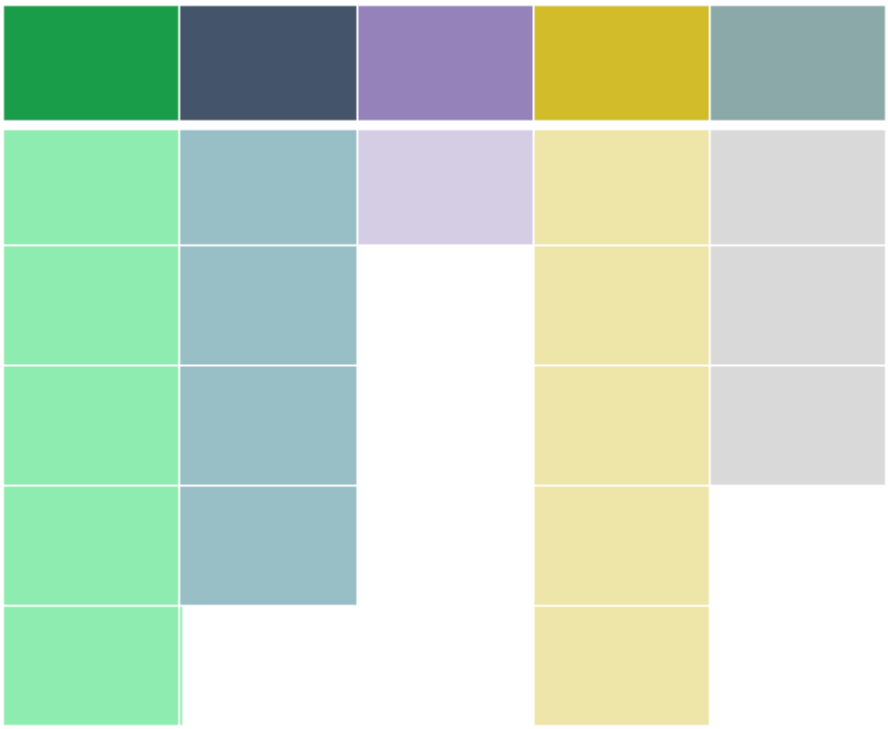
Vi kan også bruke indeksering til å endre et spesifikt element:

```
alder[1] <- 48
alder[1]
```

```
[1] 48
```

3 Datatyper

Lister samler objekter, vektorer eller datasett. I motsetning til vektorer kan de inneholde forskjellige datatyper og forskjellig lengder.



Vi lager lister ved å bruke `list()`:

```
kommune_list <- list(sted = c("Oslo", "Kongsvinger", "Halden"),
                     snitt_lonn = c(636, 504, 552),
                     antall_lonnstakere = c(467400, 8300, 12600),
                     nivaa = "Kommune")
kommune_list
```

```
$sted
[1] "Oslo"        "Kongsvinger" "Halden"
```

```
$snitt_lonn
[1] 636 504 552

$antall_lonnstakere
[1] 467400   8300  12600

$nivaa
[1] "Kommune"
```

Vi kan bruke **$** for å få tilgang til en vektor eller et element i en liste:

```
kommune_list$snitt_lonn
```

```
[1] 636 504 552
```

Vi kan kombinere dette med `[ ]` for å hente ut elementer:

```
kommune_list$snitt_lonn[1]
```

```
[1] 636
```

3.1 Datasett

Datasett er lister som samler vektorer med samme lengde.

Vi bruker `data.frame()` for å lage et vanlig R datasett:

```
kommune_data <- data.frame(sted = c("Oslo", "Kongsvinger", "Halden"),
                           antall_lonnstakere = c(467400, 8300, 12600))
```

Igjen kan vi bruke **$** for å få tilgang til en vektor og `[ ]` for å hente ut elementer:

```
kommune_data$snitt_lonn[2]
```

```
NULL
```

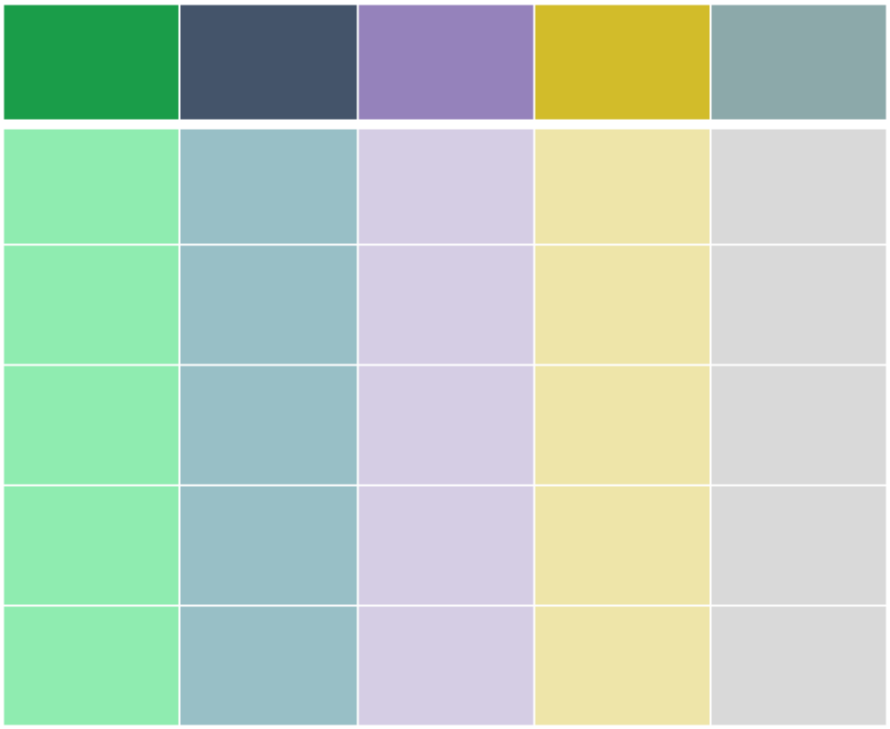


Figure 3.1: Eksempel datasett

NB: I *tidyverse* bruker vi variabelnavn istedenfor $.Dette har konsekvenser for kjøretid og noen begrensninger, likevel er *tidyverse* en veldig intuitiv og givende pakke for analysering av data.

Noen nyttige funksjoner som kan benyttes ved datasett:

```
nrow(kommune_data)
```

```
[1] 3
```

```
ncol(kommune_data)
```

```
[1] 2
```

```
head(kommune_data)
```

```
         sted antall_lonnstakere
1        Oslo              467400
2 Kongsvinger                8300
3      Halden               12600
```

```
library(tidyverse)
glimpse(kommune_data)
```

```
Rows: 3
Columns: 2
$ sted               <chr> "Oslo", "Kongsvinger", "Halden"
$ antall_lonnstakere <dbl> 467400, 8300, 12600
```

3.2 Forskjellige datasetttyper

Det er forskjlellige måte å formatere data i R. Disse er mest vanlig:

Data frame type	Code for formatting
Normal data frame	`data.frame()`
tibble (tidyverse)	`as_tibble()`

Data frame type	Code for formatting
data table (data.table)	`data.table()`

4 Kontroll

For å sammenligne og gjøre noe basert på en betingelse kan vi bruke **`ifelse()`**. Vi må spesifisere betingelsen først, og så hva som skal returneres dersom betingelsen er sann, og så hva som skal returneres om betingelsen er usann.

```
alder <- c(49, 39, 51, 73, 41)
ifelse(alder < 50, "yngre", "eldre")
```

```
[1] "yngre" "yngre" "eldre" "eldre" "yngre"
```

Dette kan brukes for å lage nye variabler i et datasett:

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   1.0.1
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.5.0
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()
```

```
dt <- data.frame(id = 1:5, alder)
dt %>%
  mutate(alder_kat = ifelse(alder < 50, "yngre", "eldre"))
```

```
  id alder alder_kat
1  1    49     yngre
2  2    39     yngre
3  3    51     eldre
4  4    73     eldre
5  5    41     yngre
```

4.1 Store prosesser

For å kontrollere store/lengre prosesser kan vi benytte **`if`** og **`else`**. Disse kan gå over flere linjer og ta formatet:

```
if (betingelsen){
  gjør dette ...
} else {
  gjøre dette istedenfor ...
}
```

For eksempel:

```
if (all(dt$alder < 70)){
  print("Alle IOer er under 70")
} else {
  print("Alle IOer med alder 70+ er fjernet.")
  dt %>%
    filter(alder < 70)
}
```

```
[1] "Alle IOer med alder 70+ er fjernet."
```

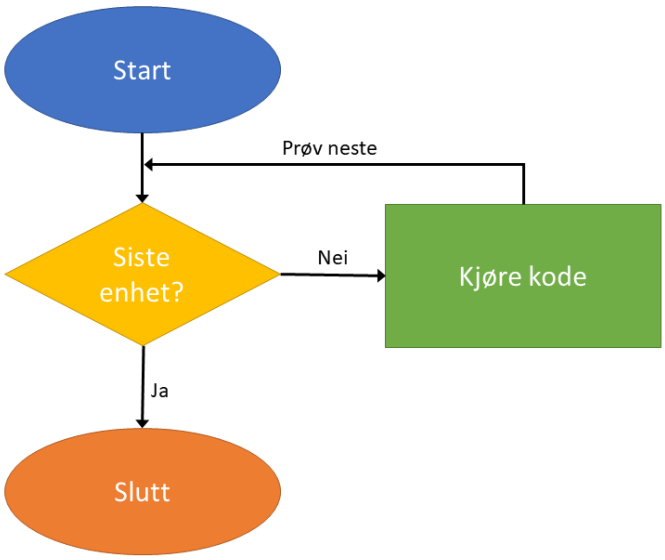
```
  id alder
1  1    49
2  2    39
3  3    51
4  5    41
```

5 Løkker

For å gjøre den samme prosessen flere ganger kan vi lage løkker. Løkker har noen fordeler:

- Vi slipper å skrive den samme koden flere ganger.
- Enklere å endre noen verdier/variabler i koden (kun ett sted).
- Hvis vi finner en feil trenger vi kun å rette den ett sted.

For-løkker brukes til å kjøre gjennom kode et bestemt antall ganger



Det er vanlig å kjøre gjennom en sekvens. For eks:

```
alder <- c(49, 39, 51, 73, 41)

for (i in 1:5){
  print(i)
  print(alder[i])
```

```
}
```

```
[1] 1
[1] 49
[1] 2
[1] 39
[1] 3
[1] 51
[1] 4
[1] 73
[1] 5
[1] 41
```

Vi kan også lage løkker med en vektor:

```
for (a in alder){
  print(a)
}
```

```
[1] 49
[1] 39
[1] 51
[1] 73
[1] 41
```

5.1 While-løkker

While-løkker sjekk en betingelse for å bestemme om den skal fortsette å kjøre.

For eksempel:

```
n <- 1
while (n < 10){
  print(n)
  n <- n + runif(1)
}
```

```
[1] 1
[1] 1.910266
[1] 2.219209
[1] 2.347715
[1] 2.841849
[1] 3.309068
[1] 3.839014
[1] 4.333365
[1] 4.378831
[1] 5.13833
[1] 5.243385
[1] 5.909941
[1] 6.624096
```

```
[1] 6.886486
[1] 7.772019
[1] 8.366856
[1] 9.319649
```

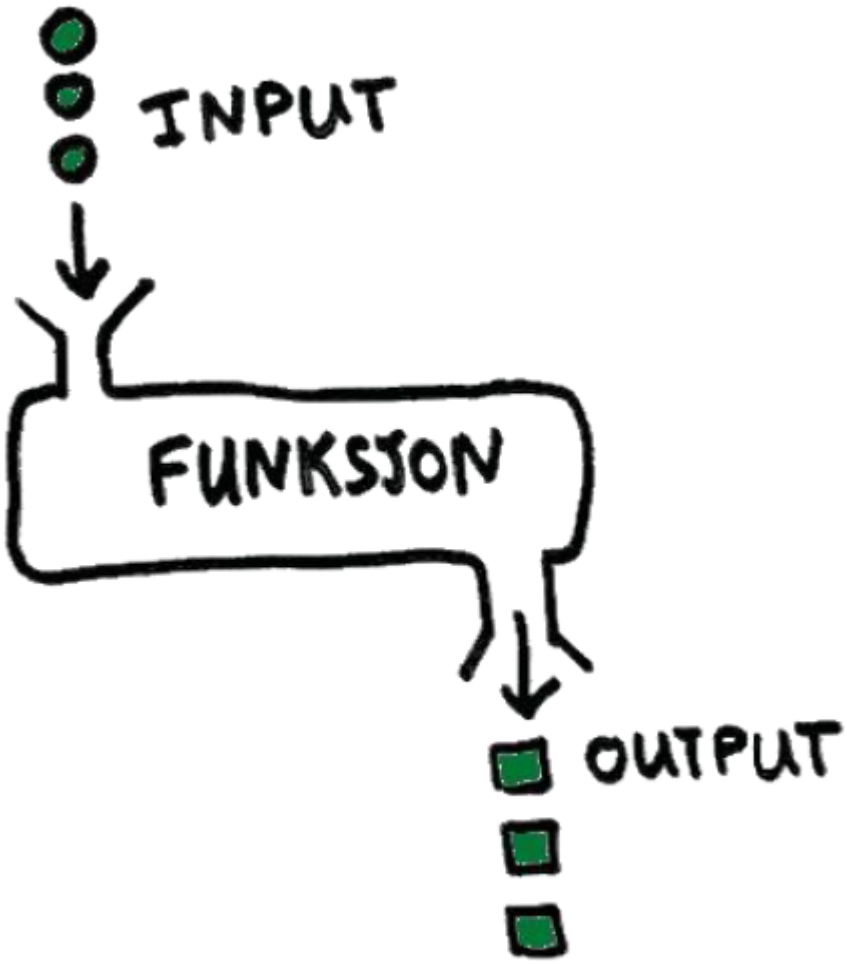
```
n
```

```
[1] 10.31281
```

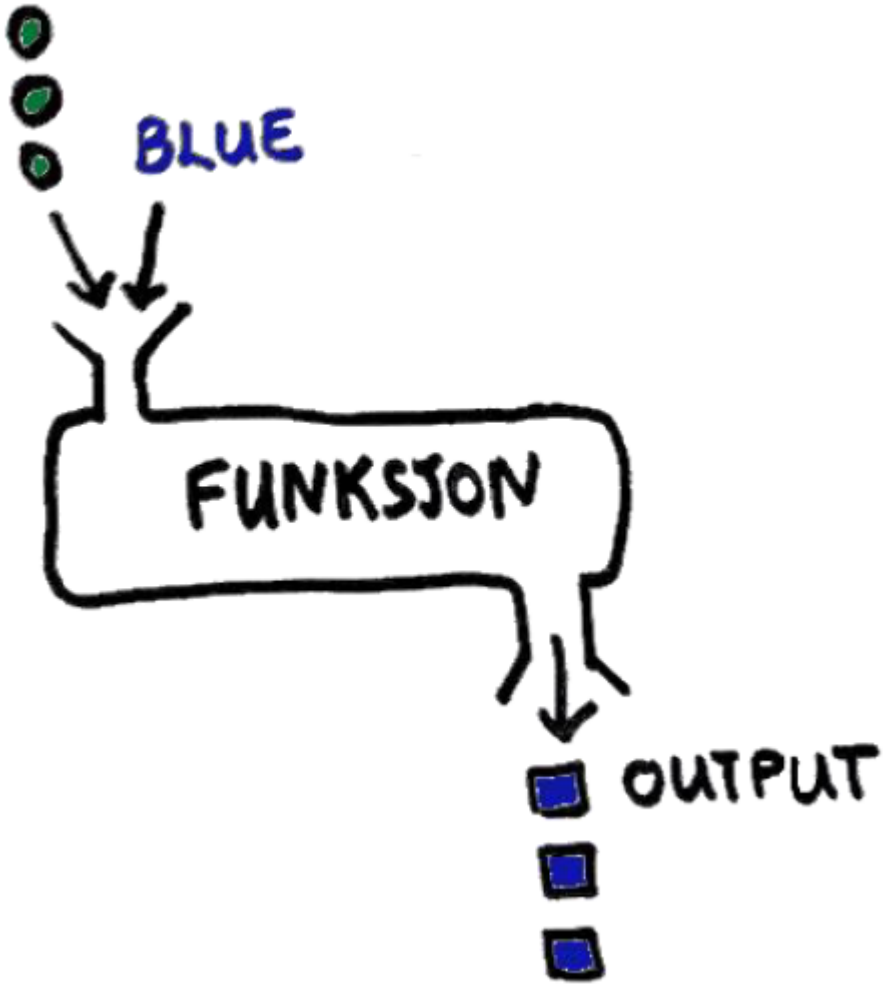
While-løkker brukes ofte i prosesser som har en tilfeldig komponent. I eksempelet over trekker `runif()` funksjonen et tilfeldig tall mellom 0 og 1.

6 Funksjoner

En funksjon er en kodedel som kan brukes om og om igjen. De ligner på SAS-makroer og brukes til å automatisere prosesser. Den har en input (det som sendes inn til funksjonen) og en output (det som kommer ut).



En parameter er tilleggsinformasjon som sendes inn til funksjonen for å spesifisere videre hva funksjonen skal gjøres.



Bruk av funksjoner kan være nyttig for gjenbruk og abstraksjon.

6.1 Lage en enkel funksjon

Vi lager en funksjon ved å allokere et navn og spesifisere `function()`:

```
min_func <- function(){
  print("hello")
}
```

Etterpå kan vi kjøre funksjonen med:

```
min_func()
```

```
[1] "hello"
```

6.2 Lage en funksjon for fylke

Her skal vi lage en funksjon som kan ta kommunenummer som input og returnere fylkenummer. Vi spesifiserer kommunenummer som en parameter i funksjonen. Vi bruker `substr()` for å plukke ut de første to sifferne.

```
lage_fylke <- function(kommunenr){
  substr(kommunenr, 1, 2)
}

lage_fylke("0301")
```

```
[1] "03"
```

Funksjoner kan gå over flere linjer. Den siste linjen er det som returneres. Hva som returneres kan også spesifiseres med `return()` ved behov, særlig nyttig i komplekse funksjoner med flere output.

6.3 Flere parameter

Funksjoner kan ta mer enn èn paramter. For eksempel i fylke-funksjonen kanskje vi ønsker å sjekke lengden for å se om ledende 0-ere har falt av.

```
lage_fylke <- function(kommunenr, sjekk_lengde){
  if(sjekk_lengde == TRUE){
    kommunenr <- ifelse(nchar(kommunenr) == 3,
                        paste("0", kommunenr, sep = ""),
                        kommunenr)
  }
  fylke <- substr(kommunenr, 1, 2)
  fylke
}
```

```
lage_fylke(kommunenr = "301", sjekk_lengde = TRUE)
```

```
[1] "03"
```

```
lage_fylke(kommunenr = "301", sjekk_lengde = FALSE)
```

```
[1] "30"
```

6.4 Standard/default parameter

Vi kan sette standard parameter verdier for å slippe å spesifisere hver gang. For eksemepel, samme funksjon over kan ha `sjekk_lengde=TRUE` som standard parameter.

```
lage_fylke <- function(kommunenr, sjekk_lengde = TRUE){
  if(sjekk_lengde == TRUE){
    kommunenr <- ifelse(nchar(kommunenr) == 3,
                        paste("0", kommunenr, sep = ""),
                        kommunenr)
  }
  fylke <- substr(kommunenr, 1, 2)
  fylke
}

lage_fylke("301")
```

```
[1] "03"
```

Noen ganger kalles disse for "named parameters" eller "keyword arguements". Standard parameter kommer alltid til sist.

6.5 Globalt vs, Lokalt-miljø

Når vi lager en funksjon, lager vi et lite lokalt-miljø. Varaiabler som lagres inn i en funksjon påvirker ikke det globale miljøet og blir slettet når funksjonen er ferdigkjørt. For eksempel, om vi har en enkel funksjon som returnerer verdien av parameter `x` vil ikke dette påvirkes om vi har en x i det globale miljøet:

```r
funcx <- function(x){
  x
}
```

```r
x <- 2
funcx(x = 4)
```

```
[1] 4
```

```r
x
```

```
[1] 2
```

6.6 Varsling i funksjoner

Noen ganger ønsker vi at funksjonen skal si ifra om noe er litt rart eller feil. For at funksjonen skal stoppe bruker vi **`stop()`**. For at det skal gi et varsel bruker vi **`warning()`**.

For eksempel, her stopper funksjon om kommunenr kun er 2-siffer. Ved 3-siffer gis et varsel at en ledende 0 er lagt på.

```r
lage_fylke <- function(kommunenr){
  if (nchar(kommunenr) <= 2){
    stop("Kommune nummer var ikke gjeldig.")
  }
  if (nchar(kommunenr) == 3){
    warning("Kommunenummer er lendge 3 og har blitt fylt med en ledende 0\n")
    kommunenr <- paste("0", kommunenr, sep = "")
  }
  fylke <- substr(kommunenr, 1, 2)
  fylke
}
```

```r
lage_fylke(kommunenr = "03")
```

```
Error in lage_fylke("03") : Kommune nummer var ikke gjeldig.
```

```r
lage_fylke(kommunenr = "301")
```

```
Warning in lage_fylke(kommunenr = "301"): Kommunenummer er lendge 3 og har blitt fylt med en
[1] "03"
```

```r
lage_fylke(kommunenr = "0301")
```

```
[1] "03"
```

7 R feilsøking

Det kan være frusterende å programmere når det oppstår feil. Det er ikke alltid lett å forstå feilmeldinger, særlig hvis du er ny til R.

Feil kan klassifiseres som "kode" eller "tekniske" feil. Kodefeil oppstår når vi programmerer ting litt feil, for eksempel bruker funksjoner på en feil måte. Disse type feilene kan vi som oftest finne hjelp til på nett og rette opp selv. Tekniske feil kan oppstår hvis vi mangler andre applikasjoner eller pakker, når vi prøve å kjøre R på forskjellige platform/ sammen med andre verktøy osv. I noen tilfelle kan vi løse disse selv men andre ganger trenger vi støtte fra IT.

7.1 Tips til feilsøking av kode

7.1.1 Generelle

- Kjør kode linje-ved-linje for å isolere feilen. Sjekk også etter røde kryss på venstre side av koden som indikere feil (om du jobber i RStudio). Av og til kommer disse kryss på linjer etter feilen, for eks. om parentes ikke er lukket.
- Kopier feilmelding du får inn til google (rens bort variabel/datasett navn osv. først).
- Sjekk om du har NA verdier i data og om det kan være årsaken til problemer. Håndtering funksjonen du kjøre NA verdier?
- For skriving av egne funksjoner kan det hjelpe å sette inn noen print-setninger for å se hvor det stopper opp. Eller bruk debug(), en innebygd R funksjon. Les dokumentasjon for debug().
- Sjekk at data er formatert som forventet (tibble vs. data.frame).
- Prøv å spørre Chatgpt
- Spør om hjelp på yammer.

7.1.2 Spesifikk

Vanlige feilmeldinger	Tips
"argument is of length zero"	Se statology
"could not find function"	Sjekk at du har stavet riktig (små og store bokstaver). Sjekk at pakkene er installerte og kalt inn.

Vanlige feilmeldinger	Tips
“subscript out of bounds”	Sjekk at indeks ikke er større en vektor/datasett
“no applicable method”	Sjekk at data er formatert som forventet (eg data.frame, tibble, data.table osv)
“cannot open file”	Sjekk at du har stavet filnavn riktig, og sjekk filsti. Hvor er du? Bruk `getwd()`. Naviger til foreldre mappen ved .. i sti. Bruk Sys.getenv(“ARBTAKER”) for å hente inn miljøvariabler. Du kan også skriv ut alle filene ved `list.files()`

7.2 Tips til feilsøking av tekniske problemer

7.2.1 Generelle

- Kopier feilmelding inn til google.
- Send en melding til kundeservice. Prøv å beskrive problemet. Lim inn feilmelding og kode du har kjørt.
- Hvis det er på Dapla - skriv på Slack: ‘hjelp_dapla’. Gjerne prøve å lage et lite eksempel av når problemet oppstår som kalles et Minimal reproducible example

7.2.2 Spesifikk

Feilmeldinger	Tips
shiny	• Bruker du jupyter? – Bruk wrapper funksjon i +fellesr
pakke installering	• Bruk renv :-) – første gang: kjør `renv::init()` – bruk `renv::install('<pakker>')` for å installere – Ikke står på hjemmeområde men i en prosjektmappe. – Hvis “no package available”, sjekk i renv.lock at URL er riktig ∗ URL skal ha “nexus.ssb.no” i navn – For multi-mappe prosjekter - har en renv i hovedmappen ∗ bruk `renv::autoload()` i notebookene

Feilmeldinger	Tips
Git	• Mangler git fane i RStudio? – Hvis du har en lokal RStudio på PC sjekk ut happygitwithr • Får ikke commit? – Husk å skrive inn en melding før du trykker på commit knappen. • Pull/push knapp er grå? – Har du noen endringer/commits til å pushe? – Bytte til kommandoer for å sjekke status med `git status` – Evt. bruk kommandoer istedenfor knapper

8 Videre bruk av R

`klassR` pakken er utviklet i SSB for å lett hente ut klassifikasjoner og kodelister fra KLASS. Mer info om hvordan du kan bruke pakken ligger i en introduksjon til klassR pakken

8.1 Metodebibliotek

Seksjon for Metoder har samlet nyttige metodiske funksjoner for bruk i et statistikk produksjonsløp. Både interne og eksterne utviklet funksjoner er inkluderte og har blitt testet. Metodebiblioteket er under utvikling men er tilgjengelig for alle på GitHub.

8.2 fellesR

R-pakken `fellesR` er en samling av funksjoner som kan være nyttige for flere på SSB. Alle er velkomne til å bidra med egne funksjoner om de synes andre kan ha nytte av de. Flere av de funksjoner skal vi gå gjennom i et nytt kurs 'R i produksjon'.

8.3 kurs: R i produksjon

Vi skal holde et nytt kurs med tips og anbefalinger om hvordan å bruke R i en produksjon setting. Tema for kurset inkluderer:

- innlesning av parquet-filer på Dapla
- pakkehåndtering med `renv`
- opplasting til Statbank
- kjøring av R og python i jupter notebooks
- generelle tips til organisering av kode