

kurs-r-viderekomne

Innhold

1	R for viderekomne	3
1.1	Del 2	3
I	Del 1	4
2	Indeksering	6
2.1	Indeksering	6
3	Data typer	8
3.1	Datasett	9
3.2	Forskjellige datasett typer	11
4	Kontroll	12
4.1	Store prosesser	13
5	Løkker	14
5.1	While-løkker	16
II	Del 2	18
6	Funksjoner	20
6.1	Lage en enkel funksjon	21
6.2	Lage en funksjon for fylke	22
6.3	Flere parameter	22
6.4	Standard/default parameter	23
6.5	Global vs Lokal-miljø	23
6.6	Varsling i funksjoner	24
7	Videre bruk av R	26
7.1	Metodebibliotek	26
7.2	fellesR	26
7.3	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. Kurset er delt i to:

- [Indexering av vektorer](#)
- [Forklaring av forskjellige datasett typer](#)
- [Kontroll setninger med if og else](#)
- [For- og while-løkker](#)

1.1 Del 2

- [Hvordan å skrive funksjoner](#)
- [Andre ting og vei videre](#)

Part I

Del 1

I del 1 av kurset skal vi snakke om:

- Indexering
- Datasett typer
- Kontroll
- Løkker

2 Indeksering

Vektorer samler flere verdier til et objekt. De må ha samme type innhold (for eks. alle numeriske). Vi kan beregne direkte på alle elementer i en vektor. For eks:

```
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. For eks:

```
alder == 39
```

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

2.1 Indeksering

Ved bruk av [] kan vi hente ut elementer i en vektor (eller datasett). I R (i motsetning til Python), starter indeksering fra 1! For å hente ut første element for eks:

```
alder[1]
```

```
[1] 49
```

For å ekskludere et element kan vi bruke -indeks. For eks:

```
alder[-1]
```

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

For å hente ut flere elementer kan vi spesifisere et sekvens

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

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

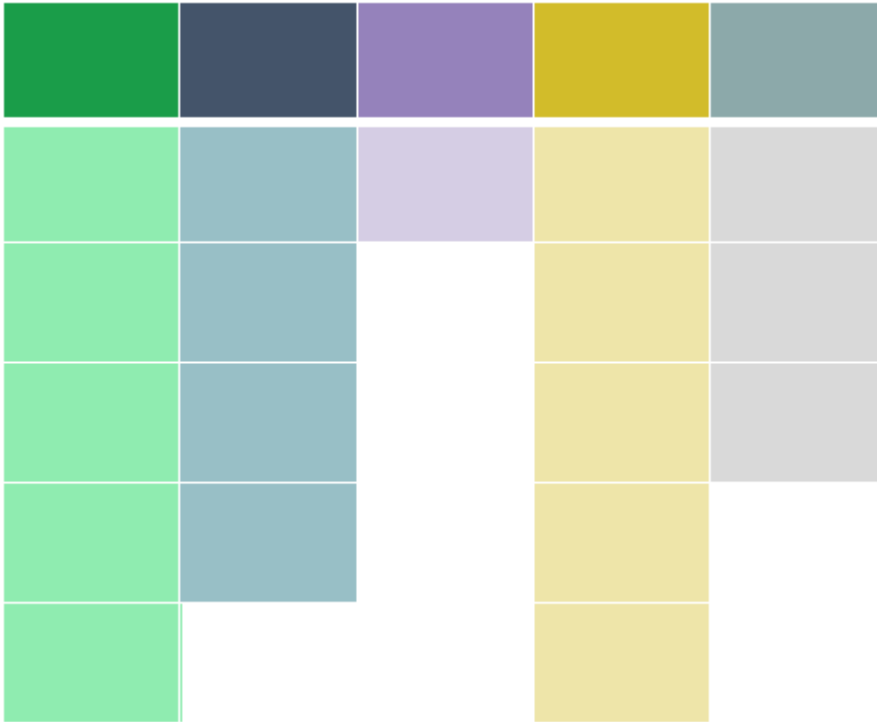
Vi kan også bruke indeksering til å endre et spesifikk element. For eks:

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

```
[1] 48
```

3 Data typer

Lister samler objekter/vektorer/datasett. De kan har forskjellige type og størrelser.



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 et vektor eller 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 bruke `data.frame()` for å lage en vanlig R datasett

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

Igjen, vi kan bruke `$` for å få tilgang til et vektor og `[]` for å hente ut elementer.

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

NULL

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

Noen nyttige funksjoner som kan benyttes ved datasett

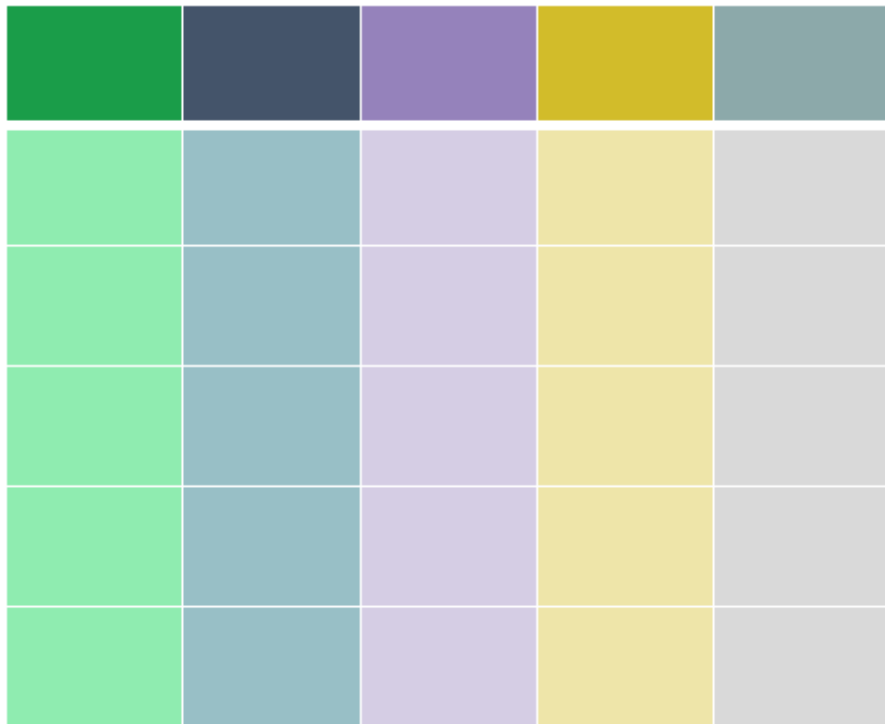


Figure 3.1: Eksempel datasett

```
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 datasett typer

Det er forskjellige måter å formatere data i R. Disse er mest vanlig:

Data frame type	Code for formatting
Normal data frame	<code>data.frame()</code>
tibble (tidyverse)	<code>as_tibble()</code>
data table (data.table)	<code>data.table()</code>

4 Kontroll

For å sammenlign og gjøre noe baserte på en betingelse kan vi bruke `ifelse()`. Spesifisere betingelsen først, og så hva skal returnere ved sann, og så hva ska returneres om betingelsen er usann.

For eksempel:

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

```
[1] "ungere" "ungere" "eldere" "eldere" "ungere"
```

Dette kan brukes for å lage nye variabler i et datasett. For eksempel

```
library(tidyverse)
```

```
-- Attaching packages ----- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   0.3.5
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, "ungere", "eldere"))
```

	id	alder	alder_kat
1	1	49	ungere
2	2	39	ungere
3	3	51	eldere
4	4	73	eldere
5	5	41	ungere

4.1 Store prosesser

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

```
if (betingelsen){
  gjør dette kode ...
} 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."
```

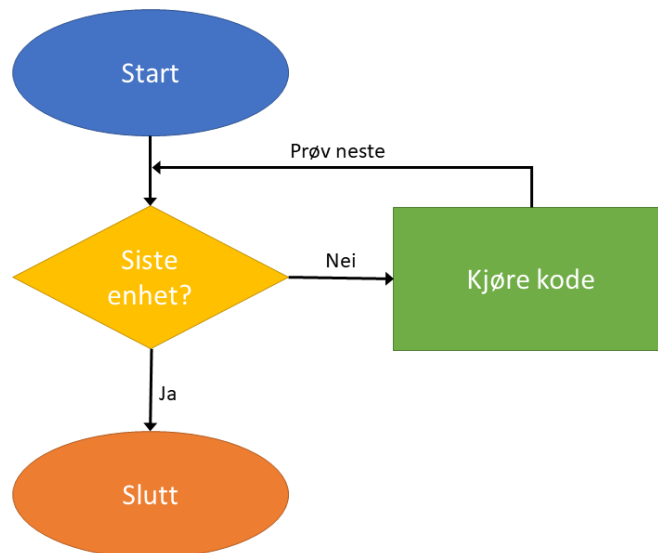
```
id alder
1 1    49
2 2    39
3 3    51
4 5    41
```

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 noe verdier/variabler i koden (kun ett sted).
- Hvis vi finner en feil, da trenger vi kun å rette det 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 eks:

```
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 det skal fortsette å kjøres.

For eksempel:

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

```
[1] 1
[1] 1.610719
[1] 1.620272
[1] 2.005332
[1] 2.294805
[1] 2.802222
[1] 3.80153
[1] 4.3314
[1] 4.920425
[1] 4.962673
[1] 5.304737
[1] 5.555578
[1] 5.644786
```



```
[1] 6.55605  
[1] 7.435769  
[1] 7.506605  
[1] 8.380082  
[1] 8.779904  
[1] 9.40241
```

```
n
```

```
[1] 10.15622
```

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

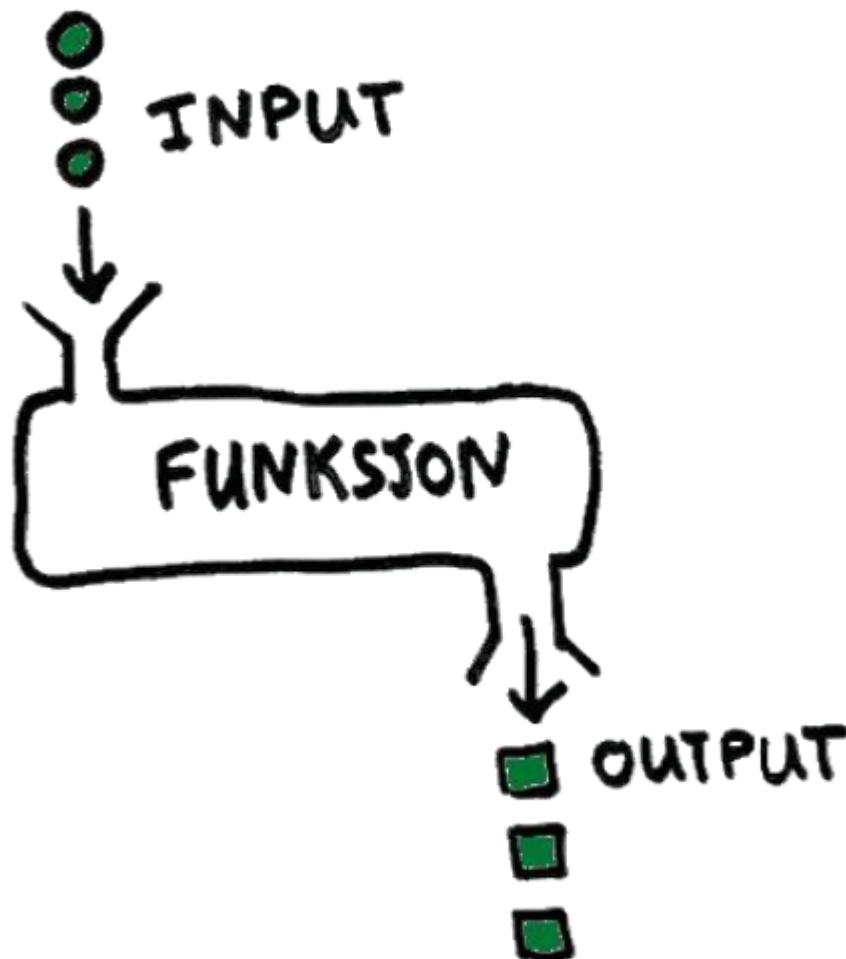
Part II

Del 2

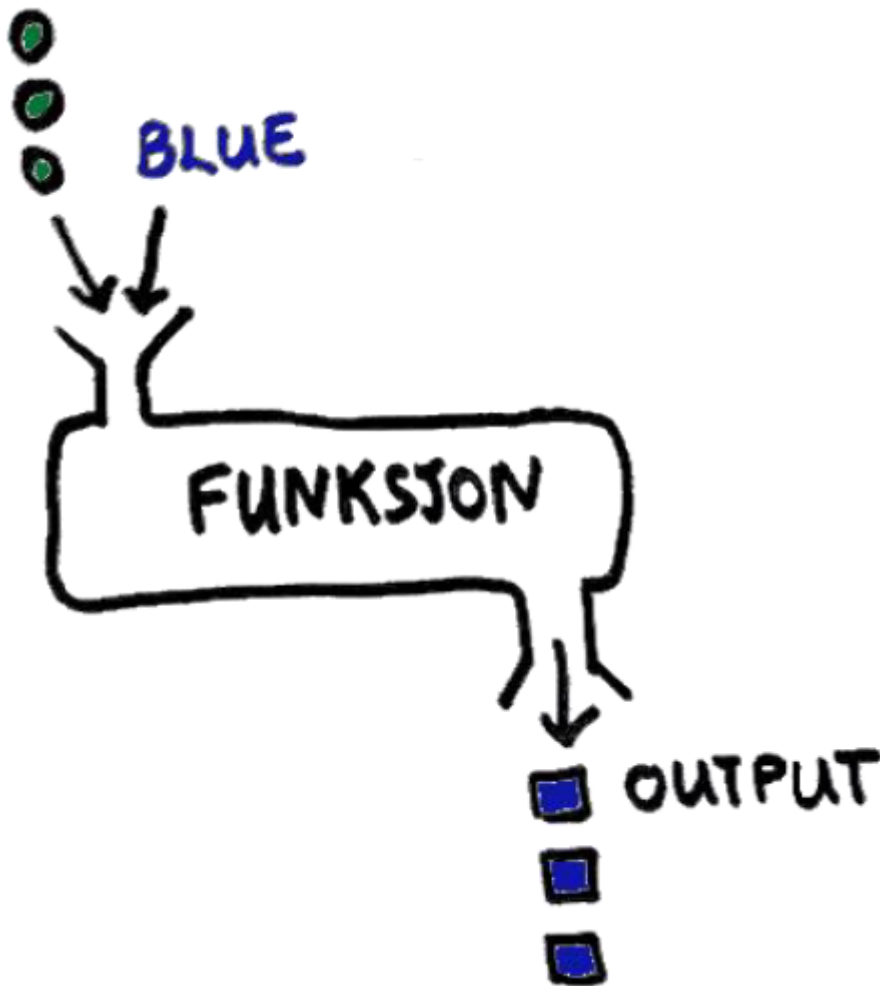
I del 2 av kurset skal vi snakke om: - [Funksjoner](#) - [Videre](#)

6 Funksjoner

En funksjon er en del av kode som kan brukes om-og-om igjen. Den har en input (det som sendes inn til funksjonen) og en output (det som kommer ut).



Parameter er tilleggsinformasjon som sendes inn til funksjonen for å spesifisere mer.



Bruk av funksjoner kan være nyttig: - Gjenbruk - Abstraksjon (trenger ikke å vite hvordan)

6.1 Lage en enkel funksjon

Vi lager en funksjon ved å allokere det et navn og spesifisere `function()`. For eks

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

Og etterpå kjøre vi funksjonen ved

```
min_func()
```

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

6.2 Lage en funksjon for fylke

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

```
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. Det kan også spesifiseres med `return()` ved behov, særlig i komplekse funksjoner med multiple output.

6.3 Flere parameter

Funksjoner kan ta mer enn én parameter. For eksempel i fylke-funksjonen vi kanskje ønsker å sjekke lengde for å se om ledende 0-er 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 sett et parameter verdier for å slippe å spesifisere hver gang. For eksempel, samme funksjon over kan ha `sjekk_lengde=TRUE` som standard.

```
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 arguments”. Standard parameter alltid kommer til sist

6.5 Global vs Lokal-miljø

Når vi lage en funksjon, lage vi en liten lokal-miljø. Variabler som lagres inn i en funksjon påvirke ikke global-miljø og er slettet når funksjonen er ferdig-kjørt. For eksempel om vi har en enkel function som returneres verdien av parameter `x` vil ikke dette påvirke om vi har en `x` i det global-miljøet:

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

```
}
```

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

```
[1] 4
```

```
x
```

```
[1] 2
```

6.6 Varsling i funksjoner

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

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

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

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

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

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

```
Warning in lage_fylke(kommunenr = "301"): Kommunennummer er lendge 3 og har blitt fylt med en
```



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

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

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

7 Videre bruk av R

`klassR` pakken er utviklet i SSB for å lett hente ut klassifikasjoner og kodelister fra KLASS. Mer info om hvordan å bruke pakken ligger i en [introduksjon til klassR pakken](#)

7.1 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](#).

7.2 fellesR

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

7.3 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 jupyter notebooks
- generelle tips til organisering av kode