Political Data Science

Lektion 4:

R Workshop III: Programmering & Git

Undervist af Jesper Svejgaard, foråret 2018
Institut for Statskundskab, Københavns Universitet
github.com/jespersvejgaard/PDS

I dag

1. Opsamling fra sidst
2. Eksamen
3. Pipes, funktioner & loops
4. Git, GitHub & version control
5. Workshop
6. Opsamling og næste gang

Overblik

1. Intro til kurset og R
2. R Workshop I: Explore
3. R Workshop II: Import, tidy, transform
4. R Workshop III: Programmering & Git
5. Web scraping & API
6. Tekst som data
7. Visualisering
8. GIS & spatiale data
9. Estimation & prædiktion
10. Superviseret læring I
11. Superviseret læring II
12. Usuperviseret læring
13. Refleksioner om data science
14. Opsamling og eksamen

1. Opsamling fra sidst

Opsamling fra sidst

- tidying:
  - `gather()`, `spread()`, `separate()` og `unite()`
- relationelle data, keys, joins:
  - mutating joins: `left_join()`, `right_join()`, `full_join()`, `inner_join()`
  - filtering joins: `semi_join()`, `anti_join()`
  - set operations: `intersect()`, `union()`, `setdiff()`
- gennemgang af opgaver (se `03_script.R` på GitHub)

2. Eksamen

Om eksamen I

Form:

- Omfang: Fri seminaropgave på 10 - 20 ns.
- Fri opgave => vide rammer, fx mere som en rapport. Afhænger af indholdet.

Indhold:

- Fri opgave => vide rammer. Eneste krav: Politologisk problemstilling (men meget er politologisk).
- Typer: Egen problemstilling, replikationsstudium, specialeforstudium
- Litteratur/teori: Afhængigt af opgavetype. Ikke krav om ekstra litteratur
- Abstraktionsniveau: Er lidt noget andet, end vi kender det, fx er eksplorativt studiue fint. Men man kan både lave et elegant og gennemtænkt eksplorativt studie, som er godt håndværk, og et, som ikke er.
- Se evt. målbeskrivelsen på GitHub (men brug den ikke som tjekliste)

Tilgang:

- Se det som en mulighed for for én gangs skyld at kunne eksperimentere
- En del af kurset handler om at introducere en række metoder og klæde jer på til videre udforskning
- Er man lidt lost, så overvej replikationsstidum eller tænk: hvad vil jeg skrive speciale om?
- Metode => problemstilling VS problemstilling => metode

Om eksamen II

Proces:

- Det er ikke nemt at finde på en god problemstilling
- En god problemstilling kræver forarbejde.
- En problemstilling er som et oplæg til en smash
- Vi vil have eksamens-sessioner undervejs, bl.a. næste gang

Eksempler:

- Eksplorativ analyse af, om man kan bruge Twitter-data til at visualisere politisk holdningsdannelse i en dansk kontekst
- Forudsigelse af stemmeadfærd vha. data fra valgundersøgelser
- Usuperviseret analyse af partiernes historiske stemme-alliancer fra ft.dk
- Analysere politiske skillelinjer tekstanalyse af politiske taler
- Gode cases fra VKM?

3. Highlights fra pensum: Pipes, funktioner & loops

This is not a pipe

This is!

```
# %>% - 'then'
flights %>% select(year:dep_time)

# %T>% - parser venstresiden
flights %>% select(dep_delay, arr_delay) %T>% plot() %>% select(dep_delay)

# %$% - referere til variable i funktioner fra Base R
flights %$% cor(dep_delay, arr_delay)

# %<>% - assignment
flights %<>% select(year:dep_time)
```

Funktioner

- Automatisering af arbejde
- Nemmere at vedligeholde
- Nemmere at overskue
- Færre fejl

Funktioner

Består af tre elementer: navn + argumenter + body

```
# Definerer funktion
funktion_division <- function(arg1, arg2){
  arg1 / arg2
}

# Eksekverer funktion med 15 og 3 som inputs
funktion_division(15, 3)
```

```
## [1] 5
```

Læg mærke til 'stilen': indryk + prefix/suffix + linjeskift + {}

Funktioner

Default values

```
# Definerer funktion
funktion_eksponent <- function(tal, eksponent = 2){
  tal ^ eksponent
}

# Eksekverer funktion
funktion_eksponent(5)
```

```
## [1] 25
```

```
funktion_eksponent(5, 3)
```

```
## [1] 125
```

Conditionals - if, then, else

```
# Definerer et køn
køn <- "t"

# Skriver en conditional
if (køn == "k") {
  print("Kvinde")
} else if (køn == "m") {
  print("Mand")
} else print("Andet")
```

```
## [1] "Andet"
```

Funktioner & conditionals i lykkeligt ægteskab

```
# Funktion til at forudsige konsekvens af at køre i byen
strafudmåler <- function(fart){
  if (fart < 51){
    "Du er en flink bilist!"
  } else if (fart > 50 && fart < 67){
    "Du er en bandit. Men du kan slippe med en bøde!"
  } else if (fart > 66 && fart < 81){
    "Du får et klip. Og tag dig lige sammen."
  } else if (fart > 80 && fart < 101) {
    "Betinget frakendelse."
  } else {
    "En velfortjent, ubetinget frakendelse."
  }
}
```

Funktioner & conditionals i lykkeligt ægteskab

```
strafudmåler(40)
```

```
## [1] "Du er en flink bilist!"
```

```
strafudmåler(66)
```

```
## [1] "Du er en bandit. Men du kan slippe med en bøde!"
```

```
strafudmåler(130)
```

```
## [1] "En velfortjent, ubetinget frakendelse."
```

For-loop

```
# Definerer tibble
tbl <- tibble(a = rnorm(30), b = rnorm(30), c = rnorm(30))

# Skriver loop
means <- vector("double", ncol(tbl))  # output

for (i in seq_along(tbl)){            # sequence
  means[[i]] <- mean(tbl[[i]])        # body
}

# Printer output
means
```

```
## [1] -0.07245496  0.02095043  0.09003708
```

While-loop

```
# Definerer vektor med mandater
mandater <- 0

# Definerer loop
while (mandater < 90){
  print("Stem dørklokker!")
  mandater <- mandater + 1
}
```

```
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
## [1] "Stem dørklokker!"
```

Iteration med `purrr`

- `map()`
- `map_lgl()`
- `map_int()`
- `map_dbl()`
- `map_chr()`
- minder om `apply()`-familien

Iteration med `apply()`

```
# Definerer data frame
flights_small <- flights %>% select(dep_time, dep_delay, arr_time, arr_delay)

# Iteration med lapply()
lapply(flights_small, mean, na.rm = TRUE)
```

```
## $dep_time
## [1] 1349.11
##
## $dep_delay
## [1] 12.63907
##
## $arr_time
## [1] 1502.055
##
## $arr_delay
## [1] 6.895377
```

Iteration med **purrr**

```
# Iteration med vapply()
vapply(flights_small, mean, na.rm = TRUE, FUN.VALUE = numeric(1))
```

```
##    dep_time   dep_delay    arr_time   arr_delay
## 1349.109947   12.639070 1502.054999    6.895377
```

```
# Iteration med map_dbl()
map_dbl(flights_small, mean, na.rm = TRUE)
```

```
##    dep_time   dep_delay    arr_time   arr_delay
## 1349.109947   12.639070 1502.054999    6.895377
```

4. Git, GitHub & version control

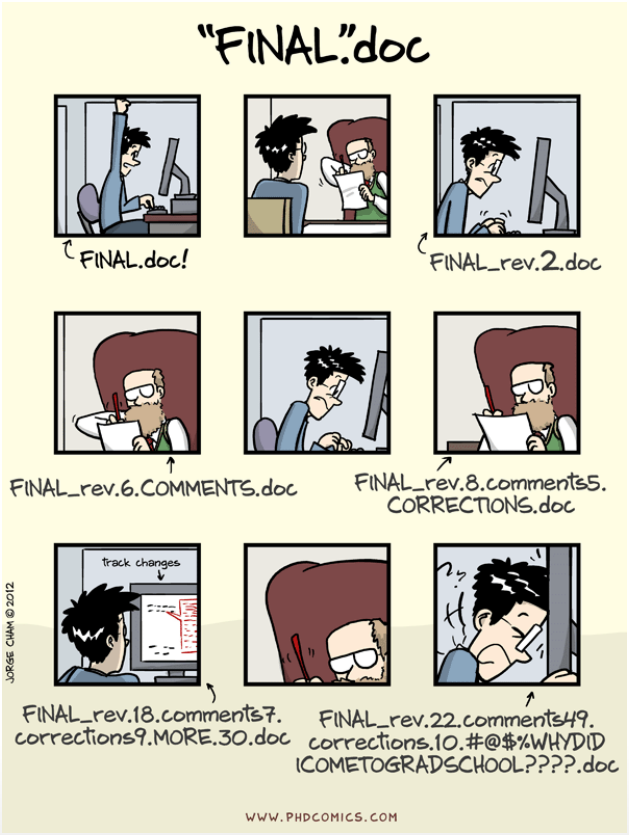
Hvad er version control?

- Snapshots af ændringer i alle filer i et projekt over tid
- Gør samarbejde
  - veldokumenteret
  - sikkert
  - fleksibelt
  - skalerbart

Those who cannot learn from history are doomed to repeat it - maybe because they didn't bother to use version control.

– Santayana & Svejgaard

Hvorfor version control?



Fordele ved version control

- Tracking af ændringer i dokumenter
- Forstå tidligere ændringer af dokumenter
- Finde tilbage til tidligere versioner af et dokument
- Flere kan arbejde på samme dokument samtidig
- Merging af to versioner af det samme dokument og "konflikthåndtering"

Hvorfor ikke bare bruge Google Drive eller Dropbox?

- Mange filtyper, fx .R, .csv, .Rmd
- Fuld historik med kommentarer
- Nyttig og veldokumenteret konflikthåndtering (særligt ift. Dropbox)
- Skalerbart samarbejde

Terminologi

- `Git` = en implementering af version control
- `GitHub` = en virksomhed som tilbyder at opbevare repositories
- `GitHub Desktop` = software med grafisk interface til at bruge Git
- `Repository` = en 'mappe' med filer, fx alle filer i et projekt
- `Branch` = en parallel-version af et repository, hvor man kan eksperimentere
- `Commit` = at 'gemme' ændringer, fx i en branch
- `Publish`/`push` = skubber dine ændringer op på github.com
- `Pull request` = forespørger at dine ændringer bliver en del af 'masteren'
- `Fork` = opret et repo, der er en kopi af en andens repo på github.com
- `Clone` = download en lokal kopi af et repo, som du kan holde synkroniseret
- `Fetch`/`pull` = synkroniseret din lokale klon med et repo på github.com
- GitHub-ordbog

Typisk workflow

1. Lav en branch
2. Lav dine ændringer og commit dem
3. Lav en pull request

Commitment

- En helt central komponent i version control!
- Svarer lidt til "at gemme"
- Mere end det: Det er et snapshot på et givent tidspunkt, som vi altid kan vende tilbage til!
- Man kan (skal) skrive kommentarer til dem, man samarbejder med
- Commit summaries gør historikken brugbar
- Man bør commit'e på meningsfulde punkter i arbejdet (atomic commits)

Pull requests & merging

Koncept:

- Har man arbejdet på en branch og lavet ændringer i et repo, kan man lave en pull request, hvor man laver en forespørgsel på, at de nye ændringer bliver merged med master branch'en

Muligheder:

- Kan være fra én branch til en anden, fx fra en `this-crazy-idea` til `master`
- Kan være fra en forked udgave af et repo til originalen

Konflikthåndtering

- Konflikter er nogen man taler om - saml og hersk!
- Historik om håndteringen af konflikten gemmes

Vi tager den lige igen - typisk workflow

1. Opret eller tilføj et repo
2. Lav en branch
3. Lav dine ændringer og commit dem undervejs
4. Publish/push dine ændringer til github.com
5. Lav en pull request
6. Diskuter ændringer med dem, man samarbejder med
7. Arbejd videre og lav rettelser og push dem til github.com
8. Merge
9. Fetch/pull til din klonede kopi af master branchen
10. Slet branchen, du arbejdere på, når du er færdig

At forstå Git

Eksempel

Flere ressourcer om Git og GitHub

- GitHub On Demand Training
- GitHub Guides
- Hello World guide
- Forking projects
- Video om GitHub Desktop
- Software Carpentry: Version Control with Git
- DataCamp: Introduction to Git for Data Science

5. Workshop

Workshop

1. Log ind på github.com og fork repository'et PDS.
2. Find nu repository'et på din egen profil. Klon og åben det åben i GitHub Desktop.
3. Lav en ny branch i GitHub Desktop.
4. Find repository'et på din computer.
5. Du har nu alle slides, script mm. liggende. Nice!
6. Åben mappen 'opgaver' i repository'et og find filen `04_opgaver.R`
7. Løs opgaverne - og lav et commit i GitHub Desktop for hver gang, du løser en opgave. Husk sigende kommentarer.
8. Når du er færdig med opgaverne, eller der er 10 minutter tilbage af lektionen, så:
   - publish/push dine ændringer fra GitHub Desktop til github.com, og
   - gå ind på dit repo på github.com og opret en pull request, hvor du merger ændringerne i filen `04_opgaver.R` i din branch med master-branchen på github.com
9. Gå tilbage i GitHub Desktop, vælg branchen master og fetch/pull dine ændringer.
10. Enjoy the fame & glory.

6. Opsamling og næste gang

Vigtigste pointer fra i dag

- Pipes
- Conditionals - if, then, else
- Funktioner
- For- og while loops
- Iteration med `purrr`
- Version control \o/

Næste gang

- Indhold:
  - Web scraping & API
  - Workshop: Første spadestik til seminaropgave
- Pensum:
  - Bemærk ændringer!
  - The Economist (2016) - skimmes
  - Shiab (2015) - skimmes
- DataCamp:
  - Working with Web Data in R - fokuser på denne