

Crash course: Geospatial Datavisualisering

Jeppe Vierø

April 25, 2022

1 Introduktion

2 The Basics: Geodata i R

3 Visualisering med {ggplot2}

4 Datakilder

5 Interaktiv visualisering med {tmap}

6 Værktøjer i R

Introduktion

Motivation

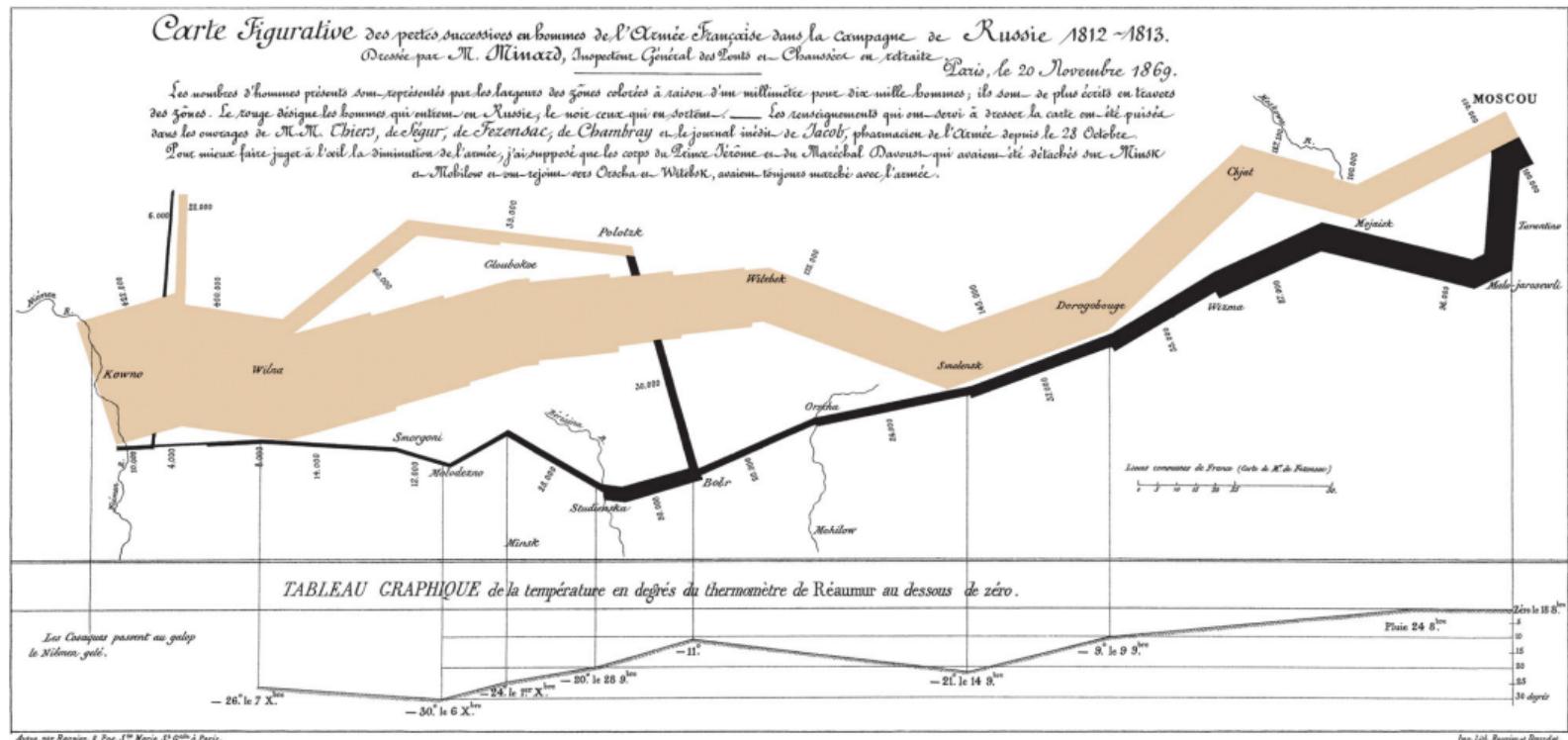


Figure 1: by Charles Joseph Minard, 1869

Afgrænsning

Jeg (regner med) at snakke **en del** om:

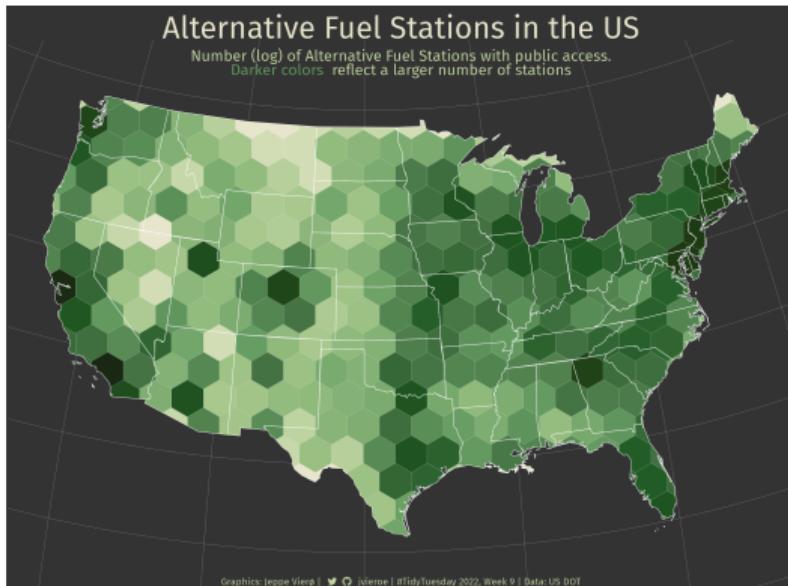
- Hvad **spatial** data er
- Hvordan vi kan bruge spatiale datakilder til at **visualisere** andet data
- Hvordan vi gør det i R

Jeg kommer **ikke** til at snakke (så meget) om:

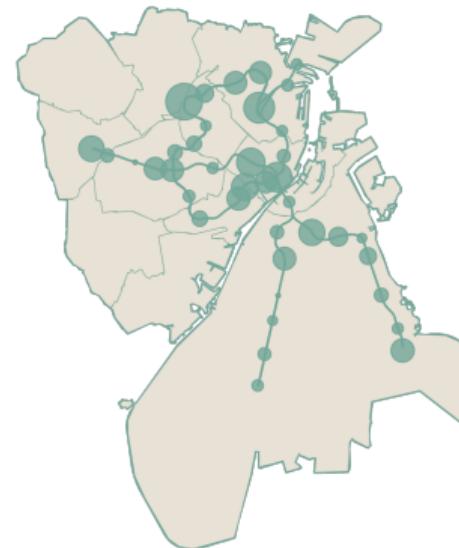
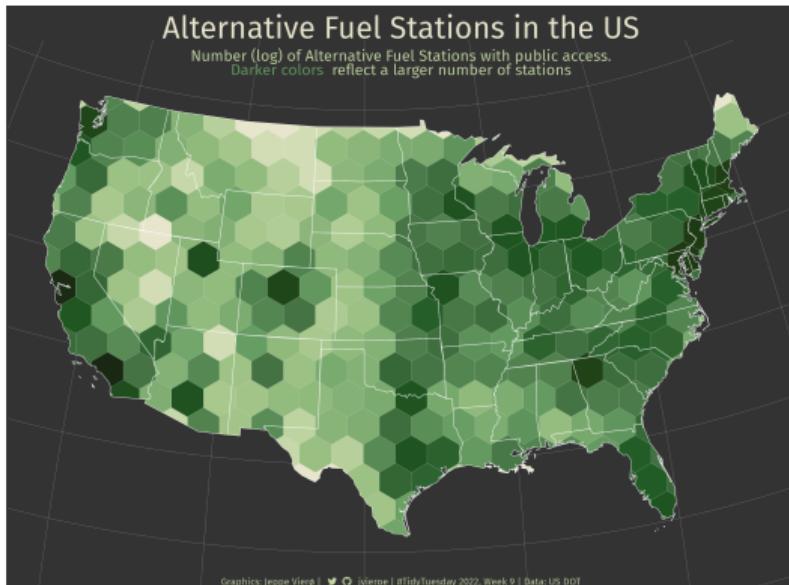
- Datawrangling og -manipulation med geospatial data
- Datavisualisering generelt

Hvordan kan vi bruge geodata til at visualisere vores pointer?

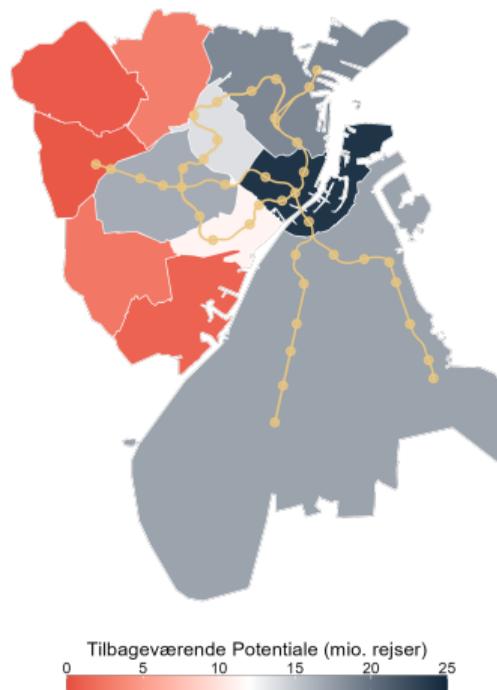
Hvordan kan vi bruge geodata til at visualisere vores pointer?



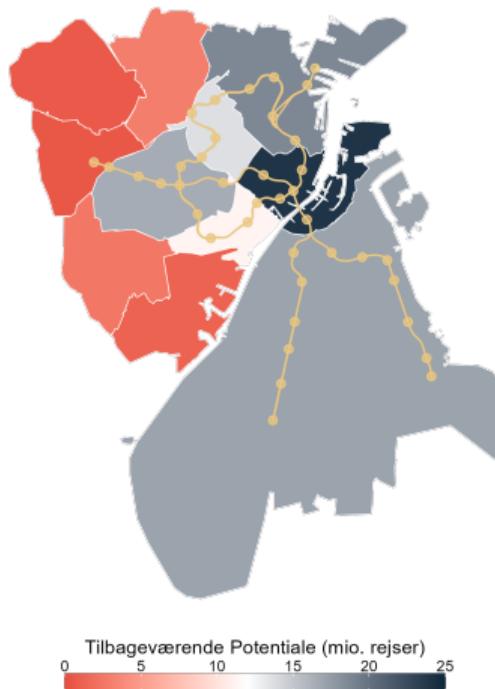
Hvordan kan vi bruge geodata til at visualisere vores pointer?



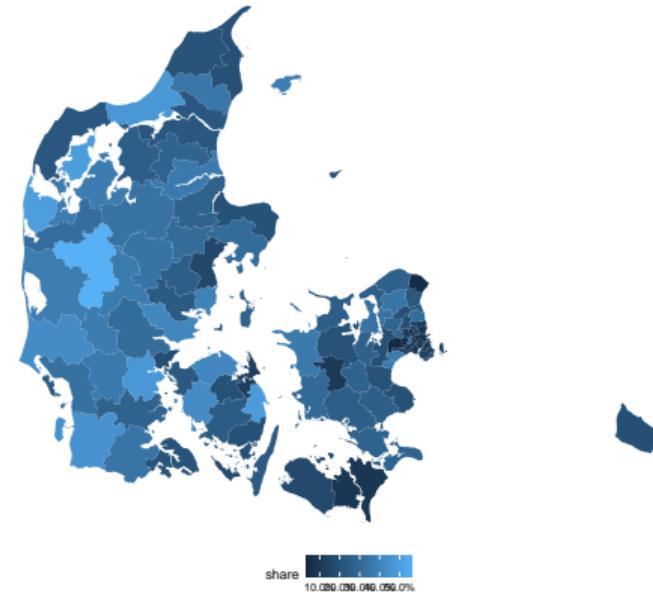
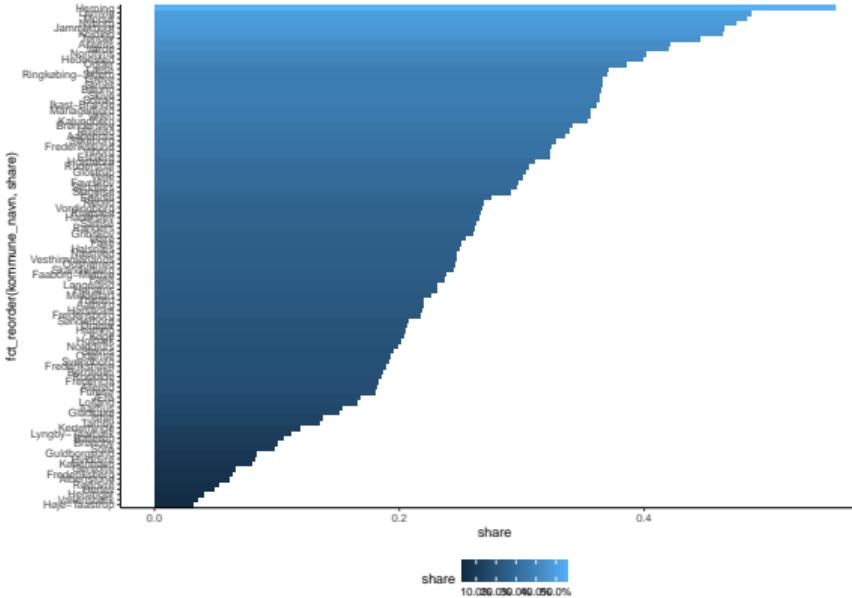
Hvordan kan vi bruge geodata til at visualisere vores pointer?



Hvordan kan vi bruge geodata til at visualisere vores pointer?



Hvorfor ikke bare bruge “klassiske” visualiseringer? (I)



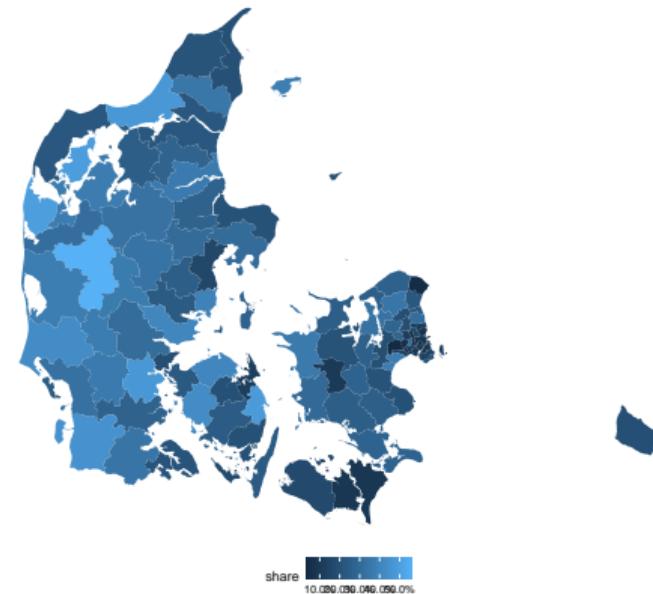
Hvorfor ikke bare bruge “klassiske” visualiseringer? (II)

Typisk kan man med fordel (overveje at) visualisere sit data grafisk, hvis

- Der er nogle **substantielle geografiske mønstre** i data, der er interessante (case in point:)

eller,

- Det data, vi gerne vil visualisere fundamentalt set har en geografisk dimension



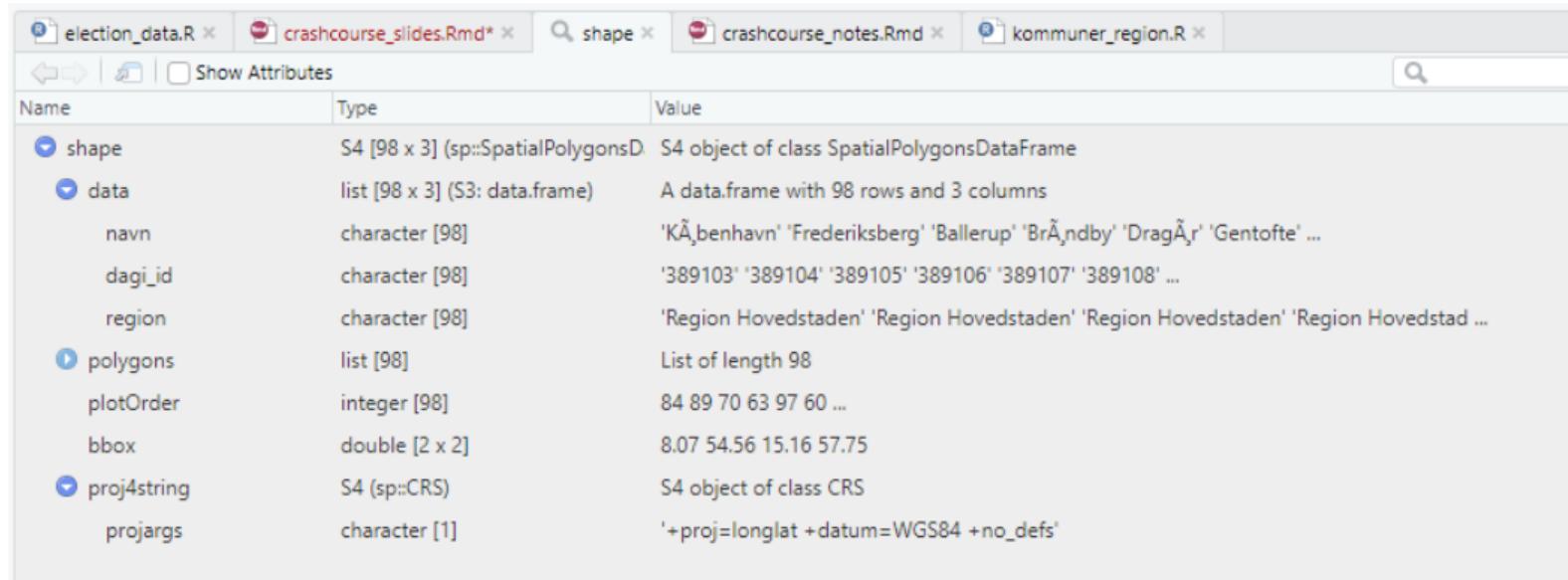
Kan man gøre det hele *endnu* nemmere?

- Der er lavet et par R-pakker, der spytter ready-made kort ud for dig
- `{mapDK}` er efterhånden fem år gammel og er ikke rigtigt blevet opdateret. Den er et no-go
- `{plotDK}` blev lanceret i 2021. Den kan meget af det samme, men er udgivet på CRAN og er up to date.
- Min anke med de pakker er primært, at
 - De (meget!) hurtigt bliver spændetrøjer ift. hvad man kan visualisere og hvordan
 - Det faktisk er rigtig nemt at gøre det selv!
- Det er så det, vi er her for at snakke lidt om i dag!

The Basics: Geodata i R

A Blast from the past: {sp}

- Det har tidligere været relativt besværligt at arbejde med spatialt data i R
- sp-pakken var det førende framework, men selv simple datasæt var... irriterende:



The screenshot shows the RStudio interface with several tabs open at the top: election_data.R, crashcourse_slides.Rmd*, shape, crashcourse_notes.Rmd, and kommuner_region.R. The 'shape' tab is active, displaying a data frame with 98 rows and 3 columns. The table has three columns: Name, Type, and Value.

Name	Type	Value
shape	S4 [98 x 3] (sp::SpatialPolygonsDataFrame)	S4 object of class SpatialPolygonsDataFrame
data	list [98 x 3] (S3: data.frame)	A data.frame with 98 rows and 3 columns
navn	character [98]	'KÃ¸benhavn' 'Frederiksberg' 'Ballerup' 'BrÃ¸ndby' 'DragÃ¸r' 'Gentofte' ...
dagi_id	character [98]	'389103' '389104' '389105' '389106' '389107' '389108' ...
region	character [98]	'Region Hovedstaden' 'Region Hovedstaden' 'Region Hovedstaden' 'Region Hovedstad ...
polygons	list [98]	List of length 98
plotOrder	integer [98]	84 89 70 63 97 60 ...
bbox	double [2 x 2]	8.07 54.56 15.16 57.75
proj4string	S4 (sp::CRS)	S4 object of class CRS
projargs	character [1]	'+proj=longlat +datum=WGS84 +no_defs'

Din nye bedste ven: {sf}

- Med sf er det blevet **smooth sailing**. Den måde, pakken håndterer det *spatiale* aspekt af et datasæt gør, at det ligner alle andre datasæt til forveksling:
- Magien ligger i geometry-listen til sidst. Alt (!) andet er data frames/tibbles, som I kender dem

```
library(tidyverse)
library(sf)

df <- st_read(dsn = "data/kommuner",
               layer = "kommuner")

df

## Simple feature collection with 98 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 8.07251 ymin: 54.55908 xmax: 15.15738 ymax: 57.75257
## Geodetic CRS:  WGS 84
## First 10 features:
##      navn dage_id      region           geometry
## 1   København 389103 Region Hovedstaden MULTIPOLYGON (((12.54502 55...
## 2 Frederiksberg 389104 Region Hovedstaden MULTIPOLYGON (((12.53735 55...
## 3   Ballerup 389105 Region Hovedstaden MULTIPOLYGON (((12.3423 55....
## 4    Brøndby 389106 Region Hovedstaden MULTIPOLYGON (((12.44279 55...
## 5   Dragør 389107 Region Hovedstaden MULTIPOLYGON (((12.64513 55...
## 6  Gentofte 389108 Region Hovedstaden MULTIPOLYGON (((12.59175 55...
## 7   Gladsaxe 389109 Region Hovedstaden MULTIPOLYGON (((12.47771 55...
## 8   Glostrup 389110 Region Hovedstaden MULTIPOLYGON (((12.41842 55...
## 9    Herlev 389111 Region Hovedstaden MULTIPOLYGON (((12.40838 55...
## 10 Albertslund 389112 Region Hovedstaden MULTIPOLYGON (((12.36431 55...
```

Din nye bedste ven: {sf}

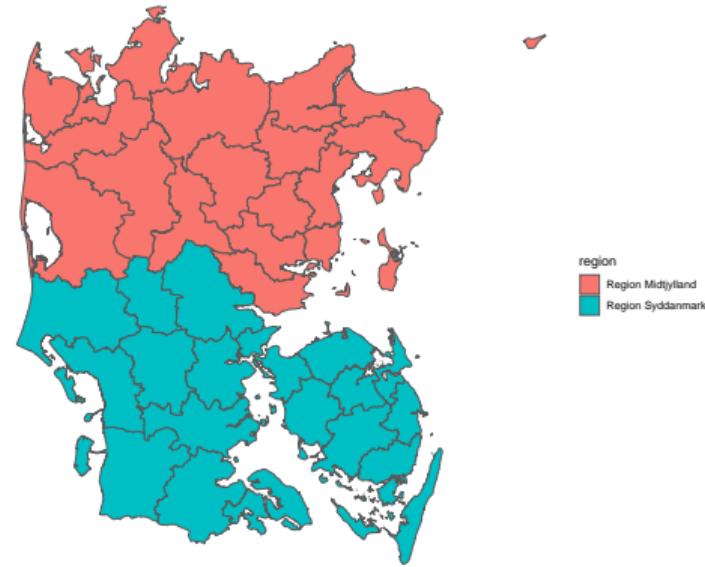
Helt konkret giver sf os mulighed for at bruge simple tidyverse-funktioner til at:

- ① arbejde med data (dplyr, tidyr osv.)
- ② visualisere data! (ggplot2)

Lad os prøve begge dele!

```
# data wrangling med tidyverse (dplyr):
df <- df %>%
  filter(region %in% c("Region Syddanmark", "Region Midtjylland"))

# data viz med tidyverse (ggplot2):
ggplot() +
  geom_sf(data = df, aes(fill = region)) +
  theme_void()
```



Datastrukturer og typer af geodata

Grundlæggende arbejder vi med **tre typer af geospationale datakilder**

Hver type har en (nogenlunde) parallel til graftyper, I er vant til at arbejde med:

① Punkter

- Tænk på dem som almindelige *punkter i et scatterplot*

② Linjer

- Tænk på dem som *linjer i et linechart*

③ Polygoner

- Her er parallelen ikke lige så tydelig
- ... men i en data viz-kontekst kan I tænke på dem som *søjler i et bar chart* (ish...)

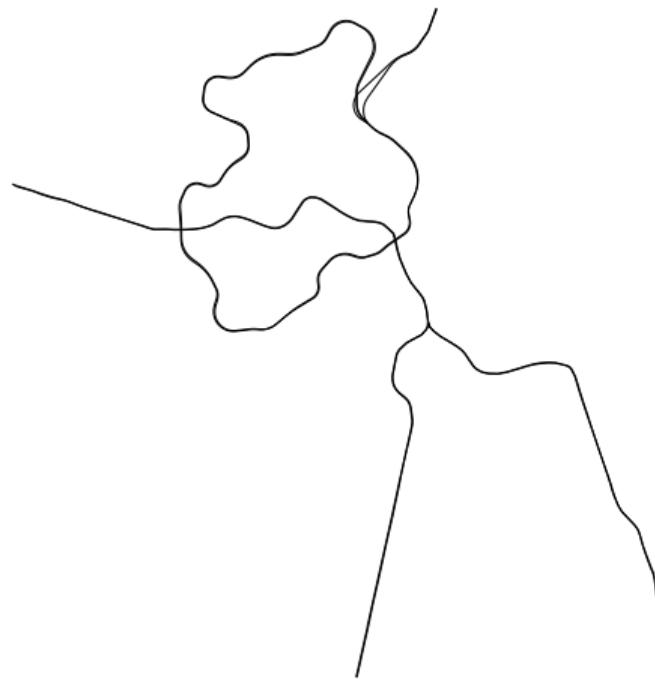
(1) Punkter

- Punkter består af simple koordinater (x, y), der refererer til en specifik lokation
- Punkter har ingen størrelse (og intet *areal*), de er uendeligt små
- Eksempler: byer, stationer, skoler osv.



(2) Linjer

- Linjer består – grundlæggende – af punkter, der er kombineret til en *linestring* vha. en defineret rækkefølge
- Konstruktionen er sjældent noget, I skal bekymre jer om: linjedata ligger typisk opbevaret som linjer (\neq punkter). Her er det bare plug 'n play
- Linjer har intet *areal* (fordi de består af punkter)
- Eksempler: veje, floder, jernbanenetværk osv.



(3) Polygoner

- Polygoner består – ligesom linjer – af punkter, der er kombineret til en *polygon* vha. en defineret rækkefølge. Ingen, det er sjældent noget, I skal bekymre jer om
- Forskellen er, at polygoner er *lukkede linjer*, der former et afgrænset område
- De kan have alle tænkelige former. Det centrale er, at polygoner har et *areal*
- Eksempler: stater, kommuner, valgkredse osv.



Visualisering med {ggplot2}

Visualisering med {ggplot2}

XXX

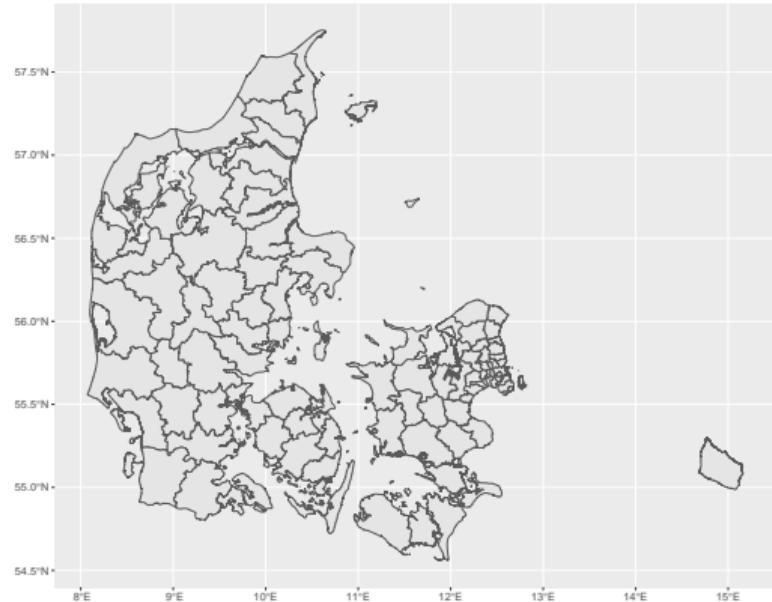
```
vshare <- st_read(dsn = "data/kommuner_98",
                    layer = "kommuner_98")

head(vshare)

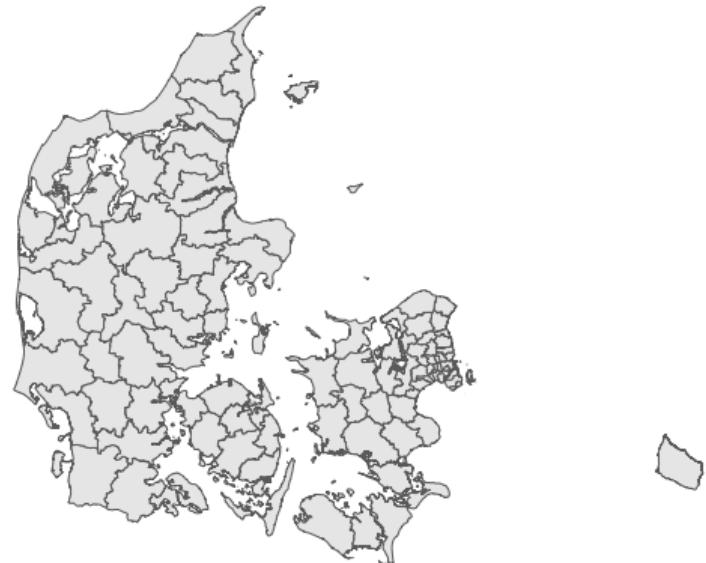
## Simple feature collection with 6 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: 12.2635 ymin: 55.53633 xmax: 12.73425 ymax: 55.77944
## Geodetic CRS: WGS 84
##   dagi_id      navn kommune_nr      region party  total    share
## 1 389103 København      101 Region Hovedstaden 23652 300216 0.07878328
## 2 389104 Frederiksberg      147 Region Hovedstaden 3748 59298 0.06320618
## 3 389105 Ballerup      151 Region Hovedstaden 2739 25949 0.10555320
## 4 389106 Brøndby      153 Region Hovedstaden 1689 16921 0.09981680
## 5 389107 Dragør      155 Region Hovedstaden 1732 8403 0.20611686
## 6 389108 Gentofte      157 Region Hovedstaden 2649 40430 0.06552065
##
##   geometry
## 1 MULTIPOLYGON (((12.54502 55...
## 2 MULTIPOLYGON (((12.53735 55...
## 3 MULTIPOLYGON (((12.34223 55....
## 4 MULTIPOLYGON (((12.44279 55...
## 5 MULTIPOLYGON (((12.64513 55...
## 6 MULTIPOLYGON (((12.59175 55...
```

Visualisering med {ggplot2}

```
ggplot() +  
  geom_sf(data = vshare)
```

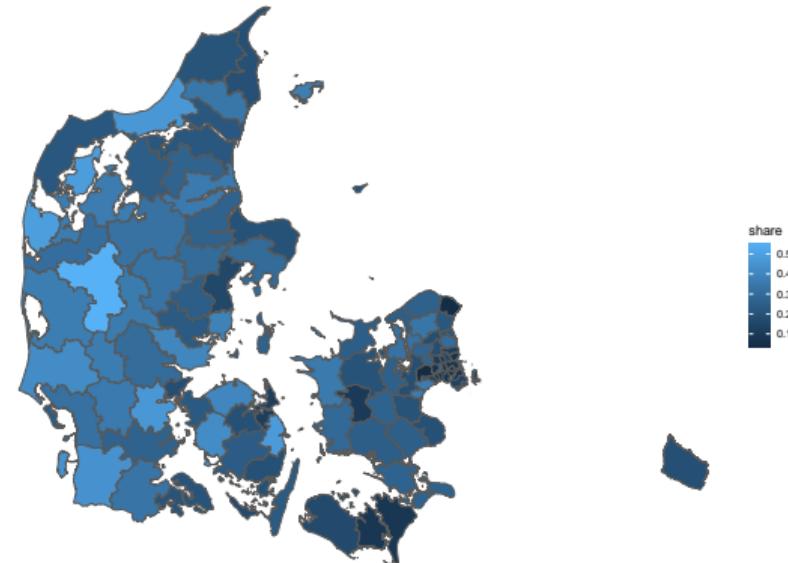


Visualisering med {ggplot2}



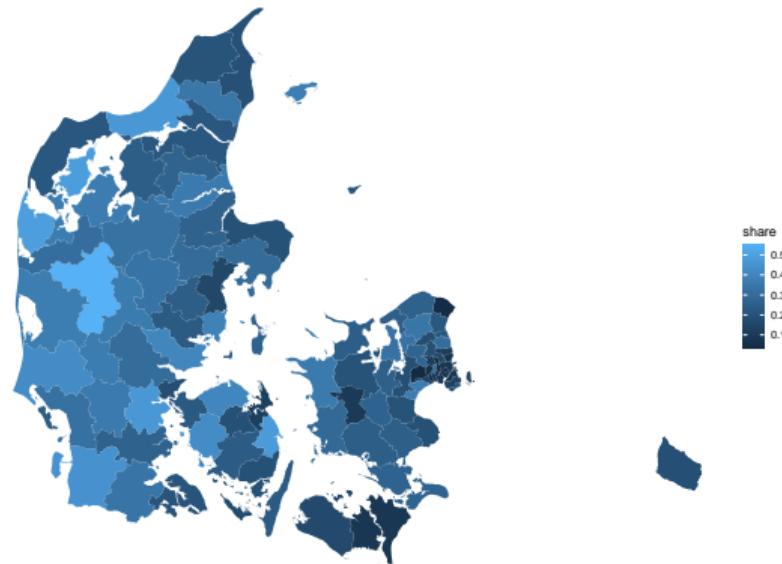
```
ggplot() +  
  geom_sf(data = vshare) +  
  theme_void()
```

Visualisering med {ggplot2}



```
ggplot() +  
  geom_sf(data = vshare,  
          aes(fill = share)) +  
  theme_void()
```

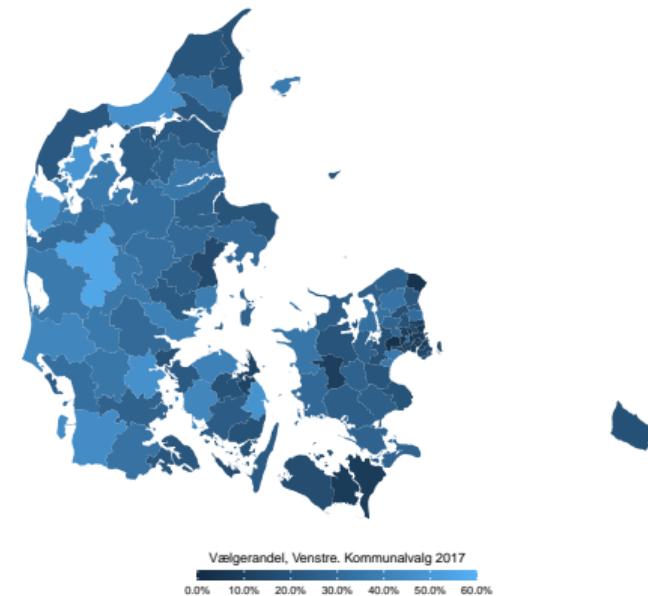
Visualisering med {ggplot2}



```
ggplot() +  
  geom_sf(data = vshare,  
          aes(fill = share),  
          color = "transparent") +  
  theme_void()
```

Visualisering med {ggplot2}

```
ggplot() +  
  geom_sf(data = vshare,  
           aes(fill = share),  
           color = "transparent") +  
  scale_fill_continuous(name = "Vægerandel, Venstre. Kommunalvalg 2017",  
                        labels = scales::percent,  
                        breaks = seq(0, 0.6, .1),  
                        limits = c(0, .6)) +  
  theme_void() +  
  theme(legend.position = "bottom") +  
  guides(fill = guide_colorbar(title.position = "top",  
                               title.hjust = .5,  
                               barwidth = 16,  
                               barheight = .5))
```



Datakilder

DAGI

- “*Danmarks Administrative Geografiske Inddeling (DAGI)* beskriver landets administrative og geografiske inddeling i kommuner, regioner, sogne, retskredse, politikredse, postnumre, opstillingskredse og lignende.” – [DAWA](#)
- ... med andre ord; alt hvad vi kunne drømme om
- DAGI-data kan hentes via Styrelsen for Dataforsyning og Effektiviseringens [Datafordeler](#)
- Det er en ret håbløs hjemmeside, til gengæld er der masser at vælge mellem (inkl. historiske enheder!)



Styrelsen for
Dataforsyning og
Effektivisering

DAGI

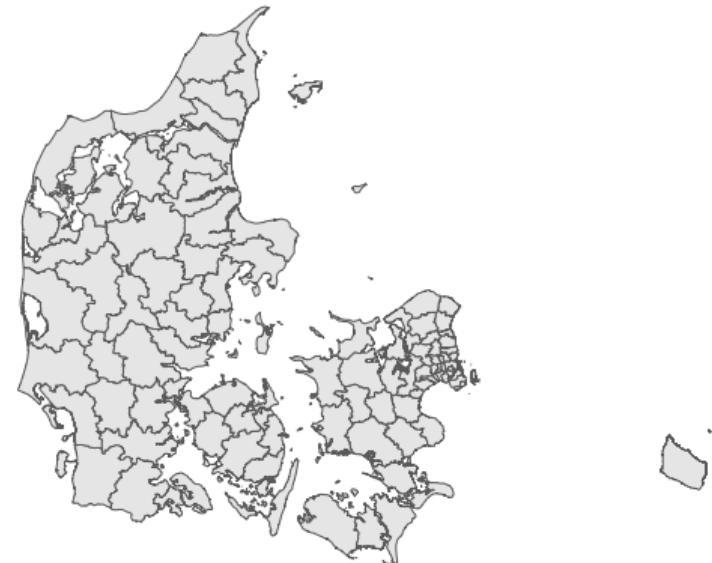
- Til de fleste formål kan vi hoppe uden om Datafordelen ved at bruge [DAWA \(Danmarks Adressers Web API\)](#) og den tilhørende [API](#)
- API'en er plug 'n play, hvor vi kan vælge de [enheder](#), vi skal bruge, og specificere [format](#):
- <https://api.dataforsyningen.dk/\protect\unhbox\voidb@x{\color{purple}kommuner}?format=\protect\unhbox\voidb@x{\color{teal}geojson}>

DAGI: et eksempel

```
# definér data
url <-
  "https://api.dataforsyningen.dk/kommuner?format=geojson"

# indlæs data
kommuner_raw <-
  read_sf(url)

# plot data
ggplot() +
  geom_sf(data = kommuner_raw) +
  theme_void()
```



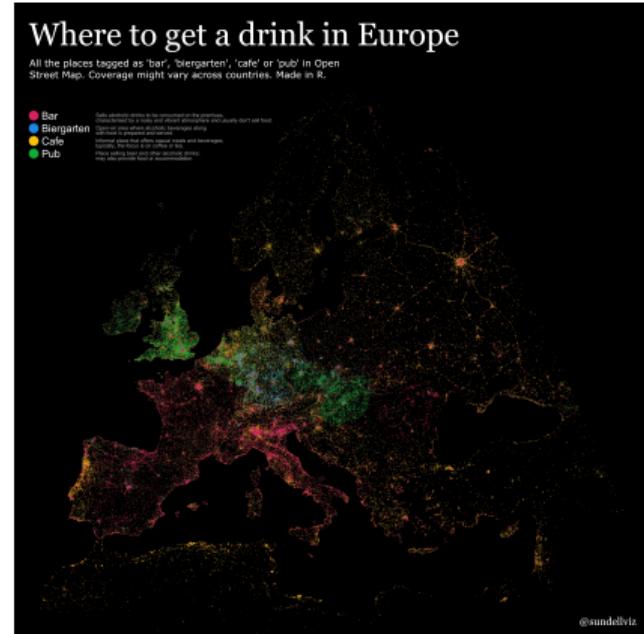
OpenStreetMap

- OpenStreetMap (OSM) er en crowd sourced geografisk database med detaljeret information om hele verden
- OSM indeholder data på (næsten) alt, hvad hjertet begærer
- OSM har en tilhørende [wiki](#), med en oversigt over de forskellige features



OpenStreetMap

- Vi kan bruge R-pakken `{osmdata}` til at hente OSM-data direkte i R
- Her skal vi bruge
 - En geografisk afgrænsning
 - Valg af features (vha. argumenterne key og value):



OpenStreetMap: et eksempel

```

library(osmdata)

# Hent OSM-data for Vejle
vejle <- kommuner_raw %>%
  filter(navn == "Vejle")

vejle_bbox <- vejle %>%
  st_bbox()

osm <- vejle_bbox %>%
  opq()

# Hent udvalgte veje
roads <- osm %>%
  add_osm_feature(key = 'highway',
                  value = c('motorway', 'trunk',
                           'primary', 'secondary',
                           'tertiary')) %>%
  osmdata_sf()

# Besør
roads <- roads$osm_lines %>%
  st_intersection(., vejle)

# Plot vejene
ggplot() +
  geom_sf(data = vejle, fill = "white", linetype = "dashed") +
  geom_sf(data = roads, aes(color = as.numeric(maxspeed)),
          size = 1) +
  scale_color_viridis_c(direction = -1, name = "Speed limit (km/h)") +
  theme_void()

```

Speed limit (km/h)

Color	Speed Limit (km/h)
Dark Purple	125
Medium-Dark Purple	100
Green	75
Light Green/Yellow	50

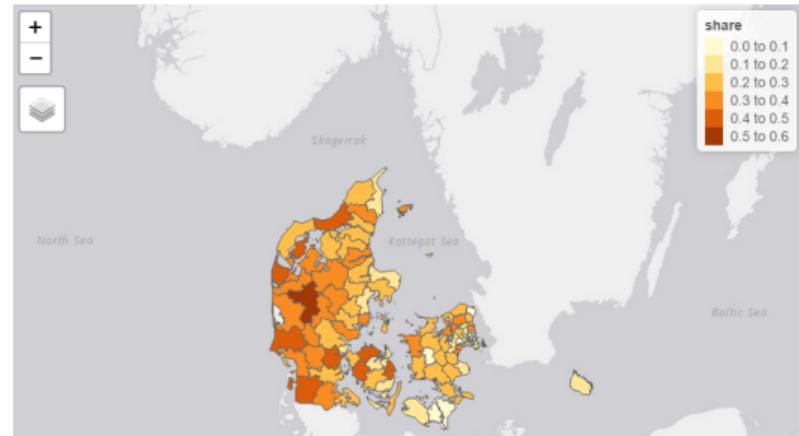
Interaktiv visualisering med {tmap}

Interaktiv visualisering med {tmap}

- tmap er nyeste skud på stammen, når det kommer til at visualisere geospatial data. I modsætning til ggplot2 er pakken udviklet *specifikt* til dette formål
- tmap-pakken er ikke helt lige så fleksibel som ggplot2. Der skal lige mere arbejde til for at gøre dit plot pænt. Derudover minder syntaxen meget om
- Til gengæld er tmap eminent til at generere **interaktive kort!**
- Af samme grund bruger jeg den ofte, når jeg arbejder *med* data. Det gør det nemt at inspicere dine datasæt

Interaktiv visualisering med {tmap}

```
library(tmap)  
  
tmap_mode("view")  
  
tm_shape(vshare) +  
  tm_polygons(col = "share")
```



Værktøjer i R

text

xx