Refresher data prep & interactive data visualiatie

Trainee Sessie 01

Longhow Lam

Contents

The tidyverse **1**

De dplyr library **2**
kolommen selecteren met select . . . 2
kolom maken of wijzigen met mutate . . . 2
distinct rows . . . 4
data sets filteren . . . 4
data sets aggregeren . . . 4
joins . . . 5
data frames stapelen of plakken . . . 5

Tidy data en wide data **6**

Character data bewerken met het package `stringr`. **7**
Reguliere expressies . . . 7
functies uit het package stringr . . . 8

Factor maniuplatie met forcats **8**

Dates in R **9**

dplyr met externe data **10**

Interactive grafieken **11**
Het plotly package . . . 11
Kaartjes met leaflet . . . 13
Netwerk grafieken, visnetwork . . . 16

Zoomable circle packing plots, sunburst plots en chord diagrams **18**

The tidyverse

Het `tidyverse` package is een verzameling R packages met 'dezelfde' gedachtegoed om data te bewerken / manipuleren. Installeer tidyverse en attach de library zodat je in 1 keer een aantal libraries laadt:

- dplyr
- tibble
- ggplot2
- purrr
- forcats
- readr
- readxl

- stringr
- tidyr

De dplyr library

Dit package / library is ontzettend handig om data te bewerken in R. De syntax is elegant en dplyr code kan je in sommige gevallen ook gebruiken voor data die niet in R zit. Bijvoorbeeld, data in Spark kan je met dplyr code bewerken. Er zijn veel 'sleutel' woorden, maar de belangrijkste zijn `select`, `filter`, `mutate`, `arrange`, `summarize`, `slice` en `rename`.

Het mooie is dat deze sleutel woorden ook achter elkaar gebruikt kunnen worden met de `%>%` chain operator. We gebruiken als voorbeeld data uit de datasets library. `mtcars` is een klein data set met wat auto gegevens en `iris` is de bekende data set met drie soorten bloemen.

kolommen selecteren met select

```
## enkele kolommen selecteren
my.cars = select(mtcars, hp, mpg, drat)

## via de chain operator, dit is niet alleen handig maar zo kan je verschillende operaties achter elkaa
my.cars = mtcars %>%
  select(
    hp,
    mpg,
    drat
  )

## meerdere kolommen selecteren
test1 = iris %>%
  select(
    starts_with("Petal")
  )

## bepaalde kolommen selecteren
test2 = mtcars %>%
  select(
    contains("pg"),
    starts_with("c")
  )

## kolommen selecteren op basis van positie
my.cars = mtcars %>% .[2:6]
```

kolom maken of wijzigen met mutate

We zagen al dat je kolommen in een data frame kon maken of wijzigen met `$`

```
my.cars$kolom2  = rnorm(32)
my.cars$kolom3  = my.cars$kolom2 + 9
```

Maar met `mutate` kan je dit veel eleganter doen

```
mycars = mtcars %>%
  select(
    disp,
    mpg
  ) %>%
  mutate(
    displ_l = disp / 61.0237,
    tmp = mpg *1000
  )
```

Je kan nieuwe kolommen maken die afhankelijk zijn van kolommen die je net in mutate maakt.

```
mycars = mtcars %>%
  mutate(
    displ_l = disp / 61.0237,
    tmp = mpg *1000,
    tmp2 = tmp + 1
  )
```

Je kan ook zelf gemaakte functies gebruiken.

```
normalize <- function(x){
    return((x- mean(x))/(max(x)-min(x)))
}

mycars = mtcars %>%
  mutate(
    cyln = normalize(cyl)
  )
```

Twee functies die ontzettend handig kunnen zijn: `mutate_at` en `mutate_if`.

```
ABT = data.frame(
  matrix(round(100*runif(100)), ncol=10)
)

## maak eerst een kolom die de som is van de de 10 kolommen
ABT = ABT %>% mutate(sum = rowSums(.[1:10]))

## verander alleen kolom 4 t/m 8, ze worden door sum gedeeld
ABT2 = ABT %>%
  mutate_at(
    4:8, funs(. / sum)
  )

## deel alleen door de sum als de kolom som groter is dan 500
LARGE = function(x) {sum(x) > 500}
ABT3 = ABT %>%
  mutate_if(
    LARGE, funs(. / sum)
  )

ABT
ABT2
ABT3
```

distinct rows

Soms wil je data ontdubbelen, dit kan je in R doen met de functie `distinct`

```
distinct( select(mtcars, carb, gear))

distinct(select(mtcars, 10,11))

### pipe / chain operatie
myset = mtcars %>%
  select(10,11) %>%
  distinct
```

data sets filteren

Met `filter` en `slice` kan je rijen uit een data frame halen.

Met `slice` kan je op basis van row numbers data selecteren. Als voorbeeld, de eerste 8 records, of record 8 tot en met de laatste record.

```
slice(mtcars, 1:8)
slice(mtcars, 8:n())
```

data sets aggregeren

Aggregeren met de functie `group_by` en `summarise` deze gaan vaak hand in hand samen.

```
## apart aanroep van group_by en summarise... zo kan je zien wat group_by oplevert
by_cyl = group_by(mtcars, cyl)
class(by_cyl)

summarise(by_cyl, Naampje1 = mean(disp), Naampje2 = mean(hp))


## maar vaak doe je de twee in 1 run samen
out = mtcars %>%
  group_by(
    cyl
  ) %>%
  summarise(
    mdisp = mean(disp),
    mhp = mean(hp)
  )


out = filter(
  mtcars, mpg > 11
  ) %>%
  group_by(
    cyl,
    carb
  ) %>%
  summarise(
    N = n(),
```

```
    MeanDisp = mean(disp),
    SD_HP = sd(hp)
  )
```

joins

Met dplyr kun je makkelijk tabellen joinen. Er zijn verschillende joins die ondersteunt worden. We geven hier een aantal voorbeelden.

```
### maak twee data frames aan
df1 = data.frame(
  col1 = c(1,2,3,4,5), tt = rnorm(5)
)

df2 = data.frame(
  col1 = c(3,4,5,6,7), xx = rnorm(5), zz = runif(5)
)

## selecteer rijen die zowel in df1 en df2 zitten
df3 = inner_join(
  df1,
  df2,
  by = c("col1" = "col1")
)

## alle rijen die in df1 zitten, geen match levert NA op
df4 = left_join(
  df1,
  df2,
  by = c("col1" = "col1")
)

## alle rijen die in df2 zitten
df5 = df1 %>% right_join(
  df2,
  by = c("col1" = "col1")
)

## alleen de rijen van df1 die niet in df 2 zitten
df6 = df1 %>% anti_join(
  df2,
  by = c("col1" = "col1")
)

## alle rijen van df1 en df2
full_join(df1, df2, by = c("col1" = "col1"))
```

data frames stapelen of plakken

Soms wil je twee tabellen op elkaar stapelen tot een nieuwe tabel, dat kan je doen met `bind_rows`. Als je twee tabellen naast elkaar wilt zetten tot een nieuwe tabel, gebruik dan `bind_rows`.

```
## bind_rows
A = data.frame(x=1:5, y = rnorm(5))
B = data.frame(x=11:15, y = rnorm(5))
C = bind_rows(A,B)

## kolommen die niet in een van de data frames zitten worden aangevuld
E = data.frame(x=21:25, y = rnorm(5), z = rnorm(5))

bind_rows(A,E)
```

```
A = data.frame(x=1:5, y = rnorm(5))
B = data.frame(q=11:15, q = rnorm(5))
bind_cols(A,B)

## als een van de data frames meer rijen heeft, dan wordt er NIET aangevuld
A = data.frame(x=1:5, y = rnorm(5))
B = data.frame(q=11:17, q = rnorm(7))
bind_cols(A,B)
```

Tidy data en wide data

Tidy data is wanneer data in de volgende vorm zit:

- Elke variabele is in een kolom.
- Elke observatie is een rij.
- Elke waarde is een cel.

Stel we hebben de volgende data set

```
library(tidyr)
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
stocks
```

Soms is het handig om data waarden in 1 kolom te hebben en een aparte variabele die de variabele weergeeft

```
stocksm <- stocks %>% gather(stock, price, -time)
stocksm
```

en spread is het omgekeerde proces

```
stock2 = stocksm %>% spread(stock, price)
stock2

## als er niet even veel observaties zijn?

test = data.frame(
  T = c(1,2,3,1,2),
  V = c("A","A","A","B","B"),
  price = c(4,5,6,7,8)
```

```
)

test2 = test %>% spread(V,price)
test2
```

Een handige functie in tidyr is **separate**, dit kan je gebruiken om een kolom uit elkaar te trekken. Standaard worden niet alfa numerieke tekens als separator gebruikt.

```
df = tibble(x = c("A.B", "C.P", "P.Q"))
df %>% separate(x, c("Kol1", "Kol2"))

## Er kunnen tw weinig of teveel kolommen zijn.
df = tibble(x = c("A.B", "C.P", "P.Q", "pp", "P.2.4"))
df %>% separate(x, c("Kol1", "Kol2"))


df = tibble(x = c("A.B", "C.P", "P.Q", "pp", "P.2.4"))
df %>% separate(x, c("Kol1", "Kol2", "Kol3"))
```

Character data bewerken met het package **stringr.**

Character data in R kan je bewerken/manipuleren met het **stringr** package. Voordat we daar verder op in gaan is het handig om te weten wat reguliere expressies zijn.

Reguliere expressies

Dit is een soort 'mini' taaltje om character patronen te specificeren om er vervolgens bijvoorbeeld op te zoeken. Eerst wat dummy character data.

```
test = c("Mijn nummer is 06-12345678", "dit is .. een 1628EP postcode test", "foutje?:  0234XD", "dit is
test
```

Een paar voorbeelden van reguliere expressies.

```
library(stringr)

## een digit, en precies 1 digit
patroon = "\\d"
str_extract(test, patroon)

## 1 of meerdere digits
patroon = "\\d+"
str_extract(test, patroon)

## precies twee letters
patroon ="[[:alpha:]]{2}"
str_extract(test, patroon)

## Een postcode
patroon = "\\d{4}\\s?[[:alpha:]]{2}"
str_extract(test, patroon)
```

```
## Maar een postcode begint niet met een 0
patroon = "[1-9]\\d{3}\\s?[[:alpha:]]{2}"
str_extract(test, patroon)

## special punctuation characters !"#$%&'()*+,-./:;<=>?@[]^_`{|}~
patroon = "[[:punct:]]"
str_extract(test, patroon)
```

Sommige mensen zijn de wildcard notatie gewend om te zoeken in strings, deze wildcard notatie kan je vertalen naar reguliere expressies met `glob2rx`. Een paar voorbeelden zullen hieronder geven.

```
patroon = glob2rx("*23*")
str_extract(test, patroon)
```

Voor een cheatsheet over reguliere expressies zie regex cheatsheet

functies uit het package stringr

In het package stringr zijn diverse functies om met characters te werken, een aantal zullen we laten zien

```
testset = titanic::titanic_train %>% select(Name)
## maak extra kolom aan in titanic die namen zoekt
testset = testset %>%
  mutate(
    naamlengte = str_length(Name),
    Ownes = str_detect(Name, 'Owen'),
    Plek = str_locate(testset$Name, "Mr") %>% .[,1],
    Name2 = str_replace(Name, "Mr", "")
  )

str_locate(testset$Name, "Mr")
```

Factor maniuplatie met forcats

In R kun je factor variabelen bewerken met functies uit het `forcats` package. We geven een aantal voorbeelden op random gegenereerde data

```
library(forcats)

x = rnorm(1000)
y = sample(LETTERS[1:10], replace = TRUE, size = 1000, prob = (1:10)^2)

test = tibble(gewicht = x, type = y)
table(test$type)
```

Samenvoegen van levels: Op aantallen (Weinig voorkomende levels) of handmatig

```
test = test %>%
  mutate(
    z = fct_lump(type, n=5)
  )
```

```r
table(test$z)

test = test %>%
  mutate(
    z2 = fct_other(type, keep = c("C","B")),
    z3 = fct_other(type, drop = c("A","B"))
  )

table(test$z2)
table(test$z3)
```

Hernoemen van factor levels

```r
test = test %>%
  mutate(
    z2 = fct_recode(
      type,
      ZZ = "A",
      ZZ = "B",
      YY = "C",
      YY = "D"
    ),
    z3 = fct_collapse(
      type,
      missing = c("B", "C"),
      other = "D",
      rep = c("E", "F"),
      ind = c("G","H"),
      dem = c("I","J")
    )
  )
table(test$z2)
table(test$z3)
```

Herordenen van factor levels, factor levels kunnen een order bevatten, deze kan je maken of veranderen, kan soms nodig zijn bij plotten.

```r
test = test %>%
  mutate(
    z2 = fct_reorder(type, gewicht, mean)
  )

table(test$type)
table(test$z2)

levels(test$type)
levels(test$z2)
```

Dates in R

Voor datums (en tijden) in R zijn de packages `anytime` en `lubridate` heel handig. Laten we er van uitgaan

dat we characters hebben met datums en daar mee willen rekenen. Dan kunnen we de functie `anydate` uit het anytime package gebruiken.

```
y = c("1973-09-12", "1980-05-23", "1981-12-09")

testdata = tibble(DoB = y)

testdata = testdata %>%
  mutate(
    GeboorteDatum = anytime::anydate(DoB)
  )
```

De functie `anydate` parsed diverse character notatie naar een date kolom, de gaat meestal goed :-)

```
y = c("1973/09/12", "05/23/1980", "23/05/1980","1981-12-09")

testdata = tibble(DoB = y)

testdata = testdata %>%
  mutate(
    GeboorteDatum = anytime::anydate(DoB)
  )
```

Nu we een datum echt als date column in een data frame of tibble hebben kunnen we er mee rekenen.

```
testdata = testdata %>%
  mutate(
    leeftijd = today() - GeboorteDatum,
    leeftijd2 = as.numeric(today() - GeboorteDatum)/365,
    dag = wday(GeboorteDatum, label=TRUE)
  )
```

UNIX time stamps.... Wat zeg je?

Een Unix time stamps is het aantal seconden sinds 1-1-1970. Mocht je dit ooit tegen komen dan kan anytime er ook mee uit de voeten.

```
tmp = read_csv("data/unixtimestamps.csv")
tmp$datum = anydate(tmp$timestamp)
tmp
```

dplyr met externe data

Je kan `dplyr` niet alleen gebruiken op data frames in R, ook kan je `dplyr` gebruiken op externe data in databases en, ook data in Spark. We geven hier eerst het Spark voorbeeld met het package `sparklyr`, en dan een MySQL voorbeeldje.

```
library(sparklyr)
library(dplyr)

## maak een connectie naar spark
sc <- spark_connect(master = "local")

## upload data naar spark, zie ook connections tab
iris_tbl <- copy_to(sc, iris)
```

```
## nu kan je dplyr gebruiken op spark
test = iris_tbl %>% filter(Sepal_Length >5)

class(test)
test %>% count()

spark_disconnect(sc)
```

Als je MySQL hebt en een odbc connectie hebt opgezet kan je een link leggen naar MySQL. Zie bijvoorbeeld ook deze link

```
library(odbc)
library(RMySQL)
library(DBI)
library(dplyr)

## maak een connectie naar mySQL, zie ook connections tab
con_sql <-  dbConnect(odbc::odbc(), "my-connector")

## laat zien welek tabelllen er zijn
dbListTables(con_sql)

## copier lokale R data frames naar MySQL
dbWriteTable(conn = con_sql, name = 'mtcars', value = mtcars)

## je hebt een'handle' nodig naar een tabel, daar kan je dplyr op gebruiken
tbl_handle = tbl(con_sql, "mtcars")

tbl_handle %>% count()


zz = tbl_handle %>% filter(disp > 260)
class(zz)

lokaledata = zz %>% collect()

dbDisconnect(con_sql)
```

Interactive grafieken

Naast de statistische grafieken die we hierboven hebben gemaakt kan je in R ook interactieve grafieken maken. We geven in deze sectie een aantal voorbeelden.

Het plotly package

Plotly is een zeer mooie library voor het maken van interactieve visualisaties. Zie voor meer info plotly. We zullen een aantal voorbeelden hieronder laten zien.

scatters

```
library(plotly)
mtcars = mtcars
mtcars$naampjes = row.names(mtcars)

plot_ly(data = mtcars, x=~mpg, y = ~wt)
plot_ly(data = mtcars, x=~mpg, y = ~wt, color = ~cyl)
plot_ly(data = mtcars, x=~mpg, y = ~wt, text = ~naampjes)
```

barcharts

We gebruiken de restaurants data set weer

```
Restaurants <- readr::read_csv("data/Restaurants.csv")
keuken = Restaurants %>% dplyr::group_by(keuken) %>% dplyr::summarise(n=n())

p = plot_ly(data = keuken,
  x = ~keuken,
  y = ~n,
  type = "bar"
)

p
```

Sorteren op volgorde van aantallen kan je doen door reorder factor

```
keuken = keuken %>%
  mutate(
  keuken = forcats::fct_reorder(keuken,n, .desc=TRUE)
)
p = plot_ly(data = keuken,
  x = ~keuken,
  y = ~n,
  type = "bar"
)

p
```

boxplots

```
p = plot_ly(data = Restaurants, y = ~aantalreviews, color = ~keuken,type = "box")
p
```

3D plots

If you have 3D data you can plot it in a 3D scatter plot

```
p = plot_ly(data = mtcars, x=~mpg, y=~cyl, z = ~disp)
p
```

A matrix with values can be plotted as a 3D surface plot

```
volcano
plot_ly(z = ~volcano) %>% add_surface()
```

ggplot integratie

Je kan een grafiek maken met ggplot en dan de functie 'ggplotly gebruiken om de grafiek interactief te maken. Als de grafiek veel punten bevat kan het sloom over komen. Vandaar dat onderstaande voorbeeldje alleen 3000 punten sampled.

```
p = ggplot(
  data = diamonds %>% dplyr::sample_n(3000),
  aes(x = carat, y = price)
) +
  geom_point(
    alpha = 0.05,
    aes(text = paste("Clarity:", clarity))
  ) +
  geom_smooth(aes(colour = cut, fill = cut)) +
  facet_wrap(~ cut)
p

ggplotly(p)
```

Kaartjes met leaflet

Leaflets worden gebruikt om interactieve kaartjes te maken. By deault gebruiken open street maps als achtergrond.

```
library(leaflet)

m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=174.768, lat=-36.852, popup = "The birthplace of R")
m

# kan geblokt worden door fire walls.
```

Plot de restaurants in Zwolle op een kaartje. Data van Iens gescraped.

```
Restaurants <- readr::read_csv("data/Restaurants.csv")
Zwolle = Restaurants %>% filter(plaats == "Zwolle")

## tooltip teksten kunnen geplaatst worden
ptekst = paste(Zwolle$restNamen, '<BR> aantal reviews: ', Zwolle$aantalreviews)

m2 = leaflet(data = Zwolle)
m2 = m2 %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng = ~LONGs, lat = ~LATs, popup = ptekst)
m2
```

Je kan kleurtje gebruiken, gebruik dan `addCirclemarkers`

```
reds = colorNumeric("Reds", domain = NULL)

Zwolle = Zwolle %>% filter(!is.na(aantalreviews), aantalreviews < 80)
ptekst = paste(Zwolle$restNamen, '<BR> aantal reviews: ', Zwolle$aantalreviews)

m2 = leaflet(data = Zwolle)
m2 = m2 %>%
  addTiles() %>%
  addCircleMarkers(
    lng = ~LONGs,
    lat = ~LATs,
    fillColor = ~reds(aantalreviews),
    fillOpacity = 1, popup = ptekst
  )
m2
```

polygonen op een leaflet

Je kan ook vormen/polygons op een leaflet kaart krijgen. Hieronder is een voorbeeld van provincies, eerst moet je de data van de polygonen hebben. Dit staat in zogenaamde shape files van de provincies, deze kan je downloaden van http://www.imergis.nl/asp/47.asp. Op deze site zie je provinciegrenzen_exact, download deze en pak deze uit. In mijn data directory is dat al gedaan.

Eerst is er een beetje voorbewerking nodig om de shape file te bewerken in R tot een zogenaamde SpatialPolygonsDataframe. Dan kan je dit met leaflet te kunnen plotten.

```
#### inlezen shape file
ProvNL <- readShapeSpatial("data/TopGrenzen-prov-actueel.shp")
class(ProvNL)
summary(ProvNL)

#### Zet coordinatensysteem
proj4string(ProvNL) <- CRS("+init=epsg:28992")

#### transformeer naar long /lat
ProvNL = spTransform(ProvNL, CRS("+proj=longlat +datum=WGS84"))

#### Het object ProvNL is een zgn spatialpolygons object, daar zit een data frame in
ProvNL@data

#### maak een hele simpele plot
plot(ProvNL)
```

Maar nu kun je het spatial polygons object ook op een leaflet plotten.

```
leaflet(ProvNL) %>%
  addTiles() %>%
  addPolygons(
    stroke = TRUE, weight = 1, fillOpacity = 0.15, smoothFactor = 0.15
  )
```

Je kan de kleuren van de provincies ook veranderen

```
ProvNL$Kleur = sample(c("red","blue"), 12, replace = TRUE)
```

```
ProvNL@data

leaflet(ProvNL) %>%
  addTiles() %>%
  addPolygons(
    stroke = TRUE, weight = 1, fillOpacity = 0.25, smoothFactor = 0.15, fillColor = ~Kleur
  )
```

** Kleuren Schemas **

De kleuren kan je ook data gedreven met een kleuren schema bepalen.

```
#### maak eerst wat data voor de provincies
ProvNL$Leeftijd = runif(12)

### definieer een 3 niveau schema
colpal <- colorQuantile(
  palette = green2red(3), n=3,
  domain = ProvNL$Leeftijd
)

### gebruik kleurtjes en een popup tekst
leaflet(ProvNL) %>%
  addTiles() %>%
  addPolygons(
    stroke = TRUE,
    weight = 1,
    fillOpacity = 0.25,
    smoothFactor = 0.15,
    fillColor = colpal(ProvNL$Leeftijd),
    popup = as.character(ProvNL$Leeftijd)
  )
```

Hieronder is een voorbeeld om op postcode 2 posities niveau een kaart te maken. We hebben eerst de data nodig die de contouren van een PC2 gebied definieert, en dan is er wat preparatie nodig om de data in polygoon objecten te krijgen. Deze gaat iets anders dan de provincies die in 1 shape file stonden.

```
# importeer data uit een CSV file
pc2_kaart <- read_csv("data/pc2_kaart.csv")
PC2_data = dplyr::select(
  pc2_kaart,
  Longitude,
  Latitude,
  PC2CODE ) %>%
  as.data.frame()

# Het moet een lijst van gebieden zijn.
Pc2_list = split(PC2_data, PC2_data$PC2CODE)
Pc2_list = lapply(Pc2_list, function(x) { x["PC2CODE"] <- NULL; x })


### Maak van elk lijst object een Polygon
plg = sapply(Pc2_list, Polygon)

plg[[1]]
```

Nu gaan we per PC2 gebied het aantal restaurants tellen en dit in een kaartje zetten.

```
PC2 = Restaurants %>%
  mutate(
    PC2 = stringr::str_sub(PCs,1,2)
  ) %>%
  group_by(PC2) %>%
  summarise(n=n()) %>%
  filter(PC2 >= "10")

## initieer de leaflet en voeg PC2 polygon 1 voor 1 toe

pc2_lf = leaflet() %>%  addTiles()

## definieer een kleuren schema
colpal <- colorQuantile(
  palette = green2red(7), n=7,
  domain = PC2$n
)

## elk polygoontje krijgt nu op basis van zijn restaurant aantal het 'juiste' kleurtje

### teken per polygon, dat is het makkelijkst
for (i in 1:length(plg)){
  ptxt = paste(
    "PC2: ", as.character(PC2$PC2[i]),
    "<br>",
    "Aantal Restaurants",
    as.character(PC2$n[i])
  )
  pc2_lf = pc2_lf %>%
    addPolygons(
      data = plg[[i]],
      weight = 2,
      smoothFactor = 1,
      fillOpacity = 0.55,
      fillColor= colpal(PC2$n[i]),
      popup = ptxt
    )
}

pc2_lf
```

Netwerk grafieken, visnetwork

Het package `visnetwork` kan worden gebruikt om interactieve netwerk grafieken te maken. Je hebt twee data frames nodig, een met de nodes en een met de edges. Laten we beginnen met een simpel voorbeeldje.

```
library(visNetwork)

nodes = data.frame(
  id = c(1,2,3,4)
)
edges = data.frame(
```

```
  from = c(1,3,4),
  to  = c(2,2,2)
)

visNetwork(nodes, edges)
```

Er zijn diverse opties die je kan meegeven. Een deel van de opties kan je via de data frames meegeven.

```
nodes <- data.frame(
  id = 1:10,
  label = paste("Node", 1:10),                                  # labels
  group = c("GrA", "GrB"),                                      # groups
  value = 1:10,                                                 # size
  shape = c("square", "triangle", "box", "circle", "dot", "star", "ellipse", "database", "text", "diamon
  title = paste0("<p><b>", 1:10,"</b><br>Node !</p>"),          # tooltip
  color = c("darkred", "grey", "orange", "darkblue", "purple"),  # color
  shadow = c(FALSE, TRUE, FALSE, TRUE, TRUE))                   # shadow

edges <- data.frame(from = sample(1:10,8), to = sample(1:10, 8),
  label = paste("Edge", 1:8),                                  # labels
  length = c(100,500),                                         # length
  arrows = c("to", "from", "middle", "middle;to"),             # arrows
  dashes = c(TRUE, FALSE),                                     # dashes
  title = paste("Edge", 1:8),                                  # tooltip
  smooth = c(FALSE, TRUE),                                     # smooth
  shadow = c(FALSE, TRUE, FALSE, TRUE))                        # shadow

visNetwork(nodes, edges)
```

Andere opties kan je via de functie `visnetwork` meegeven. Bijvoorbeeld highlight nearest

```
nodes <- data.frame(
  id = 1:15,
  label = paste("Label", 1:15),
  group = sample(LETTERS[1:3], 15, replace = TRUE)
)

edges <- data.frame(
  from = trunc(runif(15)*(15-1))+1,
  to = trunc(runif(15)*(15-1))+1
)

visNetwork(nodes, edges) %>%
  visOptions(
    highlightNearest = TRUE,
    nodesIdSelection = TRUE
  )
```

Nog een netwerk voorbeeldje van restaurant bezoekers in het mooie stadje Hoorn. Bij het scrapen van Iens, kan je ook zien dat een reviewer eerst een bepaald restaurant heeft gereviewed en dan een ander, dat is in een netwerk graph te zetten.

Focus nu alleen op het mooie stadje Hoorn.

```
Hoornnodes = readRDS("data/HoornNodes.RDs")
HoornEdges = readRDS("data/HoornEdges.RDs")
```

```
visNetwork(Hoornnodes, HoornEdges) %>%
  visLegend() %>%
  visOptions(
    highlightNearest = TRUE,
    nodesIdSelection = TRUE
  ) %>%
  visInteraction(
    navigationButtons = TRUE
  ) %>%
  visPhysics( maxVelocity = 25)
```

Zoomable circle packing plots, sunburst plots en chord diagrams

Circle packings zijn een leuke manier om hierarchische data te visualiseren. Installeer eerst het package van Github.

```
devtools::install_github("jeromefroe/circlepackeR")
```

Maak wat dummy data aan. Volgend voorbeeldje is een hiërarchischer lijst.

```
hierarchical_list <- list(name = "World",
                          children = list(
                            list(name = "North America",
                                 children = list(
                                   list(name = "United States", size = 308865000),
                                   list(name = "Mexico", size = 107550697),
                                   list(name = "Canada", size = 34033000))),
                            list(name = "South America",
                                 children = list(
                                   list(name = "Brazil", size = 192612000),
                                   list(name = "Colombia", size = 45349000),
                                   list(name = "Argentina", size = 40134425))),
                            list(name = "Europe",
                                 children = list(
                                   list(name = "Germany", size = 81757600),
                                   list(name = "France", size = 65447374),
                                   list(name = "United Kingdom", size = 62041708))),
                            list(name = "Africa",
                                 children = list(
                                   list(name = "Nigeria", size = 154729000),
                                   list(name = "Ethiopia", size = 79221000),
                                   list(name = "Egypt", size = 77979000))),
                            list(name = "Asia",
                                 children = list(
                                   list(name = "China", size = 1336335000),
                                   list(name = "India", size = 1178225000),
                                   list(name = "Indonesia", size = 231369500)))
                          )
)
```

and plot it

```
circlepackeR::circlepackeR(hierarchical_list)
```

Interactieve sunburst plots kan je ook mooi gebruiken voor hierarchische data.

```
library(sunburstR)

sequences <- read.csv(
  system.file("examples/visit-sequences.csv",package="sunburstR")
  ,header = FALSE
  ,stringsAsFactors = FALSE
)[1:100,]

sunburst(sequences)

sunburst(
  sequences
  ,count = TRUE
)

## aangepaste text in het midden
sunburst(
  sequences
  # apply sort order to the legends
  ,legendOrder = unique(unlist(strsplit(sequences[,1],"-")))
  # just provide the name in the explanation in the center
  ,explanation = "function(d){return d.data.name}"
)
```

Chord diagrams zijn een leuke manier om "A naar B" data weer te geven, relaties tussen entiteiten.

```
#devtools::install_github("mattflor/chorddiag")
library(chorddiag)

# overgangsmatrix
m <- matrix(
  c(11975,  5871, 8916, 2868,
    1951, 10048, 2060, 6171,
    8010, 16145, 8090, 8045,
    1013,   990,  940, 6907
  ),
  byrow = TRUE,
  nrow = 4, ncol = 4
)

haircolors <- c("black", "blonde", "brown", "red")
dimnames(m) <- list(have = haircolors, prefer = haircolors)
m

groupColors <- c("#000000", "#FFDD89", "#957244", "#F26223")
chorddiag(m, groupColors = groupColors, groupnamePadding = 20)
```