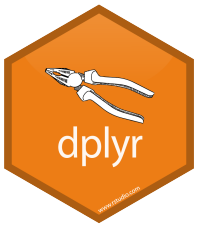
Introductie dplyr

Data manipulatie

Data wrangling met `dplyr`, `dbplyr` en `tidyr`

Inleiding



In deze les gaan we het volgende leren:

- Werken met pipes
- Rijen en kolommen selecteren op basis van voorwaarden
- Nieuwe kolommen berekenen
- Data samenvatten
- Data samenvoegen en transformeren
- We gebruiken 2 datasets
  - de interne dataset `iris` die standaard in R zit
  - `pilootstudie.csv` die meegeleverd is met de code

Voorbereidingen in Rstudio:

```
#Laadt het pakket tidyverse
library(tidyverse)

#Kijk wat je werkdirectory is, en pas die eventueel aan met setwd of het menu
getwd()

#Lees pilootstudie.csv in
#Deze code gaat er vanuit dat dit in de data folder staat in je werkdirectory
piloot <- read_csv2('data/pilootstudie.csv')
```

Waarom dplyr:

- Basis R syntax met heel veel geneste haakjes is moeilijk te lezen en te debuggen
- Alle dplyr functies hebben een dataset als input en als output, zodat er geen datatype conversie problemen ontstaan
- Werkt heel snel in vergelijking met andere R functies

Dataset datatypes

In R hebben hebben datasets doorgaans het datatype `data.frame` of `tibble`.

- `data.frame` is de standaard in R voor datasets
- `tibble` is een uitbreiding hierop
  - ondersteunt definitie van groepering (zie verder)
  - toont de dataset op een meer elegante manier in de console, zodat je geen ellenlange output krijgt

    - de dimensies worden getoond
    - eventuele groepering
    - standaard 10 rijen getoond
    - zoveel kolommen getoond als je R consolebreedte toelaat, het bestaan van extra kolommen wordt aangegeven
  - `View()` aangeraden als je de volledige data wil kunnen zien
  - `data.frames` kunnen naar `tibble` geconverteerd worden met de functie `as_tibble()`

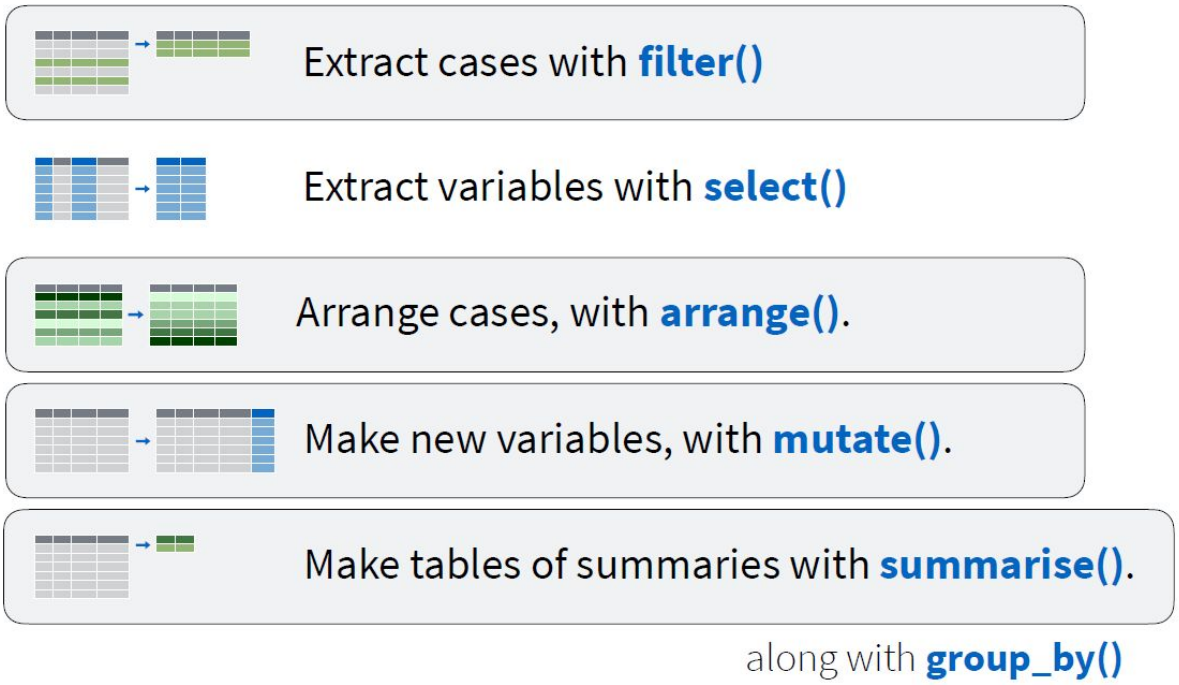
```
#output als data.frame
head(iris, n = 10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
```

```
#output als tibble
piloot #piloot is een tibble omdat die met read_csv2 ingelezen is
```

```
## # A tibble: 1,638 x 8
##    Proefvlak  Boom Ploeg Meting Omtrek Referentie Toestel Hoogte
##        <dbl> <dbl> <dbl>  <dbl>  <dbl>      <dbl>   <dbl>  <dbl>
##  1         1     1     1      1    106       106.       1   22.8
##  2         1     1     1      2    105       106.       1   21.9
##  3         1     1     1      3    105       106.       1   20.7
##  4         1     1     2      1    108       106.       1   21.3
##  5         1     1     2      2    108       106.       1   22.0
##  6         1     1     2      3    106       106.       1   20.6
##  7         1     1     3      1    110       106.       1   20.4
##  8         1     1     4      1    105       106.       2   24.1
##  9         1     1     4      1    105       106.       1   24.4
## 10         1     1     4      2    106       106.       1   23.1
## # ... with 1,628 more rows
```

Wat kan je allemaal met het pakket dplyr?



- `dplyr` (en ook `dbplyr` en `tidyr`) is een onderdeel van het `tidyverse` package
- Niet nodig om `dplyr` apart te installeren of te laden als je tidyverse laadt

```
library(tidyverse)
```

Belangrijk om dplyr vlot te gebruiken is gebruik maken van `pipes`.

Pipe %>%

Een pipe kan je gebruiken om in R code leesbaar en uitvoerbaar te maken van boven naar beneden, in plaats van uit de binnnenste haakjes naar buiten toe te werken. Een pipe lees je als: **neem de uitkomst van wat voor de pipe staat en doe daarmee wat na de pipe staat**.

Hieronder wordt de data gebruikt en daar `functie1()` op uitgevoerd, de uitkomst hiervan wordt dan gebruikt voor `functie2()`, en zo kan je een heel lange ketting maken.

Nieuwe of aangepaste kolommen die je in `functie1()` gemaakt hebt zijn direct bruikbaar door `functie2()`

```
resultaat <- data %>% functie1() %>% functie2()
```

- Eerste argument in de functie (wat een dataset is) na de pipe moet niet opgegeven worden
  - Hetgeen voor de `pipe` staat, wordt als eerste argument (data) gebruikt in de functie na de `pipe`
- *Shortcut voor pipe*: CTRL + SHIFT + M
- **Tip: Eindig de regel met een `pipe`** en zet de volgende bewerking op een nieuwe regel. Dit maakt de code beter leesbaar en vereenvoudigt het debuggen

```
resultaat <- data %>%
  functie1() %>%
  functie2() %>%
  ...
  functieN()
```

- Je kan gemakkelijk de code deels laten uitvoeren, dus als je wil weten wat het resultaat is nadat

`functie1()` uitgevoerd is, kan je de code selecteren tot net voor de pipe na `functie1()` en dan dat geselecteerde stuk uitvoeren.

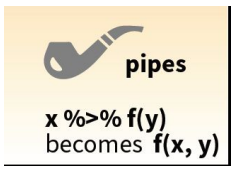
resultaat <- data %>%

functie1() %>%

functie2() %>%

...

functieN()



Dplyr syntax

Alle functies in het `dplyr` package werken op dezelfde manier.

- Als eerste argument verwachten ze de data waarop de bewerking uitgevoerd moet worden.
- Alle volgende argumenten zijn details over de uit te voeren bewerking.

Ideaal dus om gebruik te maken van pipes `%>%`.

Veel gebruikte functies

`filter()`

- Rijen selecteren op basis van één of meerdere logische voorwaarden

```
#filter code zonder pipes
filter(dataset, voorwaarden)
```

```
#filter code met pipes
dataset %>%
  filter(voorwaarden)
```

Logische voorwaarden

x < y	Less than
x > y	Greater than
x == y	Equal to
x <= y	Less than or equal to
x >= y	Greater than or equal to
x != y	Not equal to
x %in% y	Group membership
is.na(x)	Is NA
!is.na(x)	Is not NA

Voorwaarden combineren

a & b	and
a | b	or
!a	not

De ampersand `&` mag ook vervangen worden door een komma om voorwaarden te combineren die beiden voldaan moeten zijn.

Voorbeelden

`iris` data

1. Selecteer alle records voor de soort `virginica` en bewaar in het object `iris1`
2. Selecteer alle records van `virginica` waarvoor `Sepal.Length` groter dan of gelijk aan 7 is
3. Bewaar het resultaat in iris2

```
##vb1: selecteer de virginicas
iris1 <- filter(iris, Species == "virginica")
head(iris1)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          6.3         3.3          6.0         2.5 virginica
## 2          5.8         2.7          5.1         1.9 virginica
## 3          7.1         3.0          5.9         2.1 virginica
## 4          6.3         2.9          5.6         1.8 virginica
## 5          6.5         3.0          5.8         2.2 virginica
## 6          7.6         3.0          6.6         2.1 virginica
```

```
##vb2: 3 methoden om virginicas met Sepal>Length >= 7 te filteren
filter(iris, Species == "virginica" & Sepal.Length >= 7) #methode1
filter(iris, Species == "virginica",  Sepal.Length >= 7) #methode2
iris %>%                                                 #methode3
  filter(Species == "virginica") %>%
  filter(Sepal.Length >= 7)
```

```
##vb3: Bewaar je resultaat in een nieuwe dataset
iris2 <-
  iris %>%
  filter(Species == "virginica") %>%
  filter(Sepal.Length >= 7)

#vaak wordt gekozen om de data op dezelfde regel te zetten
#deze code doet hetzelfde als hierboven
iris2 <- iris %>%
  filter(Species == "virginica") %>%
  filter(Sepal.Length >= 7)

#De meest gebruikte methode op INBO is als volgt (identiek als hierboven)
iris2 <- iris %>%
  filter(Species == "virginica",
         Sepal.Length >= 7)
head(iris2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1          7.1         3.0          5.9         2.1 virginica
## 2          7.6         3.0          6.6         2.1 virginica
```

```
## 3          7.3         2.9          6.3         1.8 virginica
## 4          7.2         3.6          6.1         2.5 virginica
## 5          7.7         3.8          6.7         2.2 virginica
## 6          7.7         2.6          6.9         2.3 virginica
```

`pilootstudie.csv` data De data moet je eerst inlezen (zie voorbereidingen in Rstudio)

1. Verwijder alle records waarvoor `Omtrek` of `Hoogte` ontbrekend zijn en bewaar het resultaat in `piloot2` (voor later gebruik)
2. Selecteer in `piloot2` alle records van de ploegen 1, 5 en 7

```
##vb1: Behou enkel rijen met aanwezige Omtrek en Hoogte
piloot2 <- piloot %>%
  filter(!is.na(Omtrek) & !is.na(Hoogte))

#Kijk of er rijen verdwenen zijn
nrow(piloot)
```

```
## [1] 1638
```

```
nrow(piloot2)
```

```
## [1] 1145
```

```
##vb2: behou enkel ploeg 1, 5 en 7
piloot2 %>%
  filter(Ploeg %in% c(1, 5, 7))

#identiek maar met een logische OR
piloot2 %>%
  filter(Ploeg == 1 | Ploeg == 5 | Ploeg == 7)
```

Veel voorkomende fouten

- Gebruik van = in plaats van ==

```
filter(iris, Species = "virginica") #fout
filter(iris, Species == "virginica") #ok
```

- Vergeten van de aanhalingstekens rond tekst

```
filter(iris, Species == virginica) #fout, R verwacht het object virginica
filter(iris, Species == "virginica") #ok, R verwacht de tekst virginica
```

- Verschillende testen samengevoegd

```
filter(iris, 5 < Sepal.Length < 7) #fout
filter(iris, 5 < Sepal.Length & Sepal.Length < 7) #ok
```

`arrange()`

- Rijen ordenen van klein naar groot volgens een of meerdere kolommen (gescheiden door komma's)

```
arrange(dataset, variabele1, variabele2, ...)

dataset %>%
  arrange(variabele1, variabele2, ...)
```

- Ordenen van groot naar klein kan met `desc()`

```
arrange(dataset, desc(variabele))
```

- **Opgelet**: Volgorde van variabelen is belangrijk voor het resultaat van de rangschikking

Voorbeelden `piloot2` data

1. Vind de dunste bomen. Sorteer daarvoor volgens `Omtrek`
2. Vind de hoogste bomen. Sorteer daarvoor volgens `Hoogte` van groot naar klein
3. Sorteer de bomen eerst op `Omtrek` daarna op `Hoogte`
4. Sorteer de bomen eerst volgens `Omtrek` van klein naar groot, daarna op `Hoogte` van groot naar klein

```
##vb1: sorteer volgens kleinste omtrek
arrange(piloot2, Omtrek)
```

```
##vb2: sorteer volgens grootste Hoogte
piloot2 %>%
  arrange(desc(Hoogte))
```

```
##vb3: Sorteer eerst om omtrek, daarna op hoogte
arrange(piloot2, Omtrek, Hoogte)
```

```
##vb4: sorteer op omtrek daarna volgens grootste Hoogte
piloot2 %>%
  arrange(Omtrek, desc(Hoogte)) %>%
  View() #toon de resultaten in het Rstudio grid
```

`mutate()`

- Nieuwe variabele(n) aanmaken op basis van bestaande variabele(n) in de dataset

```
mutate(dataset, NieuweVariabele1, NieuweVariabele2, ...)
dataset %>%
  mutate(NieuweVariabele1, NieuweVariabele2, ...)
```

- Altijd in de vorm `NieuweVariabele = bewerking op bestaande variabele(n)`
- Mogelijk om meerdere variabelen tegelijk aan te maken
  - Gescheiden door komma's
  - Mogelijk om nieuwe variabele onmiddellijk te gebruiken als input
- **Belangrijk**: resultaat moet een even lange vector zijn als de input
- Voor de leesbaarheid gebruik je best een nieuwe regel voor iedere variabele die je met `mutate` aanmaakt

Voorbeeld `iris` data

1. Oppervlakte van de Sepal blaadjes
2. Verhouding van de oppervlakte van de kelk- en kroonblaadjes

```
##vb1: oppervlakte Sepal
nieuwe_iris <- mutate(iris, Sepal.Opp = Sepal.Length * Sepal.Width)
head(nieuwe_iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Opp
## 1          5.1         3.5          1.4         0.2  setosa     17.85
## 2          4.9         3.0          1.4         0.2  setosa     14.70
## 3          4.7         3.2          1.3         0.2  setosa     15.04
## 4          4.6         3.1          1.5         0.2  setosa     14.26
## 5          5.0         3.6          1.4         0.2  setosa     18.00
## 6          5.4         3.9          1.7         0.4  setosa     21.06
```

```
##vb2: Verhouding oppervlaktes kelk- en kroonblaadjes
#voor de leesbaarheid gebruik best nieuwe regels
iris %>%
  mutate(Sepal.Opp = Sepal.Length * Sepal.Width,
```

```
        Petal.Opp = Petal.Length * Petal.Width,
           Verhouding = Sepal.Opp / Petal.Opp) %>%
  head() #toon de eerste rijen met de nieuwe berekeningen
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Opp Petal.Opp
## 1          5.1         3.5          1.4         0.2  setosa     17.85      0.28
## 2          4.9         3.0          1.4         0.2  setosa     14.70      0.28
## 3          4.7         3.2          1.3         0.2  setosa     15.04      0.26
## 4          4.6         3.1          1.5         0.2  setosa     14.26      0.30
## 5          5.0         3.6          1.4         0.2  setosa     18.00      0.28
## 6          5.4         3.9          1.7         0.4  setosa     21.06      0.68
##   Verhouding
## 1   63.75000
## 2   52.50000
## 3   57.84615
## 4   47.53333
## 5   64.28571
## 6   30.97059
```

```
#Bovenstaande bewaart je resultaat niet en toont enkel een deel
#Onderstaande bewaart het resultaat als object nieuwe_iris
nieuwe_iris <- iris %>%
  mutate(Sepal.Opp = Sepal.Length * Sepal.Width,
         Petal.Opp = Petal.Length * Petal.Width,
         Verhouding = Sepal.Opp / Petal.Opp)
```

`select()` en `transmute()`

- Een of meerdere kolommen selecteren, namen gescheiden door komma's

```
select(dataset, kolomnaam1, kolomnaam2, ...)
dataset %>%
  select(kolomnaam1, kolomnaam2, ...)
```

- Alternatief voor vierkante haken `[ ]` of namen expliciet meegeven
  - Past in de hele piping filosofie
- Interessante functies om gelijkaardige kolomnamen te selecteren
  - `starts_with("xxx")`: alle kolomnamen waarvan de naam begint met `xxx`
  - `ends_with("xxx")`: alle kolomnamen waarvan de naam eindigt met `xxx`
  - `contains("xxx")`: alle kolomnamen waarvan de naam `xxx` bevat
  - `everything()`: alle kolomnamen
  - `any_of()`: alle kolomnamen die je in een charactervector meegeeft
  - `where()`:alle kolomnamen die TRUE geven voor een functie
  - Je kan ook `-` ervoor zetten om kolomnamen niet te selecteren
    - bv. `-contains("xxx")`: alle kolomnamen die `xxx` niet bevatten
  - bovenstaande wordt impliciet als de variabele `.vars` meegegeven

```
  #selecteer alle variabelen die beginnen met Petal
  select(iris, starts_with("Petal"))

  #selecteer alle variabelen die niet eindigen met Width
  select(iris, -ends_with("Width"))

  #identiek als hierboven maar met .vars expliciet opgegeven
  select(iris, .vars = -ends_with("Width"))

  #selecteer alle numerieke variabelen
  #.vars hoeft niet expliciet vermeld
```

```
select(iris, .vars = where(is.numeric))

#via kolomnaamvector
kolommen <- c("Species", "Petal.Width")
select(iris, .vars = any_of(kolommen))
```

- Mogelijk om geselecteerde variabelen ineens van naam te veranderen
  - Nadeel dat alle niet-genoemde variabelen niet meegenomen worden

```
#selecteer enkel Species en hernoem deze als Soort
select(iris, Soort = Species)
```

- `transmute()` combineert mutate en select: het werkt hetzelfde als mutate, maar enkel de kolommen die binnen `transmute()` staan worden behouden

```
#Behou de soort als uppercase en de Petal.Area in de data
#verwijder de rest
iris %>%
  transmute(Soort = toupper(Species),
            Petal.Area = Petal.Length * Petal.Width)
```

summarise() of summarize()

- Samenvattende waarde(n) berekenen
- Mogelijkheid om meerdere kenmerken (functies) te combineren
- Geeft doorgaans slechts 1 waarde (per functie) terug
- zowel het Engelse `summarise` als het Amerikaanse `summarize` kunnen worden gebruikt

```
summarise(data, functie(variabele))

data %>%
  summarise(functie(variabele))
```

- Varianten om kenmerken te berekenen voor alle variabelen, of een selectie van variabelen (gebruiken een iets andere syntax, zie `help`)
  - In de nieuwe dplyr versie wordt dit gebruik vervangen door `across`, al blijven onderstaande functies wel bestaan en mogen ze gebruikt worden
  - `summarise_all()`: vat samen voor alle variabelen
  - `summarise_at()` : vat samen voor specifiek opgegeven variabelen
  - `summarise_if()` : vat samen voor variabelen die aan een voorwaarde voldoen
- enkele voor summarise veel gebruikte speciaal gemaakte functies
  - `n()`: geef het aantal elementen terug
  - `n_distinct()` : geef het aantal verschillende elementen terug
  - `n_groups()` : geef het aantal verschillende groepen (zie group_by verder)

Voorbeelden

piloot2 data

1. Vind de dunste boom
2. Vind de hoogste boom, het aantal proefvlakken, en de mediaan voor de referentiemetingen
3. Bereken voor alle variabelen het gemiddelde

```
##vb1: kleinste omtrek
summarise(piloot2, Dunste = min(Omtrek))
```

```
## # A tibble: 1 x 1
##   Dunste
##    <dbl>
```

```
## 1     30
```

```
##vb2: maximale Hoogte, aantal proefvlakken, mediaan referentie
piloot2 %>%
  summarise(HoogsteBoom = max(Hoogte),
            AantalProefVlakken = n_distinct(Proefvlak),
            MediaanRef = median(Referentie))
```

```
## # A tibble: 1 x 3
##   HoogsteBoom AantalProefVlakken MediaanRef
##         <dbl>              <int>      <dbl>
## 1        30.6                  8       111.
```

```
##vb3: gemiddelde van alle variabelen
piloot2 %>%
  summarise_all(mean)
```

```
## # A tibble: 1 x 8
##   Proefvlak  Boom Ploeg Meting Omtrek Referentie Toestel Hoogte
##       <dbl> <dbl> <dbl>  <dbl>  <dbl>      <dbl>   <dbl>  <dbl>
## 1      4.49  6.50  3.16   1.75   111.       110.    1.33   21.5
```

iris data

1. Bereken voor alle numerieke variabelen het gemiddelde
2. Bereken voor alle variabelen (behalve Species) het minimum en het maximum

```
##vb1: gemiddelde van alle numerieke variabelen
iris %>%
  summarise_if(is.numeric, mean)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.843333    3.057333        3.758    1.199333
```

```
##vb2; minimum en maximum van alles behalve Species
#beter met across werken
#het commando list is hier nodig voor verschillende kolommen
iris %>%
  summarise_at(vars(-Species),
               list(minimum = min,
                    maximum = max))
```

```
##   Sepal.Length_minimum Sepal.Width_minimum Petal.Length_minimum
## 1                  4.3                   2                    1
##   Petal.Width_minimum Sepal.Length_maximum Sepal.Width_maximum
## 1                 0.1                 7.9                 4.4
##   Petal.Length_maximum Petal.Width_maximum
## 1                  6.9                 2.5
```

group_by()

- Gegevens groeperen volgens een of meerdere variabelen in een dataset. Dit doet niks, behalve er een gegroepeerde `tibble` van maken.

```
group_by(data, variabele1, variabele2, ...)

data %>%
  group_by(variabele1, variabele2, ...)
```

- Groepering ongedaan maken met `ungroup()`
  - Meestal niet nodig, maar sommige functies kunnen niet om met gegroepeerde data

- Meestal gebruikt in combinatie met `summarise()` om per groep samenvattende kenmerken te kunnen berekenen
  - De kolommen die in `group_by()` staan worden altijd in het resultaat opgenomen
    - als je dit niet wenst, moet je `ungroup()`gebruiken

```
data %>%
  group_by(variabele_i) %>%
  summarise(functie(variabele_j))
```

Voorbeelden

`iris` data

1. Bereken per soort het aantal waarnemingen en de gemiddelde `Sepal.Width`

```
iris %>%
  group_by(Species) %>%
  summarise(Aantal = n(),
            Gemiddelde = mean(Sepal.Width))
```

```
## # A tibble: 3 x 3
##   Species    Aantal Gemiddelde
##   <fct>       <int>      <dbl>
## 1 setosa         50       3.43
## 2 versicolor     50       2.77
## 3 virginica      50       2.97
```

`piloot2` data

1. Bereken per boom het minimum, gemiddelde en maximum van de omtrek en de hoogte (Proefvlak ook nodig om te groeperen omdat de nummering van de bomen in elk proefvlak opnieuw begint)
2. Bereken per ploeg het aantal metingen per proefvlak

```
piloot2 %>%
  group_by(Proefvlak, Boom) %>%
  summarise(MinOmtrek = min(Omtrek),
            GemOmtrek = mean(Omtrek),
            MaxOmtrek = max(Omtrek),
            MinHoogte = min(Hoogte),
            GemHoogte = mean(Hoogte),
            MaxHoogte = max(Hoogte))
```

```
## `summarise()` has grouped output by 'Proefvlak'. You can override using the `.groups` argument.
```

```
## # A tibble: 96 x 8
## # Groups:   Proefvlak [8]
##    Proefvlak  Boom MinOmtrek GemOmtrek MaxOmtrek MinHoogte GemHoogte MaxHoogte
##        <dbl> <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
##  1         1     1       105     106.        110      20.4      22.1      24.4
##  2         1     2        99     108.        111      19.3      21.4      23.9
##  3         1     3       101     102.        103      18.6      21.0      23.4
##  4         1     4       112     113.        114.     20.9      22.1      24
##  5         1     5       115     117.        118      20.8      23.4      26.3
##  6         1     6        98      99.7       109      19.5      20.9      22.6
##  7         1     7        83      84.4        86      19.5      21.0      23.9
##  8         1     8        99     101.        103      18.1      20.6      23
##  9         1     9        97      98.2        99      17        19.5      23.4
## 10         1    10       113     115.        116      19.7      21.7      23.7
## # ... with 86 more rows
```

```
piloot2 %>%
  group_by(Proefvlak, Ploeg) %>%
  summarise(Aantal = n())
```

```
## `summarise()` has grouped output by 'Proefvlak'. You can override using the `.groups` argument.
```

```
## # A tibble: 40 x 3
## # Groups:   Proefvlak [8]
##    Proefvlak Ploeg Aantal
##        <dbl> <dbl>  <int>
##  1         1     1     24
##  2         1     2     24
##  3         1     3     24
##  4         1     4     48
##  5         1     5     24
##  6         2     1     24
##  7         2     2     24
##  8         2     3     24
##  9         2     4     48
## 10         2     5     24
## # ... with 30 more rows
```

summarize .groups parameter

- In nieuwere dplyr versie is krijg je soms een melding dat je .groups moet aangeven. Dit is geen foutmelding, gewoon informatie.

```
summarise()` has grouped output by 'groeperingsvariabelenamen'.
You can override using the `.groups` argument.
```

- Standaard als je de `.groups` parameter niet gebruikt zal de functie summary het laatste groeplevel weglaten
  - dit is handig als je summaries maakt op een vorige summary
- Je kan dit gedrag wijzigen door het `.groups` argument te gebruiken binnen de summary functie
  - `.groups = "drop_last"` of `.groups = NULL`: groepeert niet langer op de laatste variabele in group_by
  - `.groups = "drop"`: verwijder alle groepering
  - `.groups = "keep"`: behou alle groepering
  - `.groups = "rowwise"`: maak van iedere rij in de output een aparte groep

```
test <- piloot %>%
  group_by(Proefvlak, Boom)
test #output: Groups: Proefvlak, Boom [96] (96 combinaties boom en proefvlak)
```

```
## # A tibble: 1,638 x 8
## # Groups:   Proefvlak, Boom [96]
##    Proefvlak  Boom Ploeg Meting Omtrek Referentie Toestel Hoogte
##        <dbl> <dbl> <dbl>  <dbl>  <dbl>      <dbl>   <dbl>  <dbl>
##  1         1     1     1      1    106       106.       1   22.8
##  2         1     1     1      2    105       106.       1   21.9
##  3         1     1     1      3    105       106.       1   20.7
##  4         1     1     2      1    108       106.       1   21.3
##  5         1     1     2      2    108       106.       1   22.0
##  6         1     1     2      3    106       106.       1   20.6
##  7         1     1     3      1    110       106.       1   20.4
##  8         1     1     4      1    105       106.       2   24.1
##  9         1     1     4      1    105       106.       1   24.4
## 10         1     1     4      2    106       106.       1   23.1
## # ... with 1,628 more rows
```

```r
#na de summary verdwijnt de laatste variabele in de groepering
#enkel nog:  Groups:   Proefvlak [8] blijft over
test %>%
  summarize(GemOmtrek = mean(Omtrek))
```

```
## `summarise()` has grouped output by 'Proefvlak'. You can override using the `.groups` argument.
```

```
## # A tibble: 96 x 3
## # Groups:   Proefvlak [8]
##    Proefvlak  Boom GemOmtrek
##        <dbl> <dbl>     <dbl>
##  1         1     1     106.
##  2         1     2     109.
##  3         1     3     102.
##  4         1     4     113.
##  5         1     5      NA
##  6         1     6      NA
##  7         1     7      84.4
##  8         1     8     100.
##  9         1     9      98
## 10         1    10      NA
## # ... with 86 more rows
```

```r
#je krijgt volgende melding boven het resultaat:
    #`summarise()` has grouped output by 'Proefvlak'.
    #You can override using the `.groups` argument.

#de groepering is volledig verdwenen
test %>%
  summarize(GemOmtrek = mean(Omtrek, na.rm = TRUE), .groups = "drop")
```

```
## # A tibble: 96 x 3
##    Proefvlak  Boom GemOmtrek
##        <dbl> <dbl>     <dbl>
##  1         1     1     106.
##  2         1     2     109.
##  3         1     3     102.
##  4         1     4     113.
##  5         1     5     117.
##  6         1     6      99.0
##  7         1     7      84.4
##  8         1     8     100.
##  9         1     9      98
## 10         1    10     114.
## # ... with 86 more rows
```

```r
#de oorspronkelijke groepering op Proefvlak en Proef blijft behouden
test %>%
  summarize(GemOmtrek = mean(Omtrek, na.rm = TRUE), .groups = "keep")
```

```
## # A tibble: 96 x 3
## # Groups:   Proefvlak, Boom [96]
##    Proefvlak  Boom GemOmtrek
##        <dbl> <dbl>     <dbl>
##  1         1     1     106.
##  2         1     2     109.
##  3         1     3     102.
##  4         1     4     113.
##  5         1     5     117.
##  6         1     6      99.0
```

```
##  7         1     7      84.4
##  8         1     8     100.
##  9         1     9      98
## 10         1    10     114.
## # ... with 86 more rows
```

```
#iedere rij is een aparte groep
test %>%
  summarize(GemOmtrek = mean(Omtrek, na.rm = TRUE), .groups = "rowwise")
```

```
## # A tibble: 96 x 3
## # Rowwise:  Proefvlak, Boom
##    Proefvlak  Boom GemOmtrek
##        <dbl> <dbl>     <dbl>
##  1         1     1     106.
##  2         1     2     109.
##  3         1     3     102.
##  4         1     4     113.
##  5         1     5     117.
##  6         1     6      99.0
##  7         1     7      84.4
##  8         1     8     100.
##  9         1     9      98
## 10         1    10     114.
## # ... with 86 more rows
```

```
#waarvoor 'drop_last' handig is
piloot %>%
  group_by(Proefvlak, Boom) %>%
  summarize(GemOmtrek = mean(Omtrek, na.rm = TRUE)) %>% #groepering op Boom is weg
  summarize(GemOmtrek = mean(GemOmtrek)) #dus nu per Proefvlak berekend
```

```
## `summarise()` has grouped output by 'Proefvlak'. You can override using the `.groups` argument.
```

```
## # A tibble: 8 x 2
##   Proefvlak GemOmtrek
##       <dbl>     <dbl>
## 1         1     104.
## 2         2     103.
## 3         3     129.
## 4         4     113.
## 5         5      98.6
## 6         6      87.8
## 7         7     126.
## 8         8     127.
```

Andere interessante functies

`distinct()`

- Verwijder dubbele rijen
- Mogelijk om één of meerdere variabelen te specifiëren
  - Indien je variabelen specifieert, zullen enkel deze nog overblijven in de dataset samen met eventuele groeperingsvariabelen

```
#1 rij valt weg omdat die identiek is aan een andere rij
#alle kolommen worden behouden
iris %>%  distinct()

#Toon alle verschillende Sepal.Length en Species combinaties
#enkel de vermelde kolommen worden behouden
```

```
iris %>% distinct(Sepal.Length, Species)
```

slice()

- Selecteer rijen op basis van de rijnummer of een vector van rijnummers
- Er zijn enkele wat varianten op slice
  - `slice_head()` en `slice_tail()`
    - Toon de bovenste n of onderste n rijen
  - `slice_min()` en `slice_max()`
    - Toon Sorteer rijen volgens een variabele en neem de bovenste of onderste n rijen
    - Je kan ook kiezen voor de bovenste fractie van rijen, bv. het bovenste kwart
    - In de help van `slice_min` vind je meer informatie
  - `slice_sample()`
    - Toon een random aantal of fractie van het totaal aantal rijen
    - De mogelijkheid is er ook om de sampling te wegen of dezelfde rij verschillende keren te selecteren

```
#Toon de eerste, vijfde en zevende rij
iris %>%
  slice(c(1, 5, 7))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          5.0         3.6          1.4         0.2  setosa
## 3          4.6         3.4          1.4         0.3  setosa
```

```
#Toon de 3 eerste rijen
iris %>%
  slice_head(n = 3)

#Toon de laatste 10% rijen
iris %>%
  slice_tail(prop = 0.10)

#Toon de 7 grootste Sepal.Length
iris %>%
  slice_max(Sepal.Length, n = 7)

#Toon de 7 kleinste Sepal.Length
iris %>%
  slice_min(Sepal.Length, n = 7)

#Toon de kleinste 25% Sepal.Length
iris %>%
  slice_min(Sepal.Length, prop = 0.25)

#Toon 10 willekeurige rijen
iris %>%
  slice_sample(n = 10)
```

pull()

- haal een waarde uit een dataset
  - resultaat is nu meestal een vector of een enkele waarde

```
#resultaat is een vector in plaats van een dataset
iris %>% pull(Sepal.Width)
```

```
#haal alle Petal.Width op voor de  virginica's
iris %>% filter(Species == "virginica") %>% pull(Petal.Width)
```

```
##  [1] 2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8 1.8 2.5 2.0 1.9 2.1 2.0 2.4 2.3 1.8 2.2 2.3
## [20] 1.5 2.3 2.0 2.0 1.8 2.1 1.8 1.8 1.8 2.1 1.6 1.9 2.0 2.2 1.5 1.4 2.3 2.4 1.8
## [39] 1.8 2.1 2.4 2.3 1.9 2.3 2.5 2.3 1.9 2.0 2.3 1.8
```

`rename(data, NieuweNaam = OudeNaam)`

- Variabelen hernoemen
- Alternatief voor hernoemen met `select()`
- Voordeel dat alle variabelen die niet hernoemd worden, identiek in de data blijven
- Je kan ook hernoemen door een functie te gebruiken (bv `toupper()` om alles uppercase te zetten) met de functie `rename_with()`

```
#Hernoem species naar soortnaam
iris_soort <- iris %>%
  rename(Soortnaam = Species)
colnames(iris_soort)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Soortnaam"
```

```
#maak alle namen uppercase
iris_soort <- iris_soort %>%
  rename_with(.fn = toupper, .cols = everything())
colnames(iris_soort)
```

```
## [1] "SEPAL.LENGTH" "SEPAL.WIDTH"  "PETAL.LENGTH" "PETAL.WIDTH"  "SOORTNAAM"
```

`count()`

- Tel het aantal rijen in de groepen gedefinieerd door 1 of meer variabelen
- Verkorte vorm voor eerst groeperen, en dan het aantal te berekenen per groep
- Heeft nog de extra optie om te sorteren: `count(..., sort = TRUE)`

```
#tel de volledige lengte van de dataset
count(iris)
```

```
##     n
## 1 150
```

```
#tel het aantal per soort
iris %>%
  count(Species)
```

```
##      Species  n
## 1     setosa 50
## 2 versicolor 50
## 3  virginica 50
```

```
    #identiek als hierboven (maar is een tibble door group_by)
iris %>%
  group_by(Species) %>%
  summarise(n = n())
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

```
#Tel het aantal Petal.Length > 5 voor de verschillende soorten
#en sorteer op het aantal (via sort = TRUE)
iris %>%
  count(Species, Petal.Length > 5, sort = TRUE)
```

```
##      Species Petal.Length > 5  n
## 1     setosa            FALSE 50
## 2 versicolor            FALSE 49
## 3  virginica             TRUE 41
## 4  virginica            FALSE  9
## 5 versicolor             TRUE  1
```

separate()

- Splits een kolom op in meerdere kolommen volgens een splitsingscharacter
- In het argument **sep** geef je het splitsinscharacter
- Sommige speciale karakters zoals een . moeten als \\. in het **sep**argument genoteerd worden

```
testdata <- data.frame(Name = c("pieter_verschelde", "raisa_carmen"))
testdata
```

```
##                Name
## 1 pieter_verschelde
## 2      raisa_carmen
```

```
separate(testdata, col = Name, into = c("voornaam", "naam"), sep = "_")
```

```
##   voornaam       naam
## 1   pieter verschelde
## 2    raisa     carmen
```

Werken met across (geavanceerd)

Vanaf dplyr versie 1.0 wordt **accross()** gestimuleerd ten nadele van **summarise_at**, **summarise_if**, **summarise_all**

- over de geselecteerde kolommen voer eenzelfde functie uit op deze kolommen
- wordt meestal gebruikt samen met een **mutate** of **summarize** functie
- via een lijst kan je verschillende functies tegelijk doorgeven die op de gekozen variabelen moeten gebeuren

```
#geef per soort de gemiddelde waarde van de Petal variabelen
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = starts_with("Petal"), mean))
```

```
## # A tibble: 3 x 3
##   Species    Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>
## 1 setosa             1.46       0.246
## 2 versicolor         4.26       1.33
## 3 virginica          5.55       2.03
```

```
#bereken per soort het gemiddelde en mediaan van de Petal variabelen
iris %>% group_by(Species) %>%
  summarize(across(.cols = starts_with("Petal"),
                   list(gemiddelde = mean,
                        mediaan = median)))
```

```
## # A tibble: 3 x 5
##   Species  Petal.Length_gem~ Petal.Length_med~ Petal.Width_gem~ Petal.Width_med~
##   <fct>                <dbl>              <dbl>            <dbl>            <dbl>
## 1 setosa                1.46               1.5            0.246              0.2
## 2 versico~              4.26               4.35           1.33               1.3
## 3 virgini~              5.55               5.55           2.03               2
```

```
#bereken de vierkantswortel van alle numerieke variabelen
iris %>%
```

```
  mutate(across(where(is.numeric), sqrt)) %>%
  slice_head(n = 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1     2.258318    1.870829     1.183216   0.4472136  setosa
## 2     2.213594    1.732051     1.183216   0.4472136  setosa
## 3     2.167948    1.788854     1.140175   0.4472136  setosa
## 4     2.144761    1.760682     1.224745   0.4472136  setosa
## 5     2.236068    1.897367     1.183216   0.4472136  setosa
```

```
#je kan ook je eigen functies maken en gebruiken
cm_to_mm <- function(x) { return(x * 10)}
iris %>%
  mutate(across(where(is.numeric), cm_to_mm)) %>%
  slice_head(n = 5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           51          35           14           2  setosa
## 2           49          30           14           2  setosa
## 3           47          32           13           2  setosa
## 4           46          31           15           2  setosa
## 5           50          36           14           2  setosa
```

```
#identiek als hierboven maar door de functie inline te gebruiken
#met .x zeg je dat dat de berekening op de respectievelijke kolom moet gebeuren
iris %>%
  mutate(across(where(is.numeric), ~ .x * 10)) %>%
  slice_head(n = 5)
```

Resultaat met meerdere records (geavanceerd)

- Doorgaans kan de summarise functie maar 1 resultaat per kolom teruggeven
- Maar het is mogelijk ook een volledige dataset als resultaat terug te krijgen

```
#maak een functie die het gemiddelde en mediaan berekent, zowel in cm als in mm
#het resultaat is een dataset met 3 kolommen: name, calc_cm en calc_mm
mean_and_median <- function(x) {
  data.frame(name = c("avg", "med"),
             calc_cm = c(mean(x), median(x)),
             calc_mm = c(mean(x * 10), median(x * 10)))
}

#voer de functie uit voor Petal.Width
iris %>%
  group_by(Species) %>%
  summarise(Resultaat = mean_and_median(Petal.Width)) %>%
  slice_head(n = 5)
```

```
## `summarise()` has grouped output by 'Species'. You can override using the `.groups` argument.
```

```
## # A tibble: 6 x 2
## # Groups:   Species [3]
##   Species    Resultaat$name $calc_cm $calc_mm
##   <fct>      <chr>             <dbl>    <dbl>
## 1 setosa     avg               0.246     2.46
## 2 setosa     med               0.2       2
## 3 versicolor avg               1.33     13.3
## 4 versicolor med               1.3      13
## 5 virginica  avg               2.03     20.3
## 6 virginica  med               2        20
```

```r
#dit kan ook met across toegepast worden op meerdere variabelen
#let op, dit is een data.frame die data.frames bevat
ingewikkeld <-
  iris %>%
  group_by(Species) %>%
  summarise(across(contains("Width"), mean_and_median))
```

```
## `summarise()` has grouped output by 'Species'. You can override using the `.groups` argument.
```

```r
ingewikkeld
```

```
## # A tibble: 6 x 3
## # Groups:   Species [3]
##   Species  Sepal.Width$name $calc_cm $calc_mm Petal.Width$name $calc_cm $calc_mm
##   <fct>    <chr>               <dbl>    <dbl> <chr>               <dbl>    <dbl>
## 1 setosa   avg                  3.43     34.3 avg                 0.246     2.46
## 2 setosa   med                  3.4      34   med                 0.2       2
## 3 versico~ avg                  2.77     27.7 avg                 1.33     13.3
## 4 versico~ med                  2.8      28   med                 1.3      13
## 5 virgini~ avg                  2.97     29.7 avg                 2.03     20.3
## 6 virgini~ med                  3        30   med                 2        20
```

```r
#Slechts 3 kolommamen
#Petal.Width en Sepal.Width zijn elk data.frames binnen de data.frame ingewikkeld
colnames(ingewikkeld)
```

```
## [1] "Species"     "Sepal.Width" "Petal.Width"
```

```r
#Selecteer enkel de berekende gemiddeldes in mm voor de Sepal.Width
ingewikkeld %>%
  pull(Sepal.Width) %>% #haal eerst de Sepal.Width dataset eruit
  filter(name == "avg") %>% #behou enkel de gemiddeldes
  pull(calc_mm) #haal hieruit dat de berekening in mm
```

```
## [1] 34.28 27.70 29.74
```

do en nest_by functie (geavanceerd)

- De `do()` functie maakt het mogelijk een heel ingewikkelde berekening uit te voeren op de groepen in de dataset
- Binnen de do zorg je dat het resultaat een dataset is
- Het object `.` binnen de do functie verwijst naar de huidige dataset
  - die kan je dan gewoon gebruiken zoals iedere andere dataset
  - alles wat je in do doet, zet je tussen accolades
  - de laatste regel moet de naam van de variabele zijn die je teruggeeft
- Een nieuw alternatief is werken met `nest_by`
  - `nest_by`: maakt een dataset waar iedere rij een volledige dataset per groep bevat
  - je gebruikt een functie zoals `mutate` of `summarise`
  - wat je hierbinnen programmeert wordt dan voor iedere subdataset uitgevoerd
  - het object `data` wordt gebruikt om binnen een `nest_by` naar de huidige dataset te verwijzen
  - via `summarize(nieuwe_variabele)` kan je de nesting weer ongedaan maken

```r
#Selecteer voor elke soort de 3 grootste Sepal.Width
# binnen do werken pipes niet
iris %>%
  group_by(Species) %>%
  do({
    resultaat <- slice_max(., Sepal.Width, n = 3, with_ties = FALSE)
    resultaat
  })
```

```
## # A tibble: 9 x 5
## # Groups:   Species [3]
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.7         4.4          1.5         0.4 setosa
## 2          5.5         4.2          1.4         0.2 setosa
## 3          5.2         4.1          1.5         0.1 setosa
## 4          6           3.4          4.5         1.6 versicolor
## 5          6.3         3.3          4.7         1.6 versicolor
## 6          7           3.2          4.7         1.4 versicolor
## 7          7.7         3.8          6.7         2.2 virginica
## 8          7.9         3.8          6.4         2   virginica
## 9          7.2         3.6          6.1         2.5 virginica
```

```
#wat doet nest: de dataset is maar 3 rijen lang (1 rij per groep),
#maar iedere rij bevat een dataset met alle informatie
iris %>% nest_by(Species)
```

```
## # A tibble: 3 x 2
## # Rowwise:  Species
##   Species                  data
##   <fct>      <list<tibble[,4]>>
## 1 setosa               [50 x 4]
## 2 versicolor           [50 x 4]
## 3 virginica            [50 x 4]
```

```
#bewaar in de kolom res de 3 grootste Sepal.Width per soort
#Dit geeft een kolom die een dataset is
#Je maakt die terug enkelvoudig door summarise op de kolomnaam
iris %>%
  nest_by(Species) %>%
  summarise(res = data %>%
              slice_max(Sepal.Width, n = 3, with_ties = FALSE)) %>%
  summarise(res)
```

```
## # A tibble: 9 x 5
## # Groups:   Species [3]
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa              5.7         4.4          1.5         0.4
## 2 setosa              5.5         4.2          1.4         0.2
## 3 setosa              5.2         4.1          1.5         0.1
## 4 versicolor          6           3.4          4.5         1.6
## 5 versicolor          6.3         3.3          4.7         1.6
## 6 versicolor          7           3.2          4.7         1.4
## 7 virginica           7.7         3.8          6.7         2.2
## 8 virginica           7.9         3.8          6.4         2
## 9 virginica           7.2         3.6          6.1         2.5
```

Datasets samenvoegen

bind_rows en bind_cols

- Datasets onder elkaar samenplakken kan via `bind_rows`
  - Kolomnamen die overeenkomen worden onder elkaar geplakt
  - Niet overeenkomende kolommen worden aan de dataset toegevoegd en krijgen een NA waarde
- Datasets naast elkaar plakken kan via `bind_cols`
  - Je bent zelf verantwoordelijk dat de rijen met elkaar corresponderen
  - De rijen van de 2 datasets moeten even lang zijn
  - Als er dubbele namen zijn zal R die naar unieke namen hernoemen, tenzij je dit met het

argument `.name_repair` anders instelt

```
#Maak 2 datasets en voeg aan de tweede een variabele toe
#en bind die terug samen
irisA <- iris %>%
  slice(1:3)
irisB <- iris %>%
  slice(148:150) %>%
  mutate(Opmerking = 'foutieve waarde')
bind_rows(irisA, irisB)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species       Opmerking
## 1          5.1         3.5          1.4         0.2    setosa            <NA>
## 2          4.9         3.0          1.4         0.2    setosa            <NA>
## 3          4.7         3.2          1.3         0.2    setosa            <NA>
## 4          6.5         3.0          5.2         2.0 virginica foutieve waarde
## 5          6.2         3.4          5.4         2.3 virginica foutieve waarde
## 6          5.9         3.0          5.1         1.8 virginica foutieve waarde
```

```
#Maak 2 datasets en bindt die samen via kolommen
#Hoe omgegaan wordt met dubbele namen kan je via .name_repair regelen (zie help)
irisC <- iris %>%
  select(Species, Petal.Length) %>%
  slice(1:3)
irisD <- iris %>%
  select(Species, Sepal.Length) %>%
  slice(1:3)
bind_cols(irisC, irisD)
```

```
## New names:
## * Species -> Species...1
## * Species -> Species...3
```

```
##   Species...1 Petal.Length Species...3 Sepal.Length
## 1      setosa          1.4      setosa          5.1
## 2      setosa          1.4      setosa          4.9
## 3      setosa          1.3      setosa          4.7
```

Join

- `bind_cols` kan foutieve resultaten veroorzaken als de data niet juist geordend zijn, daarom dat je beter werkt met joins (functies in het pakket `dbplyr` (onderdeel van tidyverse))
- Kolommen van tabellen worden aaneen gebonden door de rijen te koppelen volgens corresponderende waarden.
- Je moet de datasets koppelen aan de hand van 1 of meerdere variabelen
  - je kan dit met het argument `by`
  - bv. `inner_join(df1, df2, by = c("groep", "anderegroep")`
  - bv. `inner_join(df1, df2, by = c("groep_in_df1" = "groep_in_df2")`
  - indien je geen `by` opgeeft zal R joinen op basis van de kolomnamen die in beide datasets hetzelfde zijn
- Elke `join` resulteert in een andere combinatie van waarden uit beide tabellen.
  - `left_join()`: behou alle rijen van de eerste dataset en enkel de corresponderende van de andere, niet overeenkomende rijen worden met NA aangeduid
  - `right_join()`: hetzelfde, maar nu alle rijen van de tweede datasest
  - `inner_join()`: enkel de corresponderende rijen tussen de datasets blijven behouden
  - `full_join()`: behou alle rijen van beide datasets
  - `semi_join()`: behou enkel de eerste dataset, waarvoor de rijen corresponderen met de tweede dataset
  - `anti_join()`: behou enkel de eerste dataset, en enkel de rijen die niet corresponderen met de tweede dataset

```r
df1 <- data.frame(x = 1:3, grp = letters[1:3])
df2 <- data.frame(y = 4:6, grp = letters[c(1:2, 4)])
df1
```

```
##   x grp
## 1 1   a
## 2 2   b
## 3 3   c
```

```r
df2
```

```
##   y grp
## 1 4   a
## 2 5   b
## 3 6   d
```

```r
df1 %>% inner_join(df2, by = "grp")
```

```
##   x grp y
## 1 1   a 4
## 2 2   b 5
```

```r
df1 %>% left_join(df2, by = "grp")
```

```
##   x grp  y
## 1 1   a  4
## 2 2   b  5
## 3 3   c NA
```

```r
df1 %>% right_join(df2, by = "grp")
```

```
##    x grp y
## 1  1   a 4
## 2  2   b 5
## 3 NA   d 6
```

```r
df1 %>% full_join(df2, by = "grp")
```

```
##    x grp  y
## 1  1   a  4
## 2  2   b  5
## 3  3   c NA
## 4 NA   d  6
```

```r
df1 %>% semi_join(df2, by = "grp")
```

```
##   x grp
## 1 1   a
## 2 2   b
```

```r
df1 %>% anti_join(df2, by = "grp")
```

```
##   x grp
## 1 3   c
```

Tidy data

- Tidy data = ordelijke gegevens
- Volgens 4 principes
  - Elke observatie vormt een rij
  - Elke variabele vormt een kolom
  - Elke cel bevat een waarde
  - Elk type van observationele eenheid vormt een tabel

Untidy

Tidy

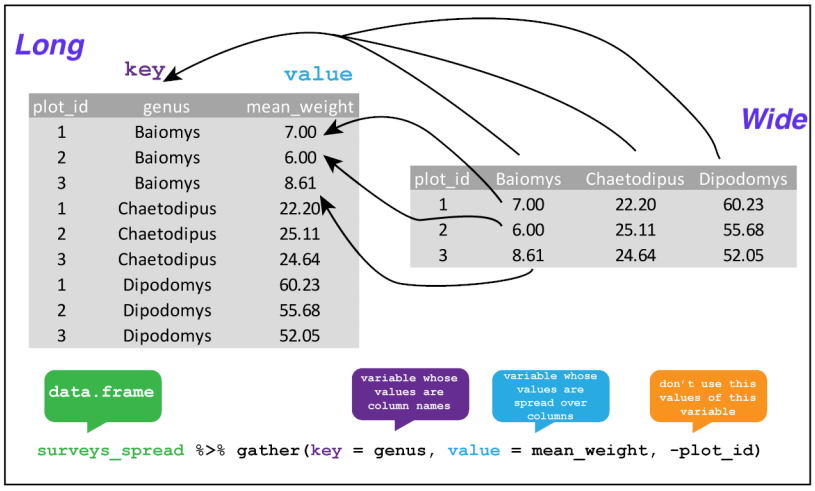
Gebruik functies uit het `tidyr` package (vervat in `tidyverse`) om de gegevens *tidy* te maken.

Data transformeren van breed naar lang `pivot_longer()` (vervangt `gather()`)

- Kolomnamen zijn geen variabelen, maar waarden van een variabele
- Informatie van deze kolommen *verzamelen* in nieuwe variabelen

```
pivot_longer(data, cols, names_to, values_to)
data %>%
   pivot_longer(cols, names_to, values_to)
```

- `cols`: naam van de kolommen waarvoor je de waarden onder elkaar wil, quotes zijn niet nodig, functies zoals by `select()` kan je gebruiken
- `names_to`: naam van de nieuwe variabele die de kolomnamen zal bevatten, indien je dit niet invult kiest R `name`
- `values_to`: naam van de nieuwe variabele die de waarden zal bevatten, indien je dit niet invult kiest R `value`



Voorbeelden

`iris` data

- We tonen deze resultaten voor een subset van de `iris` data (rijen 1, 51 en 101)

```
iris_mini <- iris %>%
  slice(c(1, 51, 101))
iris_mini
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1          5.1         3.5          1.4         0.2     setosa
## 2          7.0         3.2          4.7         1.4 versicolor
## 3          6.3         3.3          6.0         2.5  virginica
```

- Maak de kolommen `Kenmerk` en `Waarde` aan, die alle info bevatten van de 4 `Sepal` en `Petal` variabelen. Let op: `names_to` en `values_to` moeten met quotes gebruikt worden, anders worden de objecten `Meting` en `Waarde` verwacht

```
iris_mini %>%
  pivot_longer(cols = c(Sepal.Length, Sepal.Width,
                        Petal.Length, Petal.Width),
               names_to = "Meting",
               values_to = "Waarde")
```

```
## # A tibble: 12 x 3
##    Species    Meting       Waarde
##    <fct>      <chr>         <dbl>
##  1 setosa     Sepal.Length    5.1
##  2 setosa     Sepal.Width     3.5
##  3 setosa     Petal.Length    1.4
##  4 setosa     Petal.Width     0.2
##  5 versicolor Sepal.Length    7
##  6 versicolor Sepal.Width     3.2
##  7 versicolor Petal.Length    4.7
##  8 versicolor Petal.Width     1.4
##  9 virginica  Sepal.Length    6.3
## 10 virginica  Sepal.Width     3.3
## 11 virginica  Petal.Length    6
## 12 virginica  Petal.Width     2.5
```

- Nieuwe kolommen bevatten info van alle kolommen, behalve van `Species` (idem resultaat)

```
iris_mini %>%
  pivot_longer(cols = -Species, values_to = "Waarde")
```

`pilootstudie.csv` data

- We tonen deze resultaten voor een subset van 4 random rijen uit de `pilootstudie` data

```
piloot_mini <- piloot %>%
  sample_n(4)
piloot_mini
```

- Maak de kolommen `Kenmerk` en `Waarde` aan, die alle info bevatten van de variabelen `Omtrek` en `Hoogte`

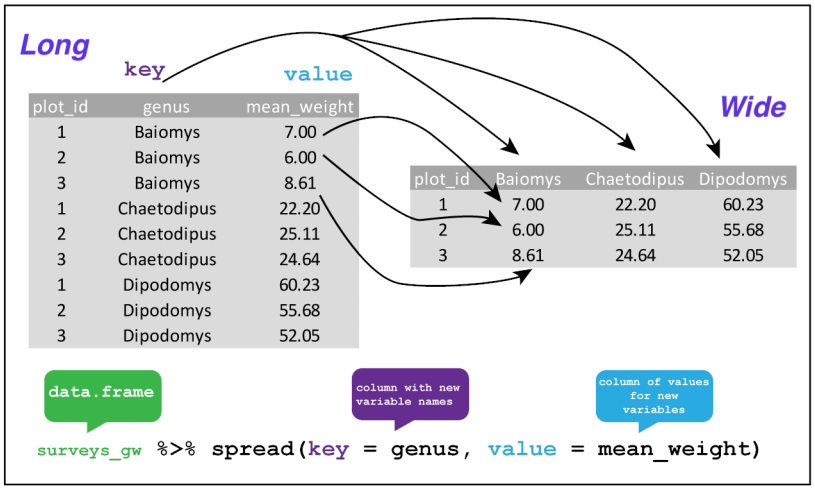
```
piloot_mini %>%
  pivot_longer(names_to = "Kenmerk",
               values_to = "Waarde",
              cols = c(Omtrek, Hoogte))
```

Data transformeren van lang naar breed formaat met `pivot_wider()` (vervangt `spread()`)

- Omgekeerde van `pivot_longer()`
- Observaties zijn verspreid over meerdere rijen, en je wil ze *uitspreiden* over de kolommen

```
pivot_wider(data, id_cols, names_from, values_from)
data %>%
  pivot_wider(id_cols, names_from, values_from)
```

- `id_cols`: de kolomnamen die als rijen moeten behouden worden. Indien je die niet invult, worden alle variabelen behouden die niet in names_from of values_from voorkomen
- `names_from`: de namen van de variabelen
- `values_from`: de variabele die de waarden bevat
- `values_fill`: wat moet gebeuren met records die geen waarden voor elke kolom bevatten



Voorbeelden

`iris` data

- In de lange dataset (uit de voorbeelden van `pivot_longer()`) splitsen we de variabele `Kenmerk` nog op in `Blad` en `Afmeting` en bewaren het resultaat in `iris_lang`.
- Maak nieuwe kolommen die de lengte en breedte van de blaadjes bevatten

```
iris_lang %>%
  pivot_wider(names_from = c(Afmeting, Blad),
              values_from = Waarde)
```

`pilootstudie.csv` data

- We behouden telkens de eerste meting en selecteren de variabelen `Proefvlak`, `Boom`, `Ploeg` en `Omtrek`.

```
## # A tibble: 6 x 4
##   Proefvlak  Boom Ploeg Omtrek
##       <dbl> <dbl> <dbl>  <dbl>
## 1         1     1     1   106
## 2         1     1     2   108
## 3         1     1     3   110
## 4         1     1     4   105
## 5         1     1     5   108.
## 6         1     1     6   107
```

- Spreid de metingen van elke ploeg over verschillende kolommen

```
piloot_lang %>%
  pivot_wider(values_from = Omtrek,
              names_from = Ploeg)
```

```
## # A tibble: 96 x 9
##    Proefvlak  Boom   `1`   `2`   `3`   `4`   `5`   `6`   `7`
##        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1         1     1   106   108   110   105 108.    107 107.
##  2         1     2   109   109   111   109 110     110 109.
##  3         1     3   103   102   103   101 103     103 102.
##  4         1     4   112   114   114   113 114.    114 112.
```

```
##  5         1     5   115   117   118   116 117     117 117.
##  6         1     6    98   100    99    98 100.    100  94.4
##  7         1     7    85    84    86    83  85.5    85  83.8
##  8         1     8    99   103   103   100 102     100  99.8
##  9         1     9    97    99    99    97  99      98  97.1
## 10         1    10   113   115   115   114 114.    114 113.
## # ... with 86 more rows
```

More to learn

- R for data science (Hoofdstuk 5 en 12)
  - Boek van Hadley Wickham en Garrett Grolemund
  - Hardcopy beschikbaar op INBO
  - Digitale versie
- Datacamp
  - (gedeeltelijk) gratis lessen (video tutorials en oefeningen)
  - Account voor 72h voor volledige toegang, daarna betalende licentie (~ €25/maand)
  - Introduction to the Tidyverse
  - Data Wrangling
- Data Carpentry
  - Manipulating, analyzing and exporting data with tidyverse
- Stat 545
  - Introduction to dplyr
  - dplyr functions for a single dataset
- INBO Coding Club
  - Tidy data
- Cheat Sheets
  - In RStudio onder `Help` menu
  - Online

Referenties

- Transform data with dplyr
- R for data science