# Introductie ggplot2

A picture is worth a thousand words

## Data visualiseren met ggplot


### Waarom ggplot?

- `ggplot` maakt grafieken gebaseerd op de Grammar of Graphics
- Werkt altijd volgens dezelfde structuur
  - geef de dataset aan die je wil plotten
  - geef aan welke variabelen op de assen moeten komen
    - de assen x en y
    - de assen color, fill, shape, ...
    - geen z as
  - geef aan welk type grafiek je wil maken (punten, lijnen, boxplot, histogram, ...)
  - detailleer je grafiek (titels, kleuren, vormen, ...)
  - toon of print de grafiek
- De basisgrafieken zien er al goed uit
- Grafieken worden stap voor stap opgebouwd
- Gemakkelijk om kleuren, groottes, ... te laten variëren per groep
- Gemakkelijk je plot op te splitsen per groep
- Automatische legende
- Mogelijk om verschillende datasets te combineren
- Alle informatie om de figuur te maken wordt in een R object bewaard
- Figuren exporteren in de meeste formaten (gebruik van png wordt aangeraden)
- ...

### Datasets

- `iris` data uit R
  - Opgesplitst in `irisSepal` en `irisPetal`
  - Variabelen hernoemd naar `Length` en `Width`
  - Extra variabele `Leaf.Type` aangemaakt, gelijk aan `sepal` of `petal`
  - Beide datasets onder mekaar geplakt en bewaard in `irisAll`
  - Deze 3 datasets zijn bewaard als `R` objecten, en worden ingelezen met de functie `load()`

```
load("data/mijnIris.Rdata")
```

## Package ggplot2 om de figuren te maken

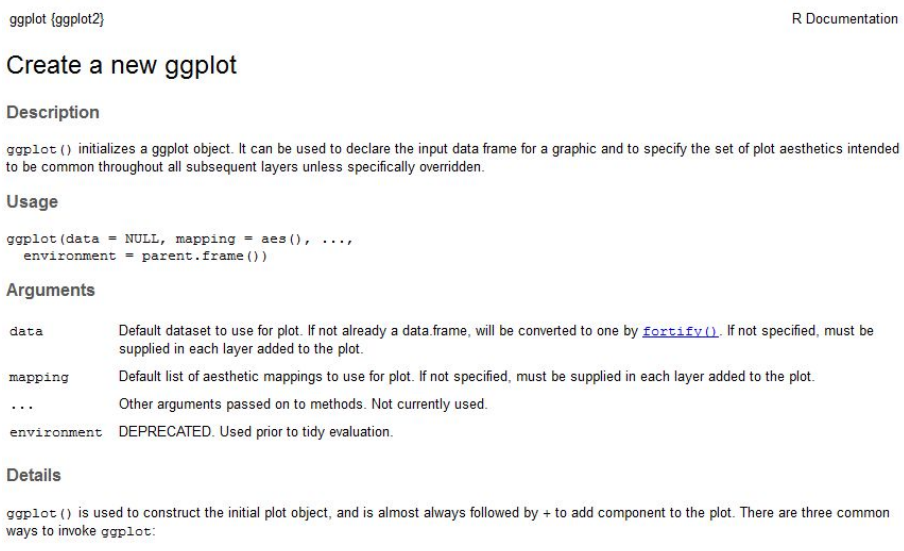
- Dit package heb je al als je het package `tidyverse` geïnstalleerd hebt
- Laad het package `tidyverse` (of alleen `ggplot2`)

```
# library(ggplot2)
library(tidyverse)
```

- Je ziet dat er een hele reeks packages geladen worden
- Er zijn 2 conflicten
  - De functie `filter()` uit het package `dplyr` overschrijft de functie `filter()` uit het `stats` package
  - De functie `lag()` uit het package `dplyr` overschrijft de functie `lag()` uit het `stats` package
  - Geen probleem, maar wees je ervan bewust wanneer je hulp zoekt over deze functies, dat je die uit het juiste package (`dplyr`) bestudeert (`R` geeft het aan als er verschillende mogelijkheden zijn)

## Basis syntax ggplot

- Het package heet `ggplot2`, de belangrijkste functie is `ggplot()`
- Zoek hulp over `ggplot` (?ggplot)



ggplot {ggplot2} R Documentation

Create a new ggplot

Description

`ggplot()` initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

Usage

```
ggplot(data = NULL, mapping = aes(), ...,
  environment = parent.frame())
```

Arguments

| | |
|---|---|
| `data` | Default dataset to use for plot. If not already a data.frame, will be converted to one by `fortify()`. If not specified, must be supplied in each layer added to the plot. |
| `mapping` | Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot. |
| `...` | Other arguments passed on to methods. Not currently used. |
| `environment` | DEPRECATED. Used prior to tidy evaluation. |

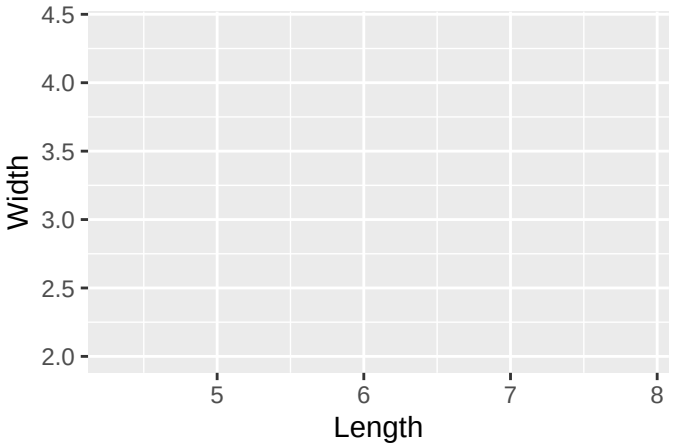
Details

`ggplot()` is used to construct the initial plot object, and is almost always followed by + to add component to the plot. There are three common ways to invoke `ggplot`:

`ggplot()` initialiseert de componenten van het `ggplot` object

- `data = NULL`: de dataset die gebruikt wordt voor de grafiek
- `mapping = aes()`: de aesthetics (*assen*), variabelen in de dataset

```
# ggplot(data = irisSepal, mapping = aes(x = Length, y = Width))
ggplot(irisSepal, aes(x = Length, y = Width))
```

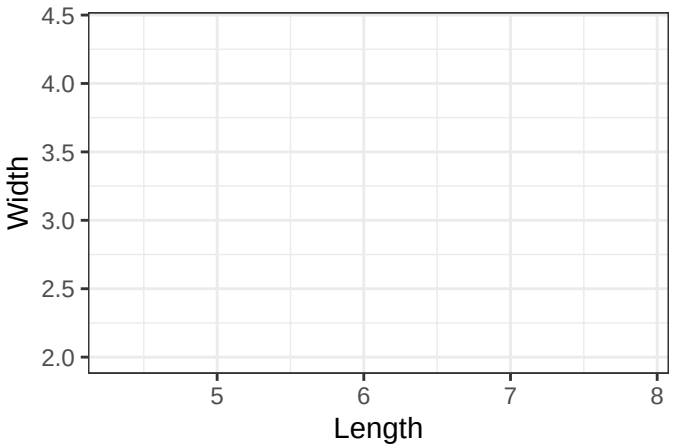



- Blanco figuur
- Enkel dataset gespecifieerd, en X- en Y-as
- Daarna worden er 1 of meerdere lagen met punten, lijnen,... toegevoegd volgens diezelfde X- en Y-assen, met `+ geom_xxx()`.

GGplot figuren gaan uit van een basisthema

- Dit bevat heel wat zaken
  - hoe de assen, astitels en achtergrond eruit zien
  - welke de basiskleuren zijn die gebruikt worden
- Standaard wordt het `theme_grey()` gebruikt, maar er zijn alternatieven zoals `theme_bw()`
- Je kan ook eigen thema's definiëren en gebruiken, zo bestaat er een thema voor INBO figuren
  - hiervoor gebruik je het package `INBOtheme` (komt in de markdown les aan bod)
  - installeer en laadt het pakket
  - `+ theme_inbo(base_size = 12)` voor INBO thema met basis font size van 12
  - bovenaan je script kan je ook het default thema wijzigen zodat je dit niet telkens voor iedere figuur moet specifiëren: `theme_set(theme_inbo(base_size = 12))`

```
# gebruik het zwart wit thema
ggplot(irisSepal, aes(x = Length, y = Width)) + theme_bw()
```




```
#als je in de console theme_ invult gevolgd door TAB zie je andere thema's
```

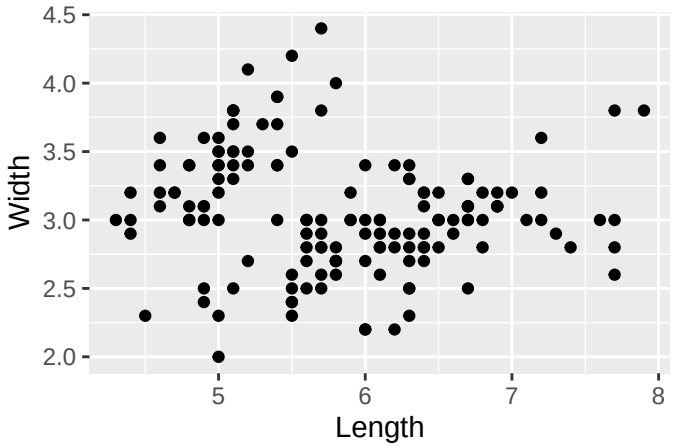
## Lagen toevoegen

Een laag wordt quasi altijd via een geom_xxx of stat_xxx gecodeerd, waarbij xxx van alles kan zijn zoals point, line, ribbon, ...

### Punten

- `geom_point`

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point()
```



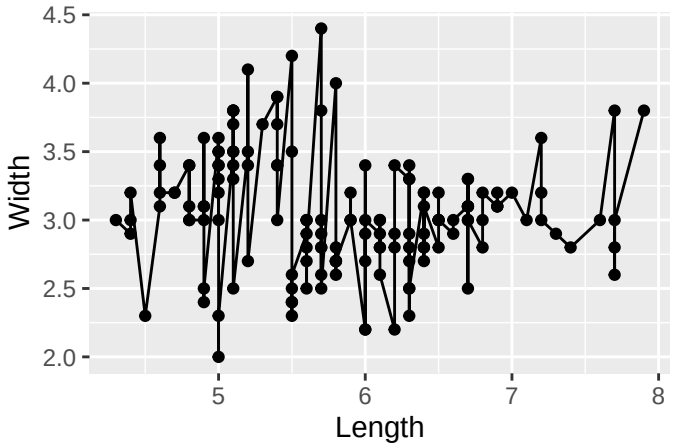

- `geom_jitter`: voegt wat ruis toe, interessant wanneer er veel punten over mekaar vallen

### Lijnen

`geom_line`

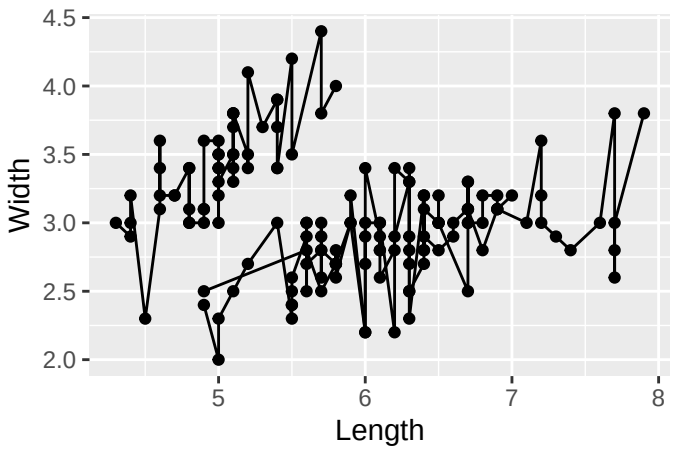
- Verbindt de punten volgens de waarden op de X-as

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_line()
```




- Eventueel groeperen per soort met het `group` aesthetic

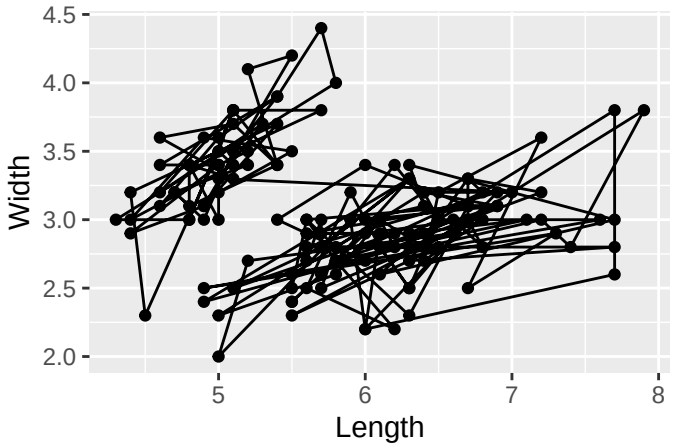
```
ggplot(irisSepal, aes(x = Length, y = Width, group = Species)) +
  geom_point() +
  geom_line()
```




**geom_path**

- Verbindt de punten volgens de volgorde in de data

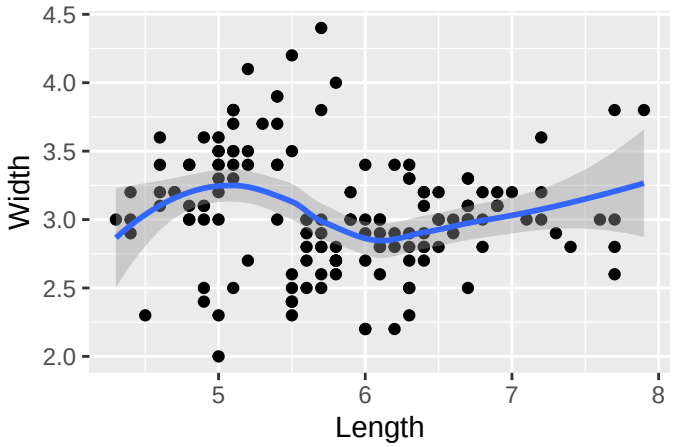
```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_path()
```




**geom_smooth**

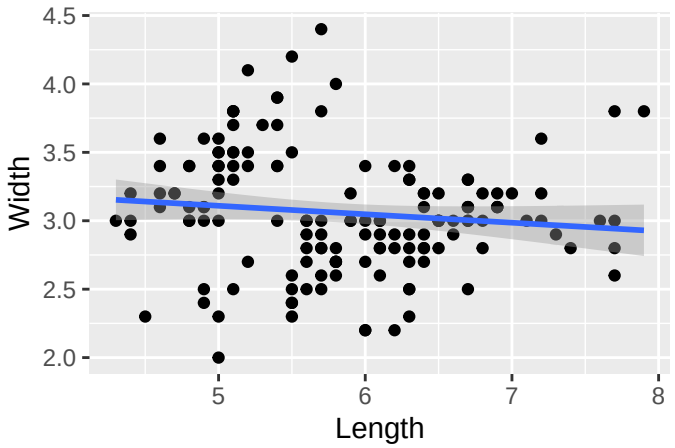
- Smoother zonder verdere specificatie, om een patroon te herkennen in de punten

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth()
```




- Lineaire smoother met `method = "lm"`

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth(method = "lm")
```




- Veel meer opties mogelijk, zie help of geef `geom_` gevolgd door TAB om een lijst te zien

### Andere lijnen

- `geom_hline`: horizontale lijn
- `geom_vline`: verticale lijn
- `geom_abline`: hellende lijn
- `geom_ribbon`: gekleurd vlak rond een lijn
- `geom_errorbar`: foutenvlaggen

## Aesthetics

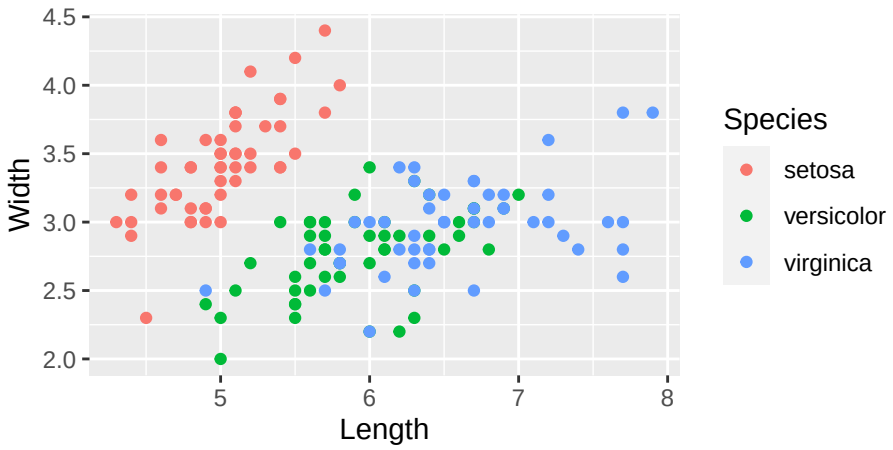
Naast de X- en Y-as kunnen we nog extra *assen* toevoegen aan de figuur:

- meest gebruikt: `color`, `shape`, `linetype`, `size`, `fill`,
- andere: `alpha`, `label`, `family`, `fontface`, ...
- Specifiëer je deze argumenten **binnen** de `aes()`, dan variëren ze volgens een bepaalde variabele (kenmerk van de observaties)
- Specifiëer je deze argumenten **buiten** de `aes()`, dan zijn het vaste kenmerken
- Dit kan voor de volledige grafiek hetzelfde, of verschillend per laag, afhankelijk in welke aesthetics `aes()` je dit argument toevoegt

`color`

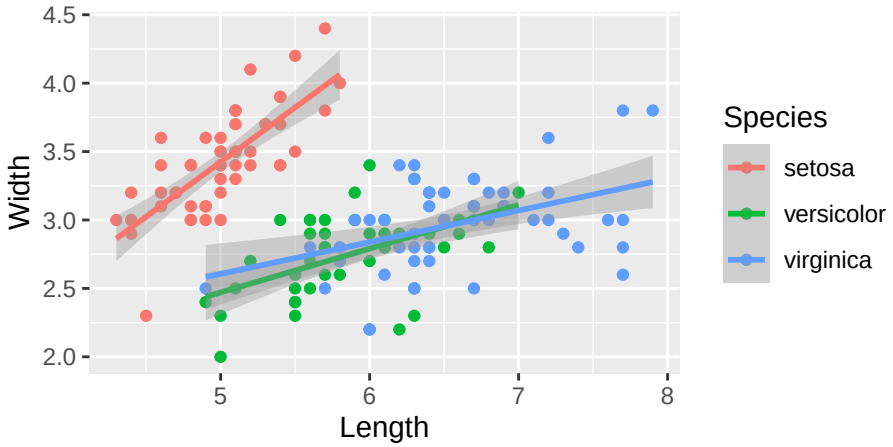
- Kleur van de punten variërend volgens soort

```
ggplot(irisSepal, aes(x = Length, y = Width, color = Species)) +
  geom_point()
```




- Kleur van de punten en lijnen variërend volgens soort

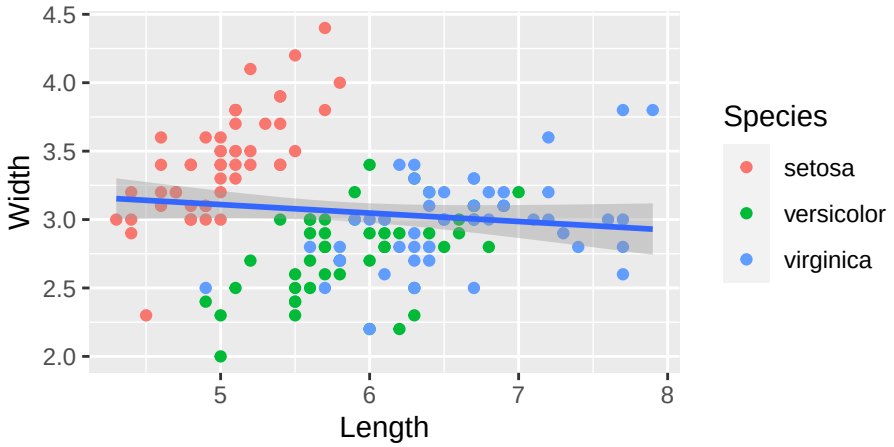
```
ggplot(irisSepal, aes(x = Length, y = Width, color = Species)) +
  geom_point() +
  geom_smooth(method = "lm")
```




  - Merk op dat we nu 3 verschillende smoothers krijgen, een per soort

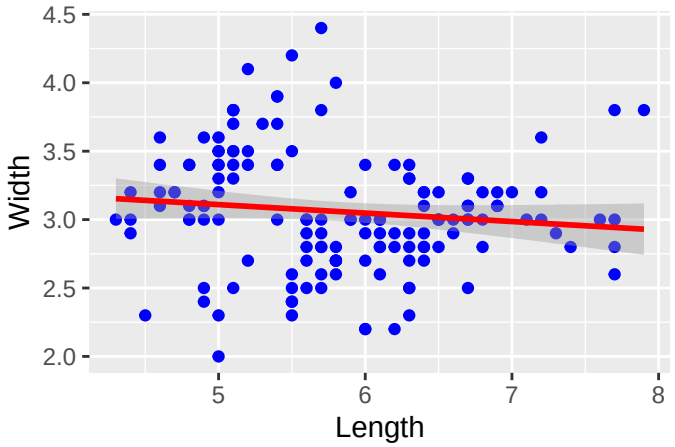
- Effect van plaatsing `color` aesthetic

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point(aes(color = Species)) +
  geom_smooth(method = "lm")
```




- Beperk het aantal kleuren (categorieën), anders zijn deze nog moeilijk te onderscheiden
- Vaste kleur, bvb blauwe punten en rode lijn

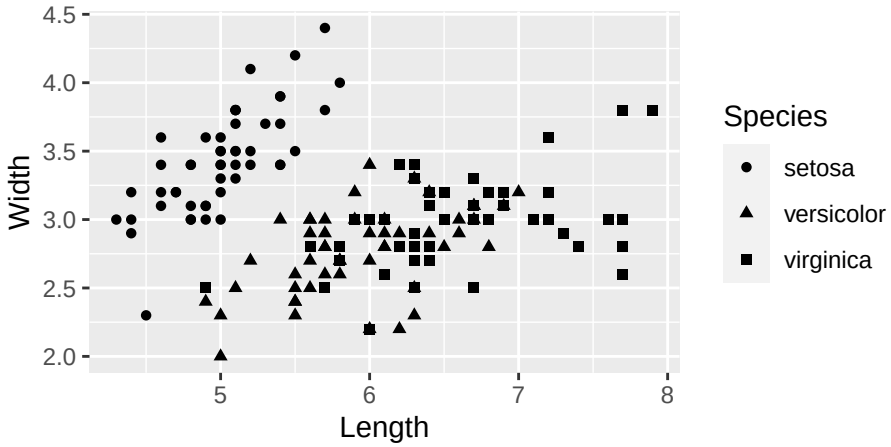
```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point(color = "blue") +
  geom_smooth(color = "red", method = "lm")
```




**shape**

- Symbool van de punten variërend volgens soort

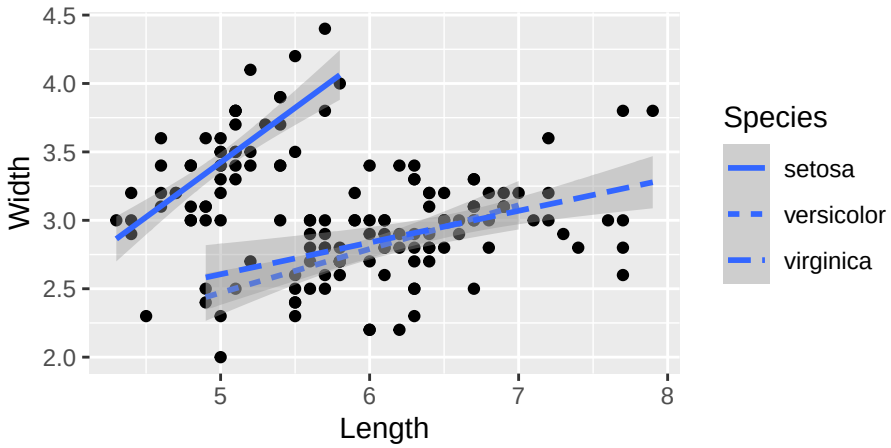
```
ggplot(irisSepal, aes(x = Length, y = Width, shape = Species)) +
  geom_point()
```




**linetype**

- Type lijn varieert volgens soort

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth(aes(linetype = Species), method = "lm")
```

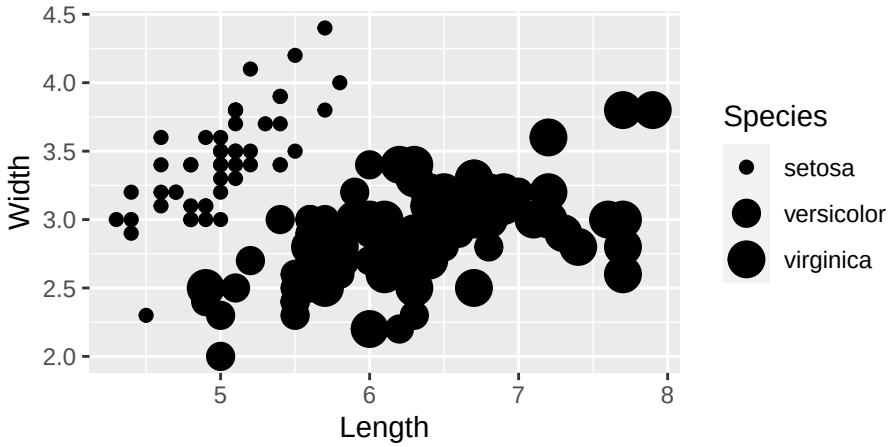



**size**

- Grootte van de punten variërend volgens soort

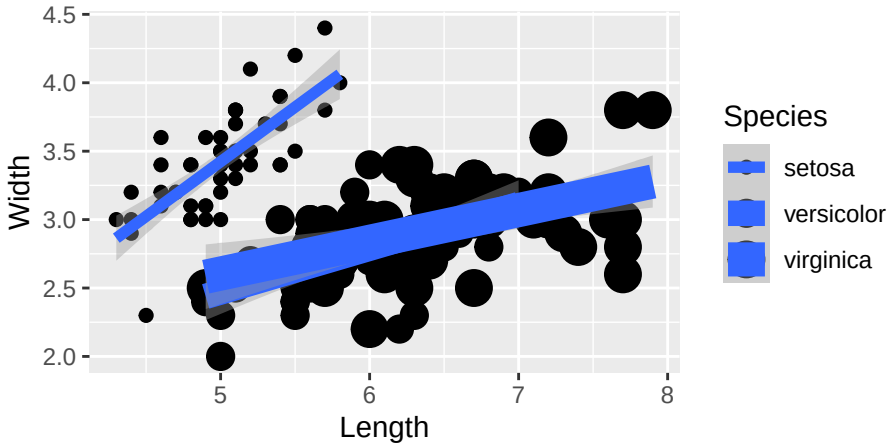
```
ggplot(irisSepal, aes(x = Length, y = Width, size = Species)) +
  geom_point()
```

```
## Warning: Using size for a discrete variable is not advised.
```



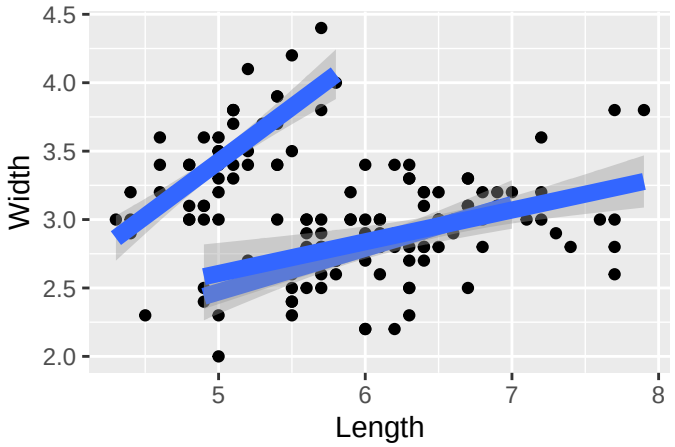

- **Waarschuwing** dat het gebruik van `size` niet aangewezen is voor een discrete variabele
- Grootte van de punten en dikte van de lijnen variërend volgens soort

```
ggplot(irisSepal, aes(x = Length, y = Width, size = Species)) +
  geom_point() +
  geom_smooth(method = "lm")
```




- Dikkere lijn, maar hetzelfde voor elke soort
  - Aangegeven door een getal, in millimeter

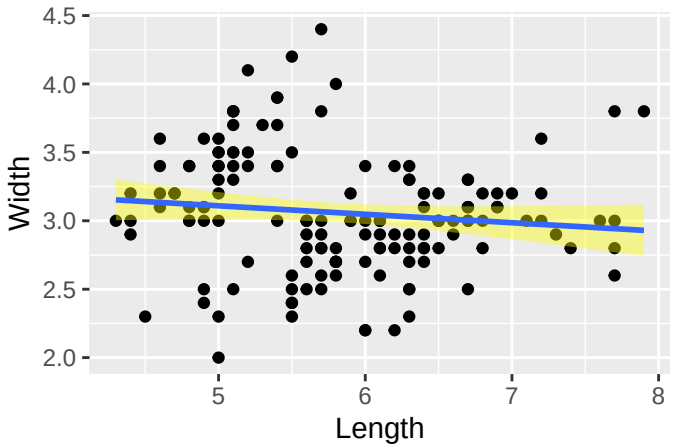
```
ggplot(irisSepal, aes(x = Length, y = Width, group = Species)) +
  geom_point() +
  geom_smooth(size = 3, method = "lm")
```



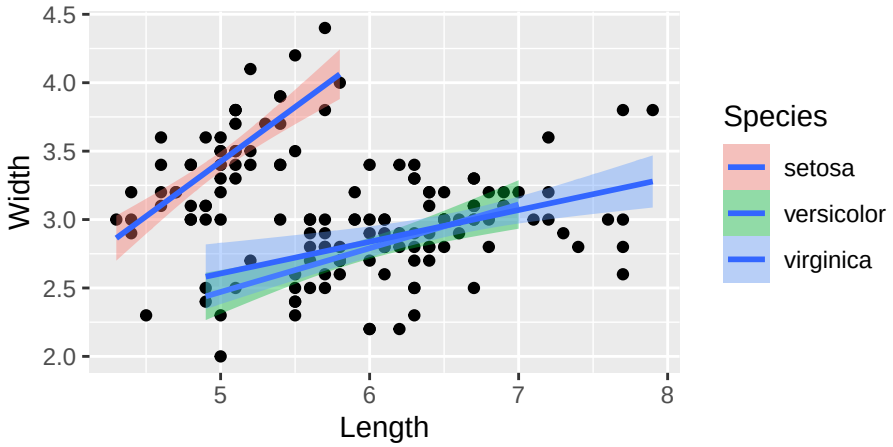

`fill`

- Hiermee kan je de kleur van het (interne) vlak bepalen
- Met `color` verander je enkel de kleur van de rand
- Punten en lijnen hebben enkel `color`, geen `fill`
- We kunnen dit wel gebruiken voor het betrouwbaarheidsinterval rond de smoother

```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth(fill = "yellow", method = "lm")
```




```
ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth(aes(fill = Species), method = "lm")
```

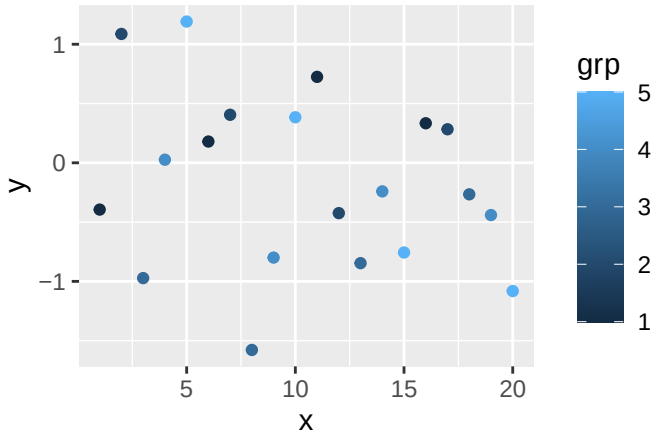



**factorvariabelen**

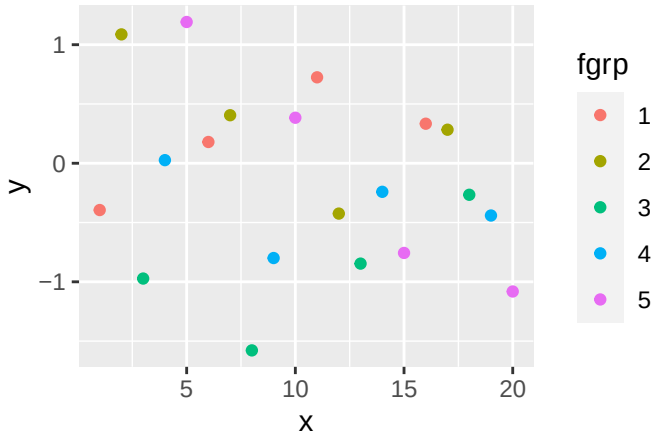
**tip: converteer numerieke waarden naar factor als je ze gebruikt voor groepering**

- Indien een variabele numeriek is zal ggplot deze als een continue range beschouwen
- Indien je een discrete range wil, maak je van de variabele een factor
  - factoren laten toe de volgorde te kiezen met het argument `levels`
  - met het `labels` argument kan je ieder level een naam geven

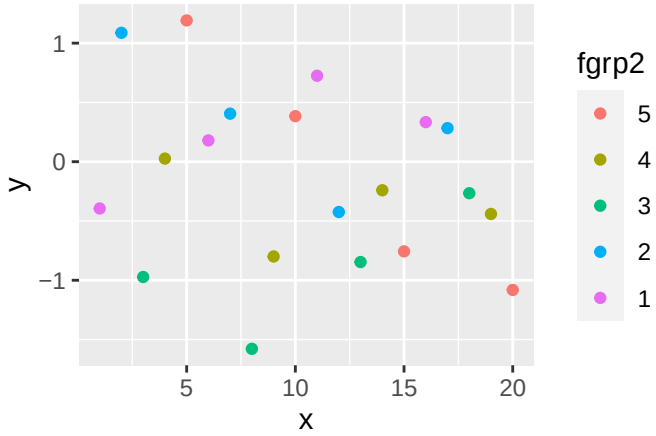
```
testdata <- data.frame(x = 1:20, y = rnorm(20), grp = rep(1:5, 4))
ggplot(testdata, aes(x = x, y = y, color = grp)) + geom_point()
```



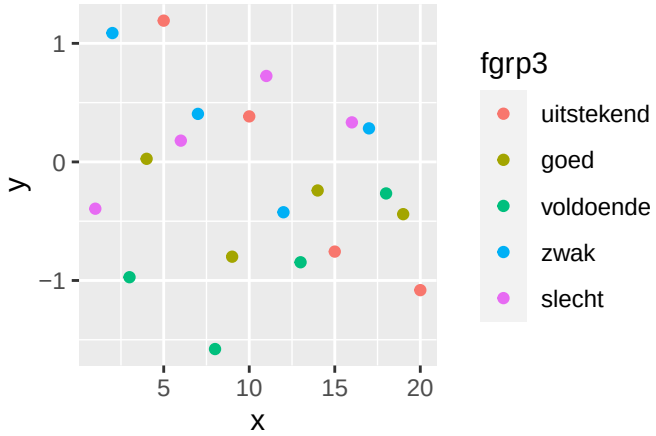

```
testdata$fgrp <- factor(testdata$grp)
ggplot(testdata, aes(x = x, y = y, color = fgrp)) + geom_point()
```




```
testdata$fgrp2 <- factor(testdata$grp, levels = 5:1)
ggplot(testdata, aes(x = x, y = y, color = fgrp2)) + geom_point()
```




```
testdata$fgrp3 <- factor(testdata$grp, levels = 5:1,
                         labels = c("uitstekend", "goed",
                                    "voldoende", "zwak", "slecht"))
ggplot(testdata, aes(x = x, y = y, color = fgrp3)) + geom_point()
```

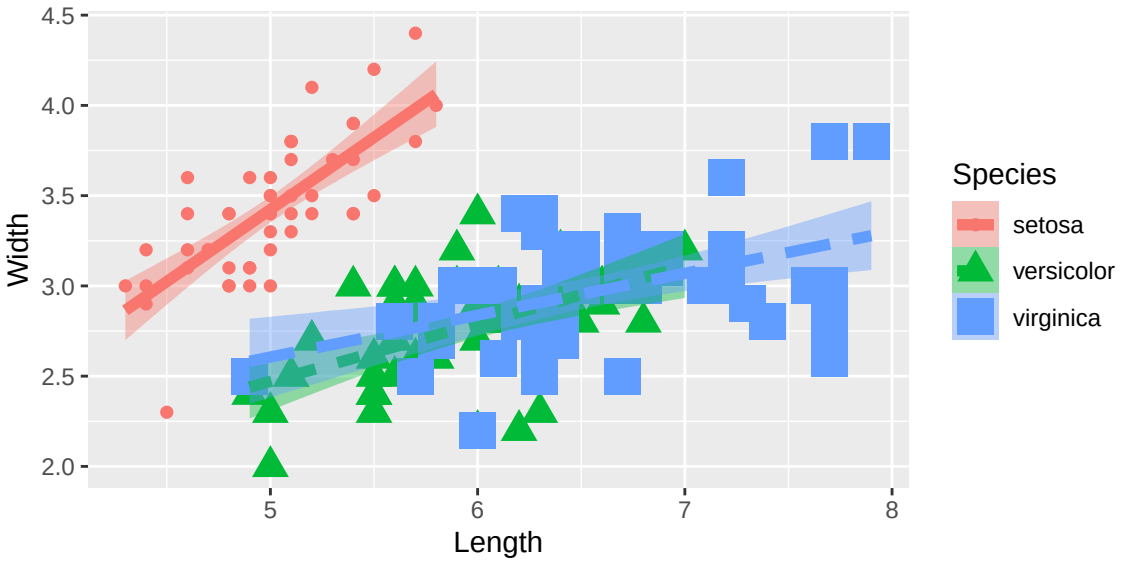



### Combinatie van aesthetics

- Verschillende kleuren voor de punten en smoothers

- Verschillend symbool voor de punten
- Verschillende grootte voor de punten
- Verschillend lijntype
- Verschillende kleur betrouwbaarheidsinterval rond de smoothers
- Dikte van de smoothers vast op 2mm

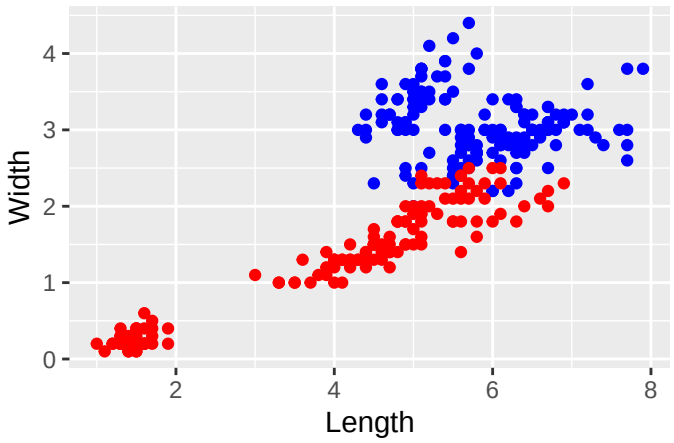
```
ggplot(irisSepal, aes(x = Length, y = Width, color = Species)) +
  geom_point(aes(shape = Species, size = Species)) +
  geom_smooth(aes(linetype = Species, fill = Species), size = 2, method = "lm")
```




**Lagen met verschillende data**

- Het is mogelijk in elke laag een verschillende dataset en/of aesthetics te gebruiken
- Beperk de informatie in de `ggplot()` functie dan tot het gemeenschappelijke
- Specifieer de rest in de afzonderlijke `geom_xxx()`
  - **Opgelet**: hier moet je het data argument expliciet benoemen met `data =`
- "Stom" voorbeeld dat op een mooiere manier kan, slechts ter illustratie

```
ggplot() +
  geom_point(data = irisSepal, aes(x = Length, y = Width), color = "blue") +
  geom_point(data = irisPetal, aes(x = Length, y = Width), color = "red")
```



```
ggplot(mapping = aes(x = Length, y = Width)) +
  geom_point(data = irisSepal, color = "blue") +
  geom_point(data = irisPetal, color = "red")
```

Ter info, je kan binnen een geom het argument `inherit.aes = FALSE` gebruiken wat ervoor zogt dat enkel de aes van de geom gebruikt worden en niet deze die je binnen ggplot() definieerde.

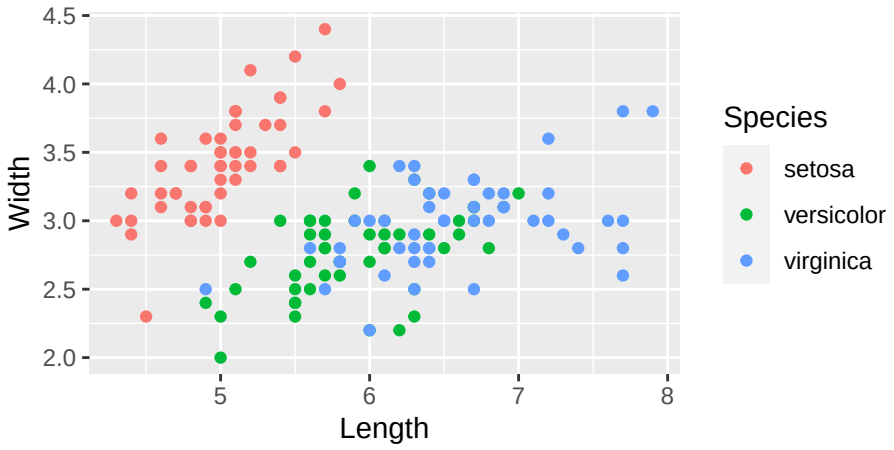
**Conclusie**

- Aesthetics in de `ggplot()` functie hebben een effect op alle lagen die toegevoegd worden
- Aesthetics in een `geom_xxx()` functie hebben enkel effect op die laag
- Aesthetics in `aes()` variëren volgens een variabele in de data
- Aesthetics buiten `aes()` zijn vast, en worden gespecifieerd met een getal of een naam
- Let ook op de aan- of aanwezigheid van een legende
  - Legende voor `color`, `shape`, `linetype`, `size`, `fill` die binnen `aes()` staan
  - Geen legende voor kleuren en groottes buiten `aes()`
- Zet in de `geom_xxx()` altijd eerst de `aes()`, en daarna pas andere opties
- Lagen worden over mekaar gelegd
  - Volgorde belangrijk voor de zichtbaarheid
  - Soms beter om de volgorde te wijzigen
- Ter info
  - ggplot voorziet niet om quasi 3D grafieken te maken
  - Meerdere Y-assen zijn mogelijk, maar afgeraden
  - Pie charts kunnen in ggplot gemaakt worden door met polaire coördinaten te werken (valt buiten de cursus)

## Object stapsgewijs opbouwen

- R kan een ggplot figuur in een object bewaren
- Kan handig zijn om gemeenschappelijke delen niet telkens te moeten herhalen

```
p <- ggplot(irisSepal, aes(x = Length, y = Width)) +
  geom_point(aes(color = Species))
p
```




- Figuur kan dan stapsgewijs verder opgebouwd worden met geschikte lagen
  - Figuur tonen

```
p + geom_smooth()
```

- Bewaren in een ander object

```
p1 <- p + geom_smooth(method = "lm")

#p1 wordt niet getoond daarvoor moet je p1 printen via
p1
#ofwel
print(p1)
```

- Bewaren in hetzelfde object (overschrijven)

```
p <- p + geom_smooth(aes(color = Species, fill = Species), method = "lm")
```

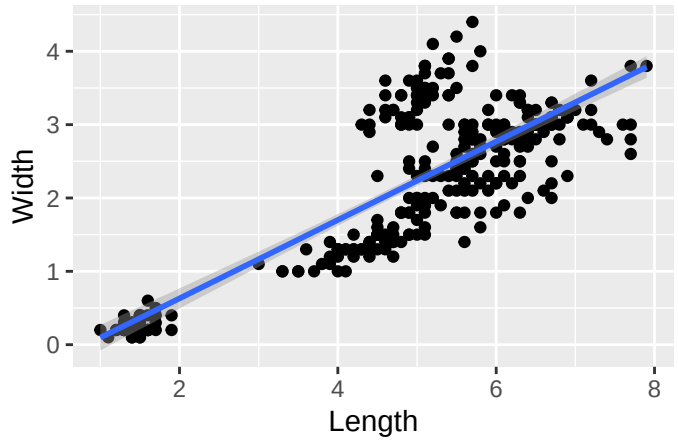
Dus als je de figuur in een object bewaart, wordt de figuur niet getoond, je kan die achteraf tonen door de figuurnaam te typen of `print(figuurobjectnaam)`

## Facets

- Gebruik van verschillende kleuren kan verwarrend zijn, zeker als er veel punten en/of lijnen op 1 grafiek staan
- Duidelijker als gegevens over verschillende deelfiguren weergeven worden
- Opsplitsen volgens een of meerdere (categorische) variabelen
- Elke deelfiguur heeft dezelfde definitie
- `facet_wrap()`: vul het raster doorlopend
- `facet_grid()`: vul het raster zoals een tabel, waarbij in de rijen 1 variabele staat en in de kolommen een andere

We definiëren onderstaande basisplot `p` die we vanaf nu telkens gaan aanvullen.

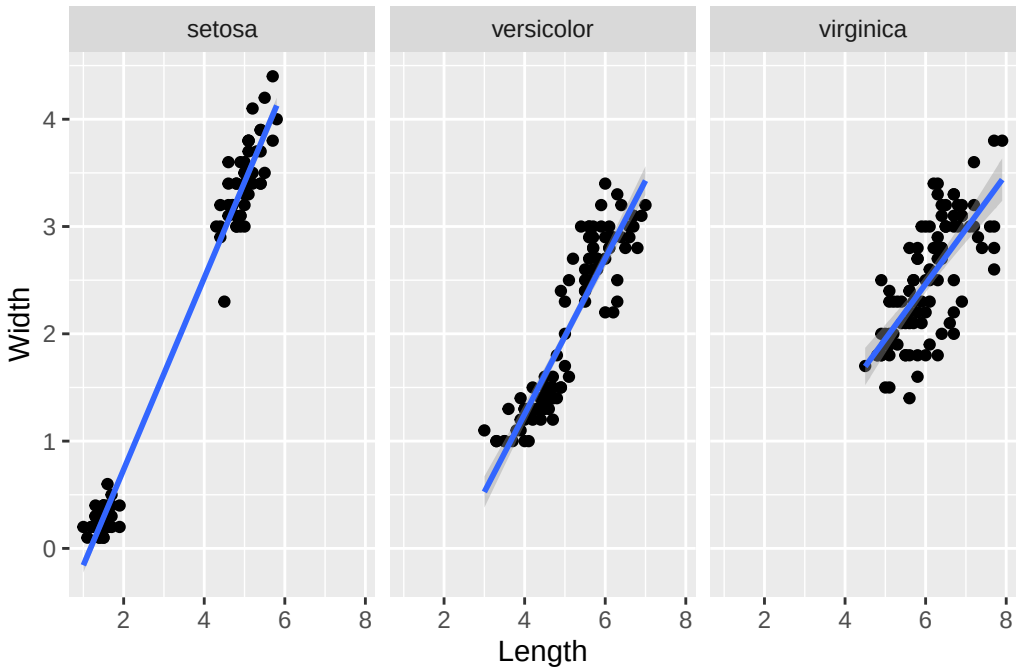
```
#de plot wordt bewaard in object p, maar om het te tonen moet het object tonen
#ofwel een regel met enkel de objectnaam ofwel print(p)
p <- ggplot(irisAll, aes(x = Length, y = Width)) +
  geom_point() +
  geom_smooth(method = "lm")

print(p)
```



`facet_wrap()`

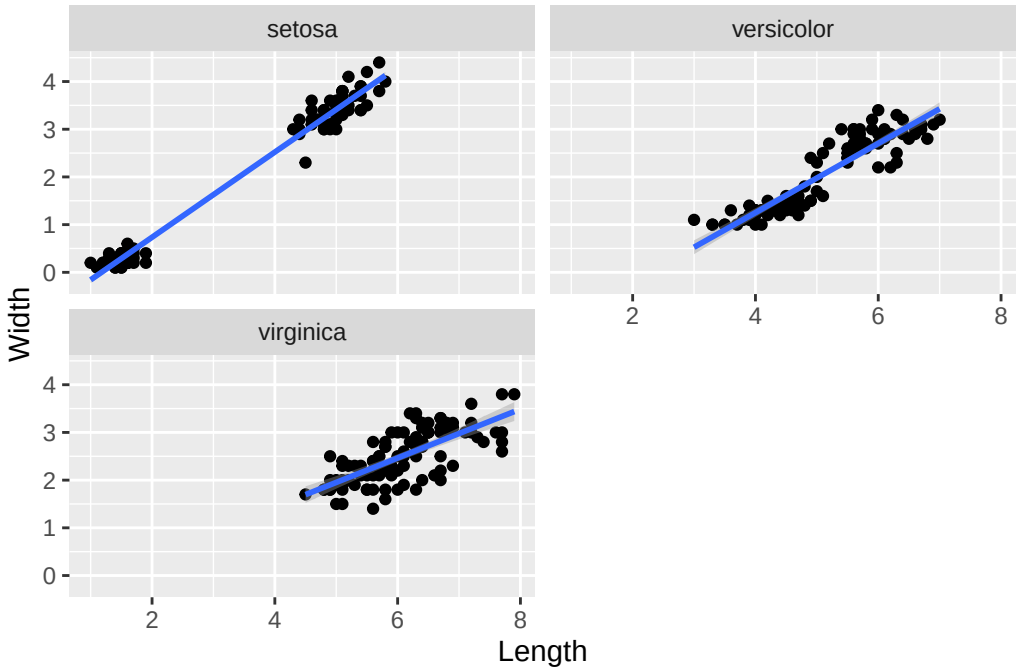
- Basisnotatie: `facet_wrap(~ NaamVariabele)`

```
p + facet_wrap(~ Species)
```




- `nrow` en `ncol`: gewenste aantal rijen en kolommen

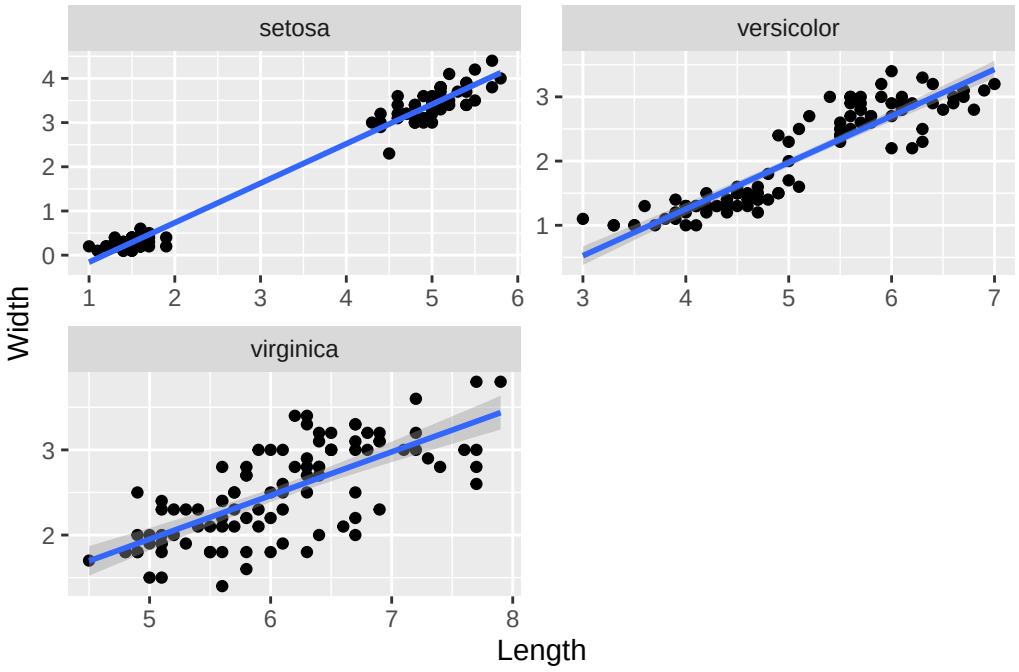
```
p + facet_wrap(~ Species, nrow = 2)
```




- `scales`
  - default: elke subplot heeft zelfde x en y-as

  - `scales = "free_x"` elke subplot heeft aangepaste x-as
  - `scales = "free_y"` elke subplot heeft aangepaste y-as
  - `scales = "free"` elke subplot heeft aangepaste x en y-as

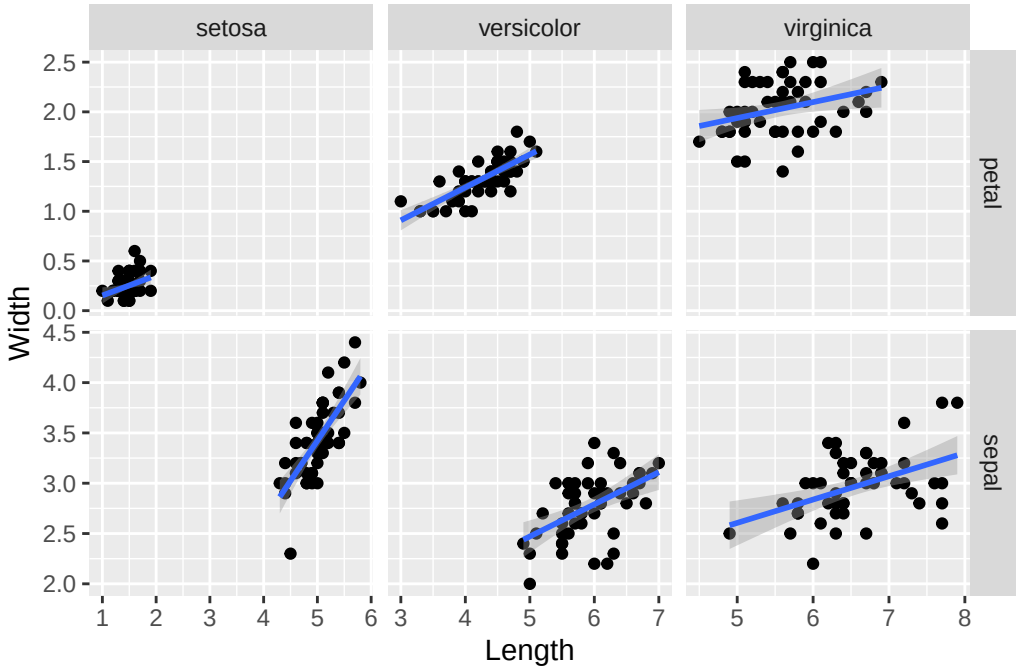
```
p + facet_wrap(~ Species, nrow = 2, scales = "free")
```




`facet_grid()`

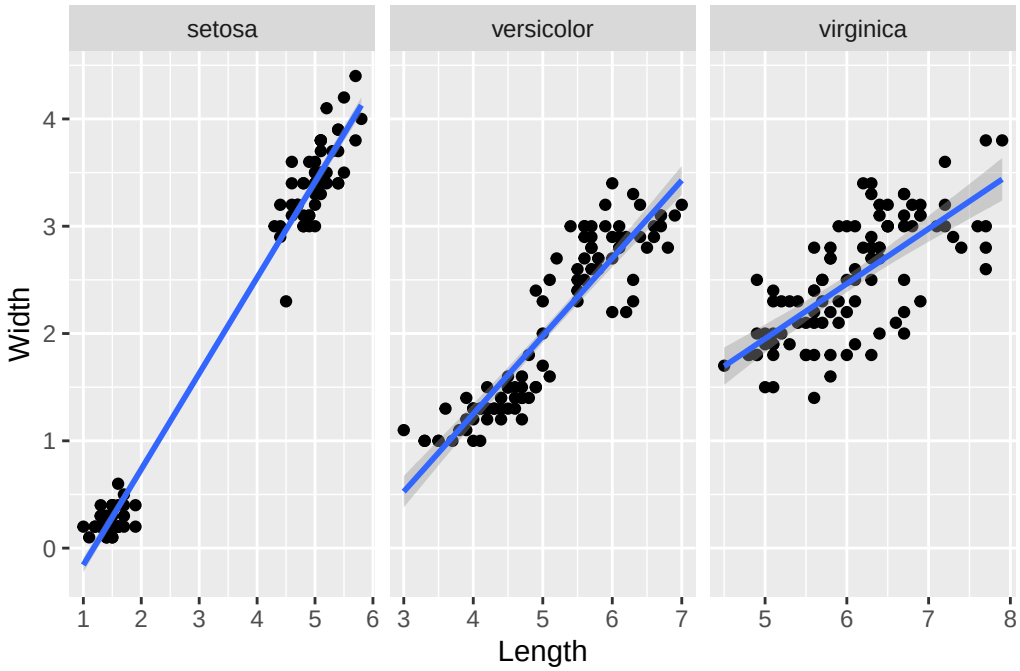
- Basisnotatie: `facet_grid(RijVariabele ~ KolomVariabele)`
- `scales`
  - default: elke subplot heeft zelfde x en y-as
  - `scales = "free_x"` elke **kolom** subplots heeft aangepaste x-as
  - `scales = "free_y"` elke **rij** subplots heeft aangepaste y-as
  - `scales = "free"` elke **kolom** en **rij** subplots heeft aangepaste x en y-as

```
p + facet_grid(Leaf.Type ~ Species, scales = "free")
```



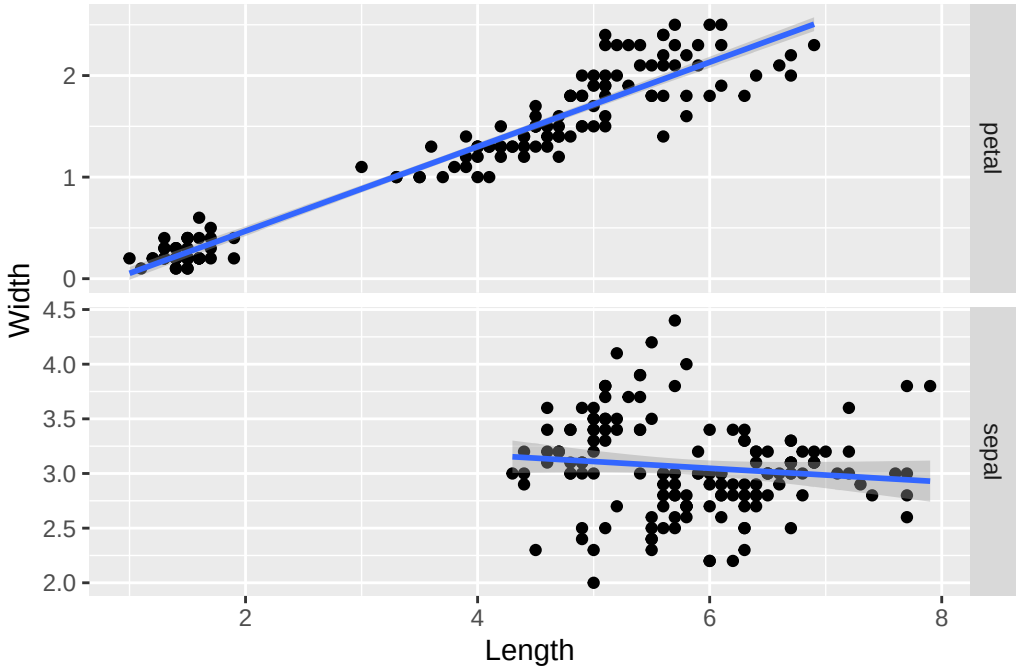

- Gebruik `facet_grid(. ~ A)` als je enkel in kolommen wilt splitsen
  - Dit geeft (bijna) hetzelfde resultaat als `facet_wrap` met `nrow = 1`

```
p + facet_grid(. ~ Species, scales = "free")
```




- Gebruik `facet_grid(A ~ .)` als je enkel in rijen wilt splitsen
  - Dit geeft (bijna) hetzelfde resultaat als `facet_wrap` met `ncol = 1`

```
p + facet_grid(Leaf.Type ~ ., scales = "free")
```



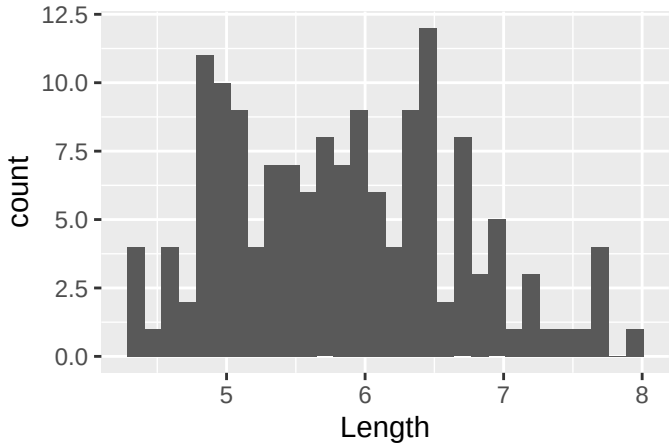

## Nog enkele handige `geom_xxx()`

- `geom_histogram`, `geom_density`: histogram en histogram density
- `geom_bar`, `geom_col`: staafdiagram
  - `geom_bar`: hoogte proportioneel tot aantal observaties in die groep
  - `geom_col`: hoogte proportioneel tot waarde in data
- `geom_boxplot`: boxplot
- `geom_errorbar`, `geom_errorbarh`: foutenvlaggen verticaal en horizontaal
  - `geom_ribbon`: band met betrouwbaarheidsinterval
- `geom_text`: tekst labels per datapunt
- `geom_tile`, `geom_contour`: bovenaanzicht 3D oppervlak

**`geom_histogram`**

```
ggplot(irisSepal, aes(x = Length)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

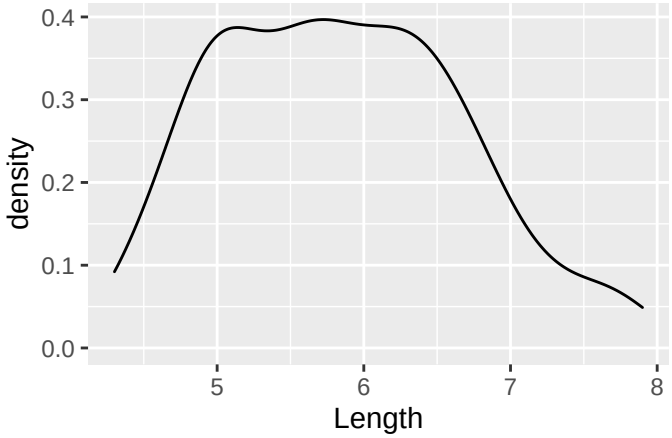



- Enkel een X-as nodig
- Waarden op de Y-as worden berekend
- Hiervoor wordt de X-as onderverdeeld in 30 `bins`, en aantallen geteld in deze intervallen
- Met `bins` of `binwidth` kan een betere keuze gemaakt worden voor deze intervallen (zie message)

```
geom_histogram(bins = 10)
geom_histogram(binwidth = 0.25)
```

**geom_density**

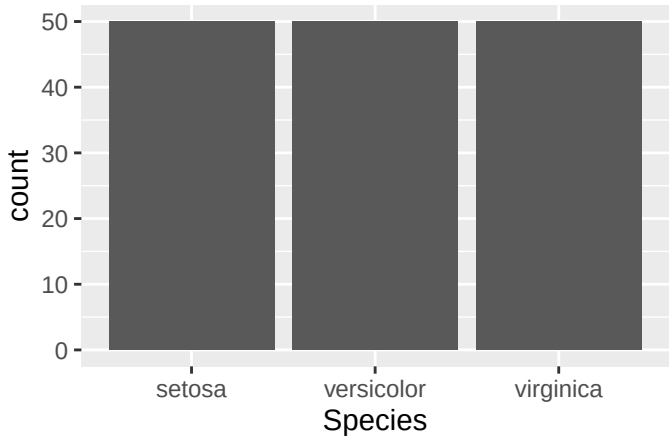
```
ggplot(irisSepal, aes(x = Length)) +
  geom_density()
```



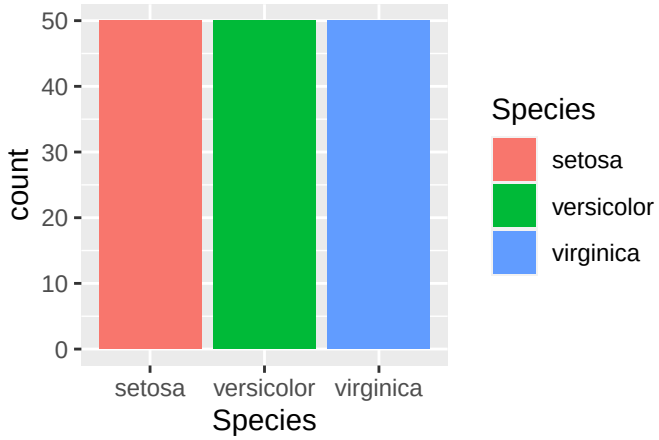

- Enkel een X-as nodig
- Waarden op de Y-as worden berekend
  - behalve als je stat = “identity” gebruikt, dan worden de waarden gebruikt die je als Y-as meegeeft

**geom_bar**

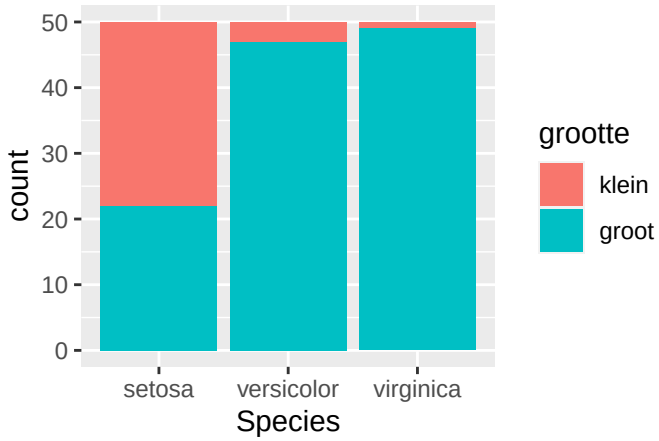
```r
#Toont het aantal irissen per soort in de dataset (is hier 3 keer 50)
ggplot(irisSepal, aes(x = Species)) +
  geom_bar()
```



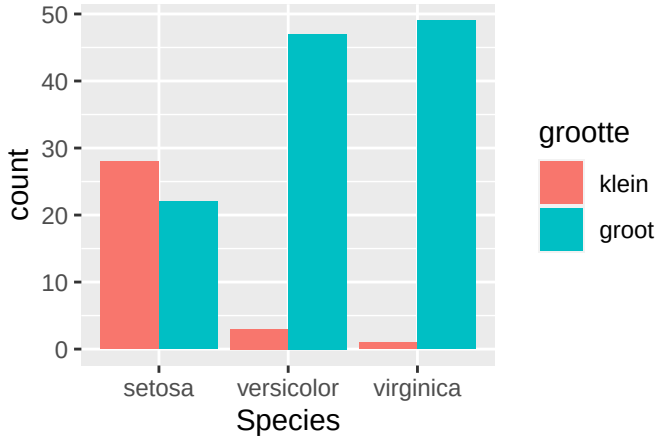

```r
ggplot(irisSepal, aes(x = Species, fill = Species)) +
  geom_bar()
```



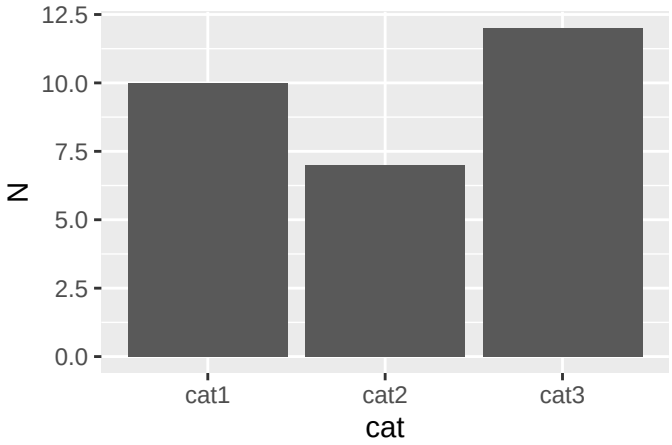

```r
#Maak een extra variabele aan die grote en kleine irissen onderscheid
irisSepal$grootte <- factor(iris$Sepal.Length > 5, labels = c('klein', 'groot'))

#plot deze boven elkaar
ggplot(irisSepal, aes(x = Species, fill = grootte)) +
  geom_bar(position = position_stack())
```




```
#plot deze naast elkaar
ggplot(irisSepal, aes(x = Species, fill = grootte)) +
  geom_bar(position = position_dodge())
```



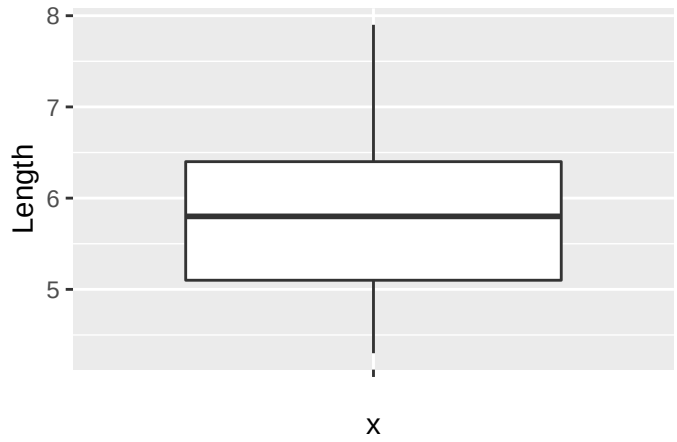

```
#soms heb je de aantallen al op voorhand berekend
#dan moet je dit aantal als Y as meegeven
#en aan R zeggen dat je de waarden zelf wil gebruiken door stat = "identity"
testdata <- data.frame(cat = c("cat1", "cat2", "cat3"), N = c(10,7,12))
ggplot(testdata, aes(x = cat, y = N)) + geom_bar(stat = "identity")
```




- Enkel een X-as nodig
- Waarden op de Y-as worden berekend

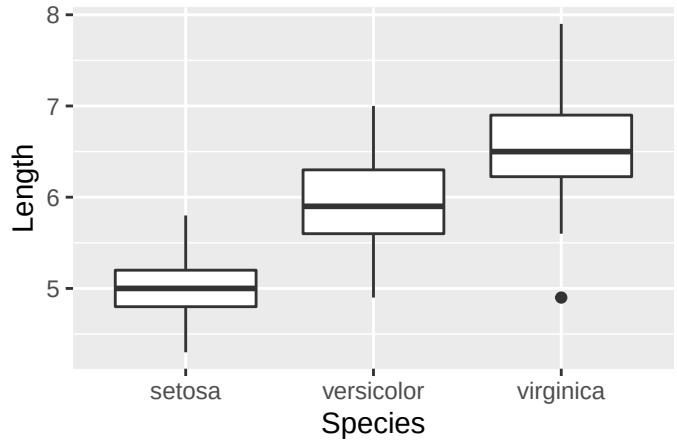
**geom_boxplot**

```
ggplot(irisSepal, aes(x = "", y = Length)) +
  geom_boxplot()
```




- Verplicht om X-as te definiëren
- Indien slechts 1 boxplot gewenst, dan is dit een lege character string
- Ofwel een categorische variabele om boxplot op te splitsen

```
ggplot(irisSepal, aes(x = Species, y = Length)) +
    geom_boxplot()
```



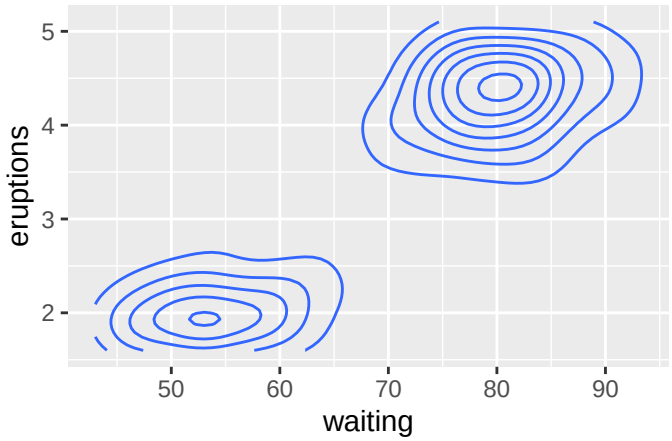

**geom_contour en geom_tile**

```
#voorbeeld uit ?geom_contour, data faithfuld is meegeleverd met R
head(faithfuld)
```

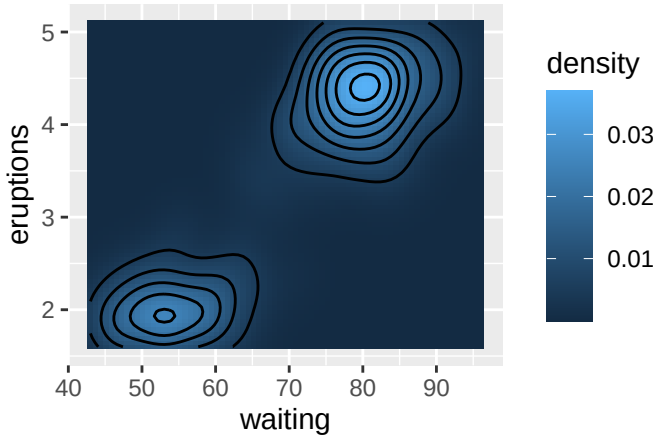
```
## # A tibble: 6 x 3
##   eruptions waiting density
##       <dbl>   <dbl>   <dbl>
## 1      1.6       43 0.00322
## 2      1.65      43 0.00384
## 3      1.69      43 0.00444
```

```
## 4      1.74      43 0.00498
## 5      1.79      43 0.00542
## 6      1.84      43 0.00574
```

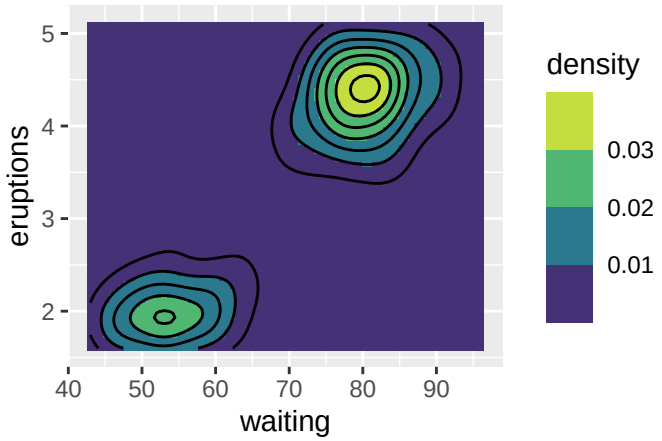
```
v <- ggplot(faithfuld, aes(waiting, eruptions, z = density))
v + geom_contour()
```




```
#kleur de plot ook in naast de contour
v + geom_tile(aes(fill = density)) + geom_contour(color = 'black')
```




```
#gebruik een ander kleurpalet
v + geom_tile(aes(fill = density)) + geom_contour(color = 'black') +
  scale_fill_viridis_b()
```

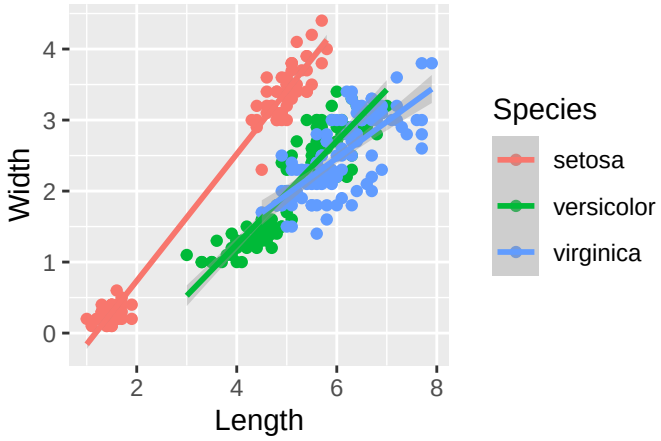



## Aanpassingen aan de defaultfiguur

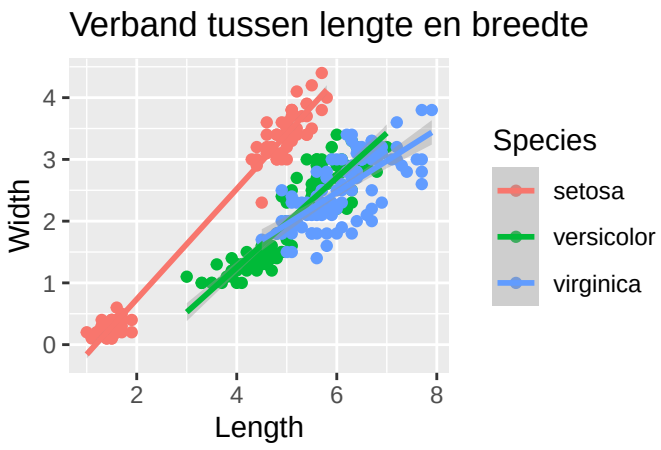
### Titel van de figuur

- Gebruik `ggtitle()` om een titel toe te voegen

```
p <- ggplot(irisAll, aes(x = Length, y = Width, color = Species)) +
  geom_point() +
  geom_smooth(method = "lm")
p
```



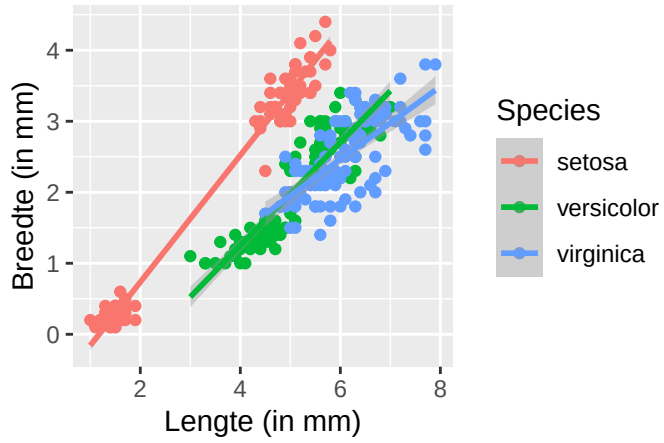

```
p + ggtitle("Verband tussen lengte en breedte")
```



### Naam van de assen

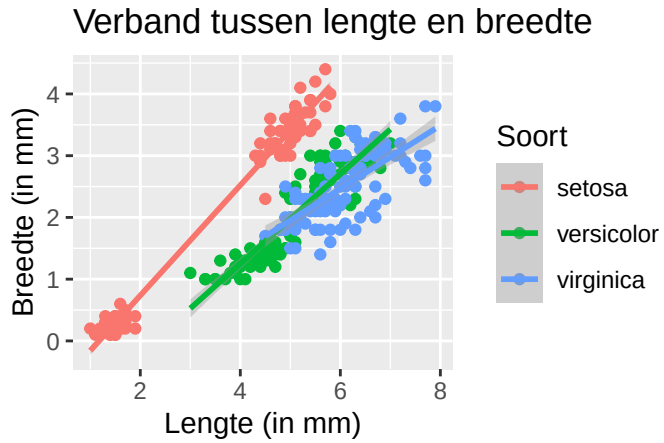
- Standaard naam van de as = de naam van de variabele
  - Dus naam variabele wijzigen = naam as wijzigen
  - Mogelijkheden beperkt door eisen kolomnamen
- Alternatief: naam van de assen instellen met `xlab()` en `ylab()`

```
p + xlab("Lengte (in mm)") + ylab("Breedte (in mm)")
```




### Gecombineerd in 1 functie

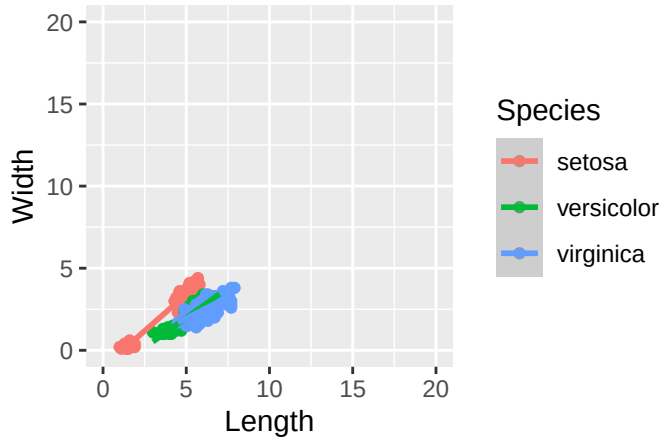
```
p + labs(title = "Verband tussen lengte en breedte",
         x = "Lengte (in mm)",
         y = "Breedte (in mm)",
         color = "Soort")
```




### Limieten van de assen

Je kan het bereik van de assen gemakkelijk aanpassen door gebruiken te maken van `+ xlim(start, einde)` voor de x-as en `ylim`voor de y-as

```r
#bv zet de limieten van de x-as en y-as van 0 tot 20
p + xlim(0,20) + ylim(0,20)
```




## Plot bewaren

- In het `plot` venster met de `Export` knop
  - Niet aan te raden, niet reproduceerbaar en je maakt fouten !!
- Ken de plot toe aan een object “‘ p <- ggplot(data, aes()) + geom_xxx() #vervang xxx door line, point, . . .
  “‘
- Bewaar dit object met een bepaalde naam
- Indien je het object niet specifieert wordt het laatst getoonde object bewaard

```r
filenaam = "output/Mijnfiguur.png"
ggsave(filenaam, p)

#alternatief
print(p)
ggsave(filenaam)
```

- Verschillende formaten door middel van het argument `device =`
  - jpeg, tiff, png, bmp, wmf, ps, pdf, . . .
  - Mogelijk om al een extensie toe te voegen aan de filenaam, dan is het overbodig om een formaat mee te geven
- Extra argumenten
  - Afmetingen: `width` en `height`
  - Eenheid van de afmetingen: `units` (“in”, “cm”, “mm”)
  - Resolutie: `dpi` (voor publicaties zet dit op minstens 300)
  - Plaats waar de plot bewaard moet worden, indien anders dan de working directory: `path =`

```r
ggsave("Figuur_iris.png", p, path = "output/",
       width = 9, height = 6, dpi = 100)
ggsave("Figuur_iris_cm.png", p, path = "output/",
       width = 9, height = 6, units = "cm", dpi = 100)
ggsave("Figuur_iris_flou.png", p, path = "output/",
       width = 15, height = 10, dpi = 10)
```

## Geavanceerde plotlayout (valt buiten de scope van de cursus)

Deze cursus ging vooral over de basisbegrippen van ggplot. Geavanceerde zaken komen hier niet aan bod, maar hierbij kort enkele typische veelgebruikte aanpassingen

### Aanpassing van de scales

Er bestaat een specifieke syntax om de schaal van de assen te controleren: scale_astype_hoofdeigenschap. Bijvoorbeeld scale_color_manual laat toe om de kleuras voorgedefinieerde kleuren te geven. In Rstudio kan je snel de mogelijkheden bij de autoaanvulling zien als je bv scale_ ingetypt hebt

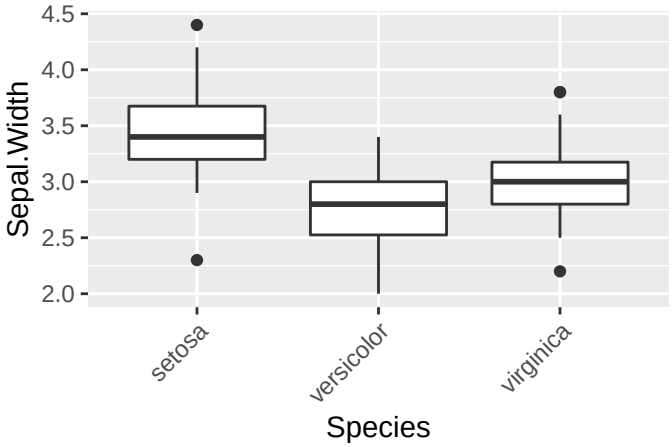
- scale_
- astype: x, y, fill, color, size, linetype, shape, . . .
- hoofdeigenschap: manual, continuous, discrete, date, datetime, time, log10, sqrt, distiller, gradient, . . .
  - bv. `+ scale_fill_manual(values = c("red", "green", "blue"))`
  - R heeft enkele standaard kleurenpalletten (rainbow, heat, terrain.colors, topo.colors, cm.colors, . . . ) `+ scale_fill_manual(values = terrain.colors(5))`
- packages als `colorbrewer`, `ggsci` bevatten andere voorgedefinieerde kleurvolgordes
  - bv. `+ scale_fill_jco()` uit het `ggsci` package
- Je kan via scales ook je plot tick labels aanpassen
  - bv. `+ scale_x_continuous(breaks = c(0,5,10,15))`
- Je kan ook via ggplot de assen in de logschaal omzetten
  - `- bv. '+ scale_x_log10()'`
- Discrete schalen kan je via `scale_x_discrete` configureren

```
ggplot(iris, aes(x = Sepal.Width, y = Petal.Width, color = Species)) + geom_point() +
scale_color_manual(values = c(setosa = "blue",
versicolor = "red",
virginica = "orange"))

ggplot(iris, aes(x = Sepal.Width, y = Petal.Width, color = Species)) + geom_point() +
scale_color_manual(values = heat.colors(3))

#als je teveel kleuren opgeeft worden enkel de eerste gebruikt
ggplot(iris, aes(x = Sepal.Width, y = Petal.Width, color = Species)) + geom_point() +
scale_color_manual(values = heat.colors(5))

#hetzelfde maar nu de x-as in log10 schaal en de y-as met onze eigen tick labels
#je ziet dat de x-as niet meer gelijk verdeeld is door de log10 transformatie
ggplot(iris, aes(x = Sepal.Width, y = Petal.Width, color = Species)) + geom_point() +
scale_color_manual(values = heat.colors(5)) +
  scale_x_log10() +
  #ylim(0,10) + dit werkt niet, want we definieren de y-as de regel hieronder
  scale_y_continuous(limits = c(0,10), breaks = seq(0, 10, by = 2))
```

### Basisthema overriden

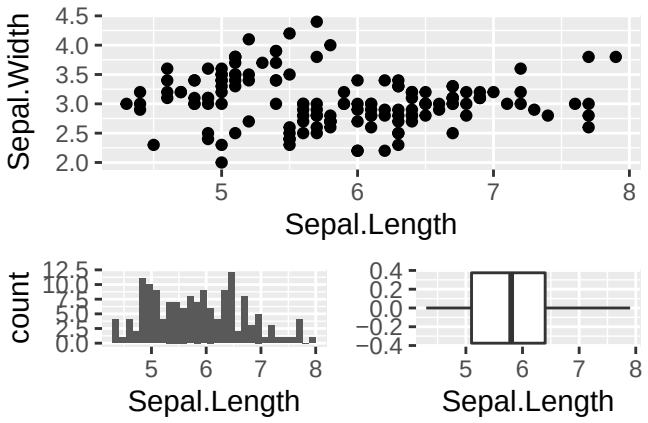
In een ggplot grafiek kan je het basisthema overriden voor specifieke cases. Je schrijft dan gewoon `+ theme(te_wijzigen_element = element_xxx(. . . ))`

```r
#Bijvoorbeeld om de x-as labels 90 graden te roteren
#hjust = 1 wil zeggen rechts uitlijnen (probeer eens andere waarden)
ggplot(iris, aes(x = Species, y = Sepal.Width)) + geom_boxplot() +
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




**Meerdere plots in 1 figuur**

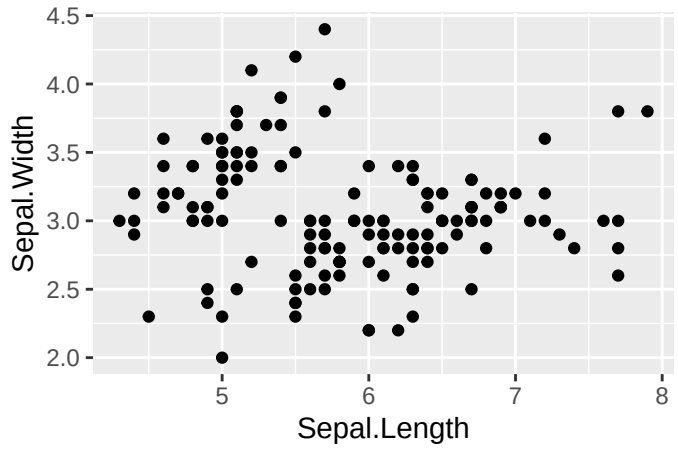
```r
#install.packages("patchwork") #eenmalig
library(patchwork)

p1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) + geom_point()
p2 <- ggplot(iris, aes(x = Sepal.Length)) + geom_histogram()
p3 <- ggplot(iris, aes(x = Sepal.Length)) + geom_boxplot()

#zet p1 bovenaan, p2 en p3 onderaan naast elkaar en maak de bovenste figuur dubbel zo hoog
p1 / (p2 + p3)  + plot_layout(heights = c(2, 1))
```



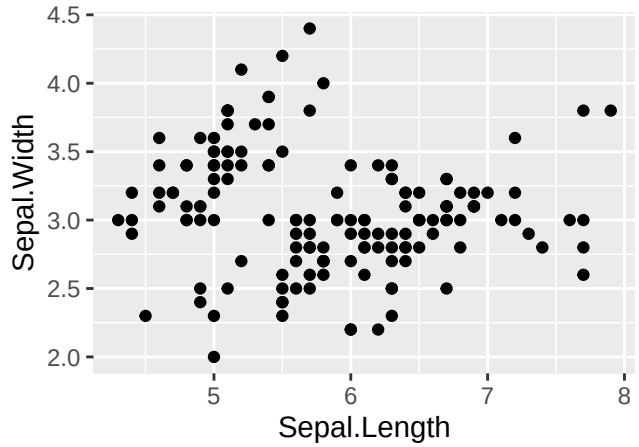

**Werken met het ggplot coördinatensysteem**

**Gelijke afstand voor x en y (bv voor kaartinformatie)** Als je wil dat 1 eenheid op de x-as exact even groot is als 1 (of een ratio van de) eenheid op de y-as kan je coord_fixed en coord_equal gebruiken

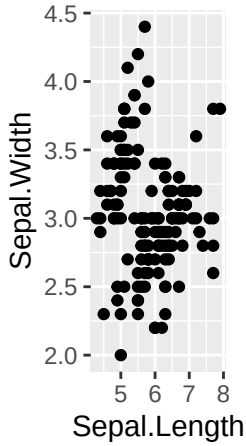
```
p1
```




```
p1 + coord_fixed()
```

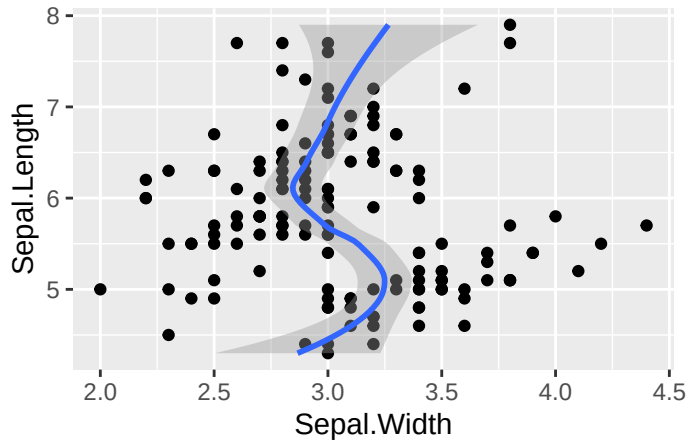



```
p1 + coord_fixed(ratio = 4)
```




**Coordinaten X en Y omwisselen** Soms kan je enkel y in functie van x berekenen, bijvoorbeeld een smoother is altijd y in functie van x. Als je dat toch zou willen omdraaien, kan je het eerst berekenen als y ten opzichte van x, en dan de X en Y coördinaten omwisselen met `coord_flip()`

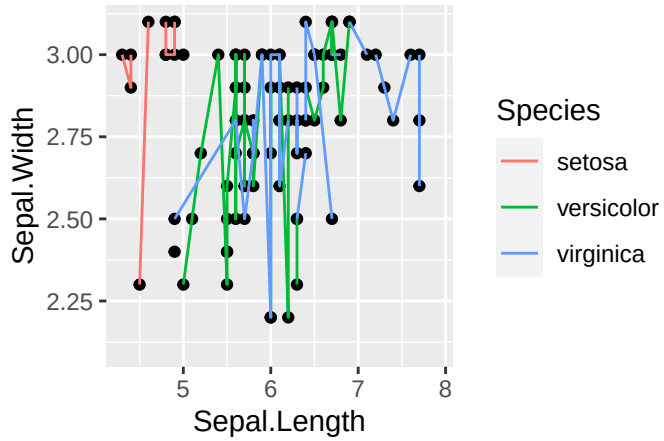
```
p1 + geom_smooth() + coord_flip()
```



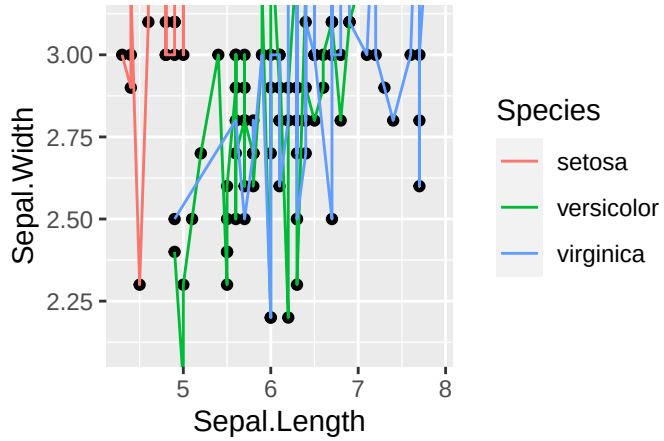

**Inzoomen op een grafiek** Je kan de limieten van de grafiek aanpassen via `xlim` en `ylim`, maar dan worden de punten buiten dit bereik ook niet meer meegenomen voor berekeningen in de grafiek, of om punten te verbinen met een lijn.

Om dit op te lossen kan je werken met `coord_carthesian`, wat de figuur wel zal maken op het totale bereik, maar dan zal inzoomen tot de limieten die je meegeeft.

```
#nu krijg je een warning dat niet alle punten in het bereik passen
#en zie je dat lijnen niet meer getekend worden naar punten buiten dit bereik
p1 + geom_line(aes(color = Species)) + ylim(2.1,3.1)
```



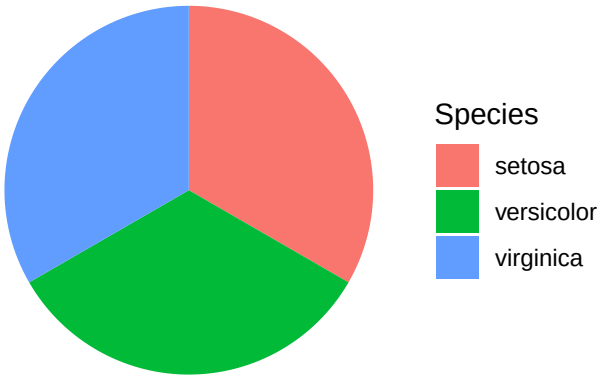

```
p1 + geom_line(aes(color = Species)) + coord_cartesian(ylim = c(2.1,3.1))
```




**Taartdiagram (pie chart)** Een piechart is een barplot, uitgezet in polaire coördinaten

- maak een gewone barplot
- zet width op 1, zodat de bars elkaar aanraken
- zet deze om naar polaire coördinaten via coord_polar, met start op 0, anders zal de cirkel kleiner zijn
- je kan de labels laten verdwijnen door theme_void te gebruiken
- ook hier kan je `facet_wrap` en `facet_grid` gebruiken

```
#Toon het aantal van elke soort in de iris dataset
ggplot(iris, aes(x = Species, fill = Species)) + geom_bar(width = 1) +
  coord_polar(start = 0) + theme_void()
```



## More to learn

- Google !!!
- Cookbook for R
- ggplot2 QuickRef
- R for data science
  - 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)
  - overzicht
- Data Carpentry
  - Visualizing Data
- Stat 545
  - All the graph things
- Cheat Sheets
  - In RStudio onder `Help` menu
  - Online

## Referenties


- R for data science
- Slides van Thierry uit 2015