# Machine Learning toepassingen

Dit hoofdstuk is een inleiding over toepassingen van Machine Learning (ML). In de toepassingen zullen we gebruikmaken van het pakket **R,** en van **RStudio**.

In het eerste deel van deze les leggen we uit wat **R** is, en hoe u **R** en **RStudio** kunt installeren op uw computer. Vervolgens zullen we laten zien wat de belangrijkste functionaliteiten zijn van base **R**, en hoe u toegang kunt krijgen tot de vele packages van **R.** De packages zijn geschreven door een grote en groeiende groep van gebruikers van **R.** Ook zullen we die datastructuren in **R** bespreken die het belangrijkste zijn voor deze module: dataframes en vectoren.

En passant maken we u wegwijs in **RStudio**. **RStudio** maakt het, kortgezegd, mogelijk om **R** te gebruiken in een omgeving die veel gebruikersvriendelijker is.

U kunt **R** het snelste leren door ermee aan de slag te gaan. En dat is wat we gaan doen in het tweede deel van deze les! Voordat u data wilt gaan analyseren en gaan gebruiken voor de ML-toepassingen in de volgende lessen, is het verstandig om uzelf een beeld te vormen van de gegevens in de dataset. Hoeveel variabelen (kolommen) bevat de dataset? En hoeveel records (rijen, of cases). Welke waarden hebben de variabelen? Als het gaat om grote hoeveelheden van gegevens is het goed om gebruik te maken van grafieken. **R** staat bekend om zijn uitstekende grafische mogelijkheden, en we zullen er een aantal van laten zien.

Deze stap in het gebruik van ML-toepassingen noemen we ook wel het exploreren of verkennen van de data. Bij het exploreren van de data maken we gebruik van statistische maatstaven. Sommige daarvan zullen u bekend voorkomen, zoals het (rekenkundig) gemiddelde. Het gemiddelde is een van de maatstaven om het midden van de gegevens te duiden (bijvoorbeeld: het gemiddelde inkomen in Nederland). Daarnaast zijn we geïnteresseerd in de spreiding van de data (wat is het inkomen van de meest verdienende Nederlander? Of hoeveel mensen leven onder de armoedegrens?). Ook deze begrippen zullen worden besproken, en natuurlijk laten we zien hoe u daarvoor gebruik kunt maken van **R.**

# R en RStudio

**R** omschrijft zichzelf op zijn website als een vrij toegankelijke softwareomgeving voor statistische berekeningen en grafische analyses (zie https://www.**R**-project.org/). De kracht van **R** schuilt in het open source karakter, en het grote aantal gebruikers dat actief aan **R** heeft bijgedragen met het schrijven van add-on packages. Zoals we zullen laten zien maken juist deze packages **R** uitermate geschikt voor toepassingen zoals geavanceerde statistiek en ML.

Om u een beeld te geven van de stormachtige ontwikkeling van **R,** het aantal beschikbare packages is exponentieel toegenomen van 100 in 2001; 500 in 2005; en 2,000 in 2010 (http://r4stats.com/articles/popularity/). In 2016 zijn er naar schatting zo'n 6,700 packages beschikbaar, en sommigen vragen zich nu af of er niet teveel zijn! Inderdaad, een nadeel van opensourcesoftware is dat er voor gelijksoortige toepassingen vaak diverse packages beschikbaar zijn. Dit is lastig voor gebruikers omdat het ondoenlijk is de voor- en nadelen van alle packages bij te houden. Dat, zou men kunnen zeggen, is de prijs die we betalen voor vooruitgang!

## Handige websites voor R

Waarschijnlijk juist door de enorme hoeveelheid aan packages en mogelijkheden zijn er uitstekende websites waar u als beginnende of gevorderde gebruiker uw voordeel mee kunt doen. Wij noemen er twee.

### Quick-R

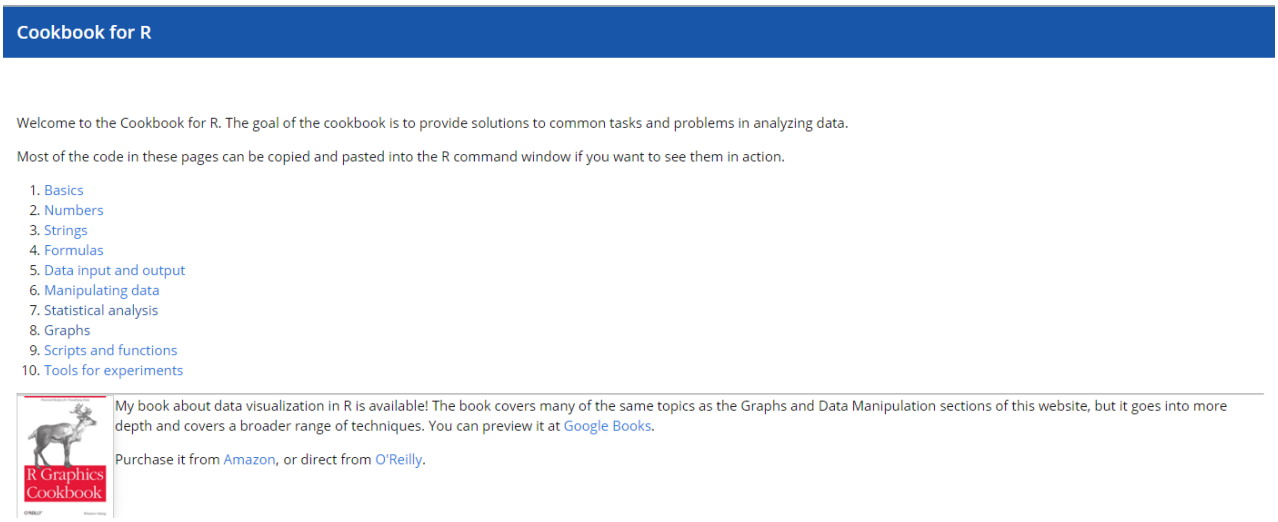
De favoriete website van velen is **Quick-R** (http://www.statmethods.net/). Op **Quick-R** (zie afbeelding 1, een screen-shot van de homepage) vindt u tal van voorbeelden, en verwijzingen naar veelgebruikte packages, als het gaat om

input en management van gegevens; elementaire en geavanceerde statistische toepassingen; en (vooral ook) data-visualisatie en grafieken.


*Afb. 1. Screenshot van Quick-R.* *Bron: http://www.statmethods.net/*

## Cookbook-R

De tweede handige website is **Cookbook-R** (http://www.cookbook-r.com/). Uit de vergelijking van de twee screenshots ziet u al snel dat de overlapping vrij groot is. In sommige gevallen is **Quick-R** handiger (vooral grafieken), in andere gevallen **Cookbook-R** (geavanceerde statistische analyses).




*Afb. 2.Screenshot van Cookbook-R.* *Bron: http://www.cookbook-r.com/*

Diegenen die ooit gewerkt hebben met commerciële statistische pakketten zoals SPSS, zullen een wezenlijk verschil merken. **R** heeft nauwelijks of geen mogelijkheden voor interactief werken met behulp van een menustructuur. Commando's moeten worden getypt in de zogenaamde **R-**console, of in een **R-**script. Omdat het natuurlijk onmogelijk is – met de vele functionaliteiten in base **R** aangevuld met die in de 6,700 add-on packages – de syntax van al die duizenden commando's uit het hoofd te leren, is het verstandig om het advies van **Cookbook R** te volgen: vind een voorbeeld op **Cookbook-R** (of **Quick-R**, of een andere site), kopieer en plak het in uw script, en pas het voorbeeld aan aan uw eigen dataset.

## Downloaden en installeren van R

Voor het downloaden van **R** gaat u naar https://www.**R-**project.org/ en volgt u de instructies. U wordt gevraagd om een CRAN mirror (Comprehensive **R** Archive Network) te selecteren. Kies voor een CRAN Mirror in de buurt (voor Nederland zijn er CRAN Mirrors in Amsterdam en Utrecht). Vervolgens kunt u kiezen voor een Windows, Linux of Mac versie. Met regelmaat komen er nieuwe versies van **R** beschikbaar. Het is zeer aan te bevelen nu en dan de oude versie te de-installeren en te vervangen door de meest recente versie, omdat nieuwe add-on packages die ontwikkeld zijn in de laatste versie van **R,** wellicht niet werken op oudere versies van **R.**

Als u een fanatiek gebruiker wordt van **R** zult u in de loop van de tijd een aantal packages veelvuldig gebruiken. Na het installeren van een nieuwe versie van **R** moet u ook deze packages opnieuw installeren. Het is daarom handig om een **R-**script aan te leggen dat dit voor u doet.

Na het installeren van **R** kunt u in principe direct aan de slag. Maar wij gaan **R** gebruiken in de gebruikersvriendelijkere **RStudio** omgeving. Om **RStudio** te installeren gaat u naar https://www.RStudio.com/, en klikt u op Download **RStudio**.

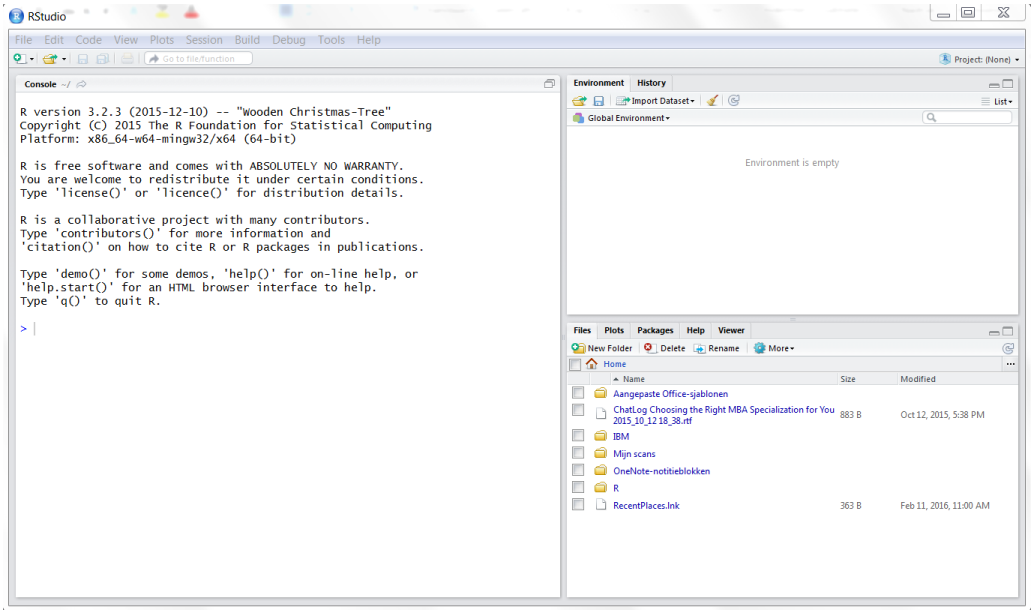
Na voltooiing van het installeren, kunt u de programma's vinden in het programmamenu, of als snelkoppeling op uw desktop. De iconen van **R** en **RStudio** zien er als volgt uit.

*Afb. 3. Icoontjes van R en RStudio.* *Bron: R en RStudio.*

## De weg weten te vinden in RStudio

Bij het openen van **RStudio** wordt automatisch de geïnstalleerde versie van **R** geladen. Oftewel, als u gebruikmaakt van **RStudio** is het niet nodig om ook nog eens snelkoppelingen naar **R** op uw desktop te hebben. Natuurlijk kunt u ook, voor snelle toegang, het **RStudio-**icoontje vastmaken aan de taakbalk.

Als dat allemaal gelukt is, en u klikt op **RStudio** dan verschijnt het volgende scherm.




*Afb. 4. De lay-out in RStudio.* *Bron: RStudio.*

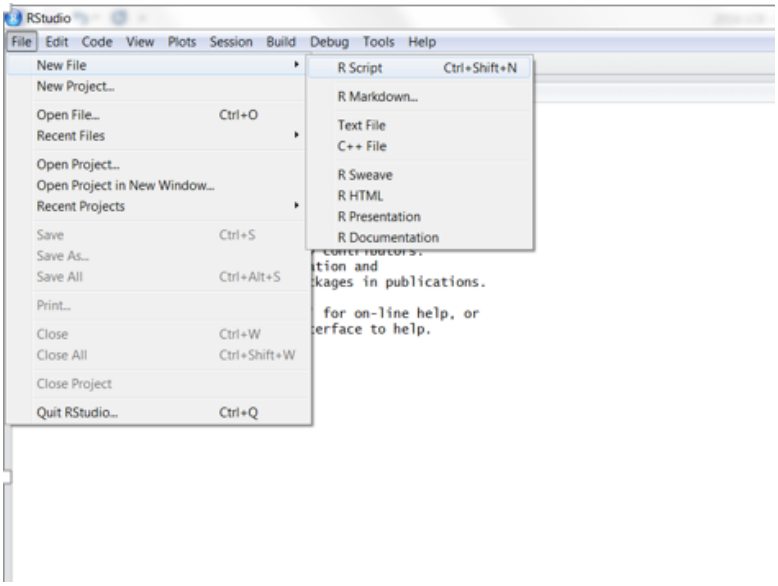
U ziet drie schermen.

## De Console

Het scherm links is de Console. In het Console kunt u commando's typen. Probeer maar de volgende commando's te typen, en zie wat er gebeurt als u op **<enter>** drukt.

```
> 2+2
[1] 4
> print("Hallo")
[1] "Hallo"
```

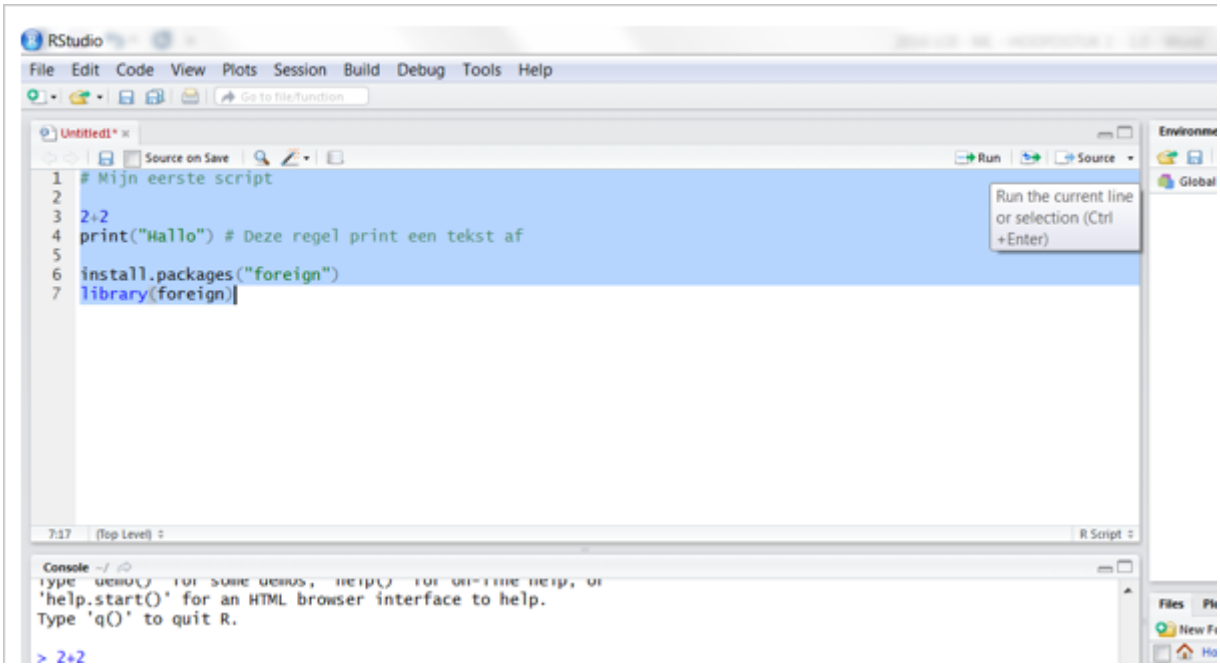
Deze manier van werken is echter niet aan te raden. Het is soms handig om allerlei snelle informatievragen in het Console te typen, maar voor werk dat u wilt bewaren is het veel beter een programma (R Script) te schrijven dat u op elk gewenst moment opnieuw kunt laten draaien. Een programma, eenmaal ontwikkeld en werkend, bevat de kennis die u in de loop van het project hebt opgedaan, en die kennis wilt u niet verloren laten gaan! Wij raden aan om aan het begin van uw project meteen een **R**-script te openen (**File>New File>R Script**).




*Afb. 5. Een (nieuw) **R**-script openen.* *Bron: RStudio.*

## Het R-script

Het **R**-script verschijnt nu als vierde scherm, linksboven. In het script kunt u commando's typen. Deze commando's kunt u vervolgens markeren (highlight), en runnen. De output verschijnt in de console, linksonder.




*Afb.6. Commando's runnen vanuit een R-script.* *Bron: RStudio.*

In het eerste script staan de volgende commando's en tekst:

```
# Mijn eerste script
2+2
print("Hallo") # Deze regel print een tekst af
install.packages("foreign")
library(foreign)
```

Enige uitleg bij deze regels:

- **R** negeert alles wat volgt achter het "#". Commentaar dat u script leesbaarder maakt, kunt u dus prima kwijt achter het "#"-teken. U kunt het "#"-teken op elke plaats in de regel zetten.
- We hebben ook een commando opgenomen voor het laden van het package genaamd foreign. Dit package stelt ons in staat om databestanden in te lezen die gemaakt zijn in andere softwarepakketten zoals **Excel** of **SPSS**. Een veelgebruikt format voor het inlezen van bestanden is het **csv** format (comma separated values); ook tab delimited tekst formats komen vaak voor. Voor die standaardformaten is het foreign package niet nodig.
- U ziet dat na de regel voor het installeren van het package er nog een regel volgt met het `library()` commando. Dit commando is nodig om binnen uw sessie ook gebruik te kunnen maken van de functies van het foreign package. Onthoudt dat het installeren van packages eenmalig is, althans tot het moment dat u een nieuwe versie van **R** installeert. Het aanroepen van de functies van het package moet in elke sessie opnieuw gebeuren!

Dit scherm, linksboven in **RStudio**, wordt niet uitsluitend gebruikt voor scripts. U kunt bijvoorbeeld ook op geopende dataframes klikken die dan in rijen en kolommen worden weergegeven.

## Het environment and history scherm

Aan de rechterkant staan nog twee schermen. Het scherm linksboven heeft twee tabs: environment en history. Onder environment vindt u een overzicht van alle objecten (zoals vectoren en dataframes) die u hebt aangemaakt binnen de sessie, en mogelijk ook van zelfgeschreven functies. Een eenvoudig voorbeeld van een vector (met een lengte van 1) is het toewijzen van een getal. U kunt bijvoorbeeld 8 toewijzen aan een object **x**, als volgt.

```
x<-8
```

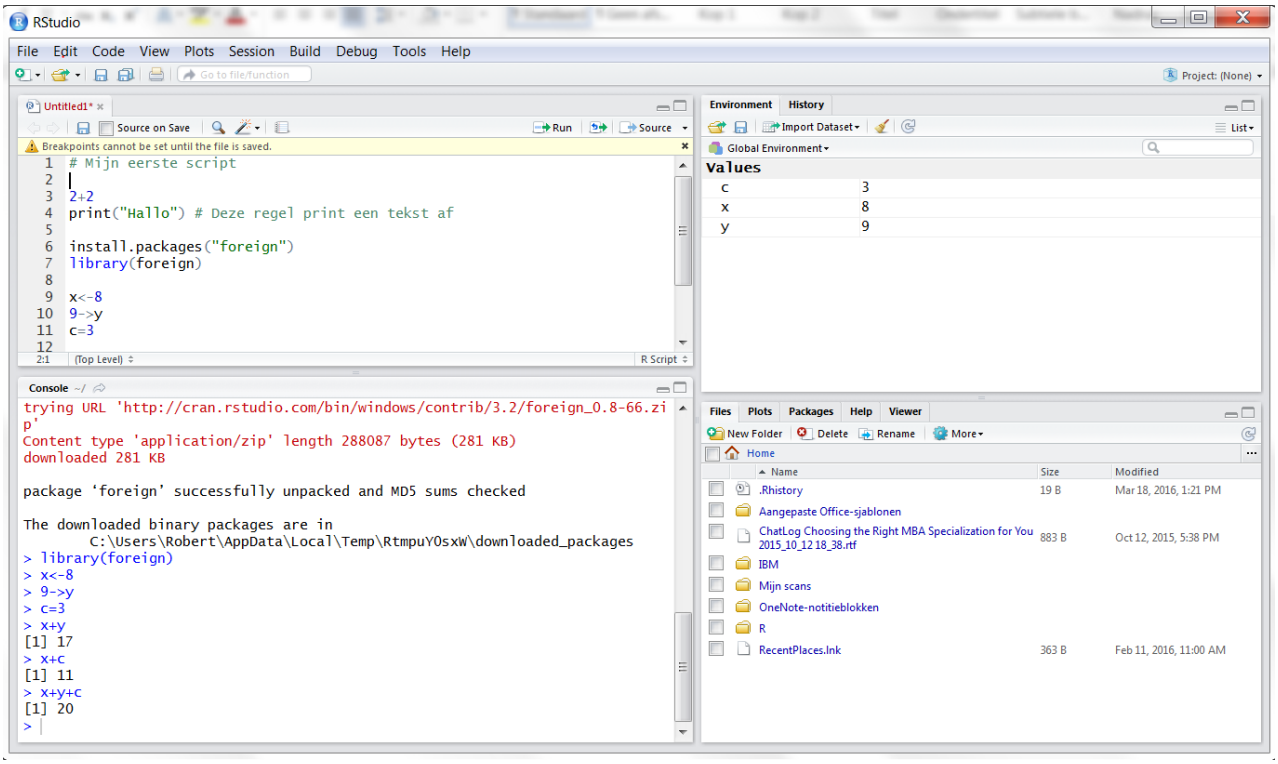
Let wel, deze pijl bestaat uit twee karakters: “<” en “-”. U kunt de pijl ook de andere kant op tekenen (zoals voor **y**); en voor toewijzingen kunt u ook gebruikmaken van het is gelijk (“=”) teken. Echter, een conventie binnen **R** is het gebruik van de pijl naar links!

```
9->y
c=3
```

Na het typen van deze regels in het script, gevolgd door markeren en runnen, ziet u dat de objecten zijn weergegeven in het environment scherm rechtsboven! U kunt nu met deze objecten gaan rekenen. Bijvoorbeeld:

```
x+y
x+c
x+y+c
```

**RStudio** laat nu de volgende informatie zien.




*Afb. 7. Informatie in alle schermen van RStudio.* *Bron: RStudio.*

## Het overzichtsscherm

Het overzichtsscherm rechtsonder biedt diverse functionaliteiten.

- Onder de tab files ziet u een overzicht van de bestanden in een map. U kunt naar een andere map gaan door te klikken op de drie puntjes (“…”) rechtsboven binnen dit scherm, waarna u naar de gewenste map kunt navigeren. In afbeelding 8 hebben we een map “C:\LOI\LOI DATA” geopend waarin een voorbeeldbestandje staat (een Excel spreadsheet, met extensie “.xlsx”). U kunt op dit bestand klikken waarna het bestand wordt geopend in Excel.

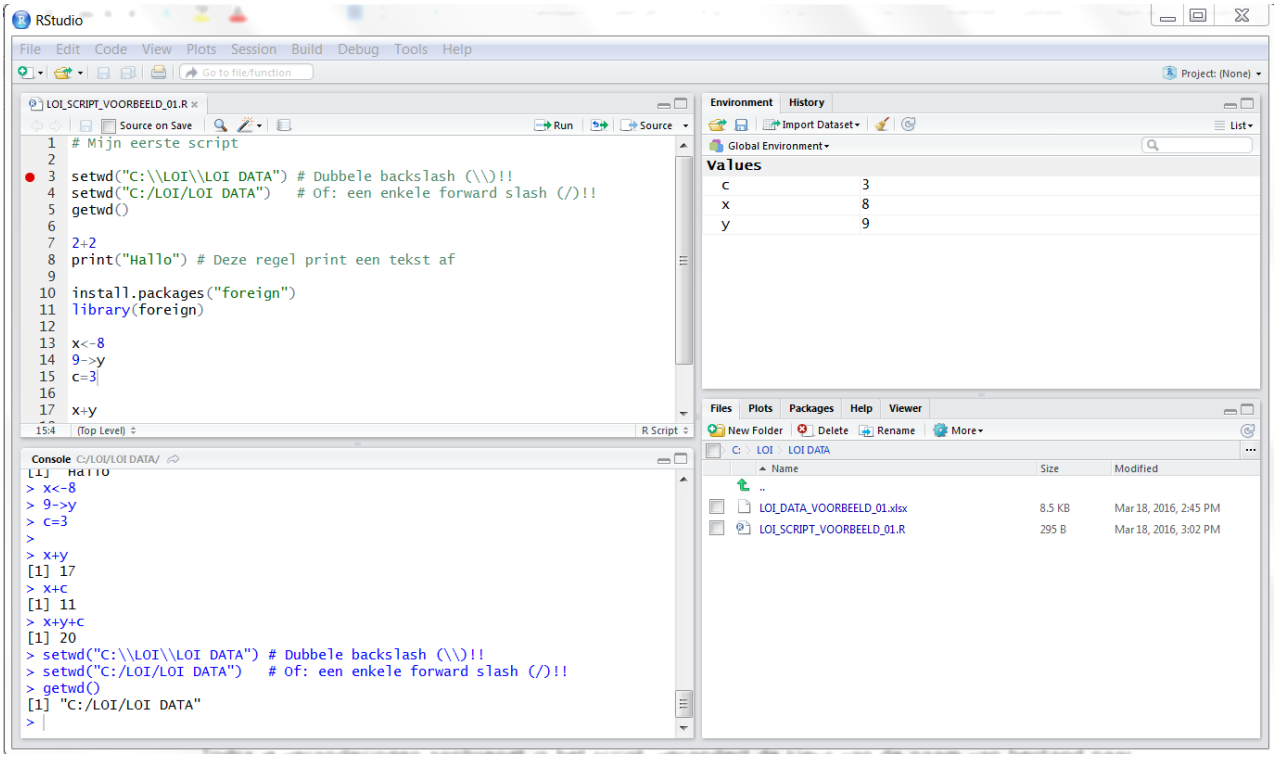
  Het is sterk aan te raden om alle bestanden (scripts; data; rapportages) die te maken hebben met een project, in een map samen te brengen. We kunnen bijvoorbeeld ons script opslaan in deze folder.

  Omdat het onthouden en typen van het volledige pad lastig en foutgevoelig is, stellen we aan het begin van ons script de werkmap (working directory) in met het commando:

  ```
  setwd("C:\\LOI\\LOI DATA") # Dubbele backslash (\\)!!
  # setwd("C:/LOI/LOI DATA") # Of: een enkele forward slash (/)!!
  getwd()
  ```

In de naam van het pad hebt u de keuze tussen een dubbele backslash (\\) en een enkele forward slash (/). Probeer beide, en zie dat het resultaat hetzelfde is! U kunt op elke moment de werkmap opvragen het **`getwd()`**. De output (de naam van de werkmap) verschijnt in de console.

Zodra u veranderingen aanbrengt in het script, verandert de kleur van de naam van bestand naar rood. Dit geeft aan dat er niet-bewaarde veranderingen in het script staan. Als u het script hebt opgeslagen dan geeft **RStudio** de volgende informatie. Rechtsonder ziet u nu ook het script in de werkmap verschijnen!




*Afb. 8. De schermen in RStudio.* *Bron: RStudio.*

- In de tab “plots” ziet u de grafieken die u hebt gemaakt in **R.** We zullen deze veelvuldig gaan gebruiken!
- In de “packages” tab vindt u een overzicht van alle geïnstalleerde packages. Er is een onderscheid in user packages, en system packages. De system packages hoeven niet apart te worden geïnstalleerd met het **`install.packages()`** commando. U kunt uitgebreide informatie over packages opvragen door erop te klikken.

  Voorbeeld: klik eens op **`foreign`**. U ziet nu een overzicht van functies binnen het package foreign. Sommige namen van functies spreken voor zich. De functie **`read.spss()`** bijvoorbeeld maakt het mogelijk om met SPSS aangemaakte bestanden te lezen. Als u doorklikt vindt u informatie over de syntax van deze functie.
- De informatie over foreign verschijnt in de volgende tab, de “help”- tab. Verder is er nog de viewer, die we niet nader zullen bespreken.

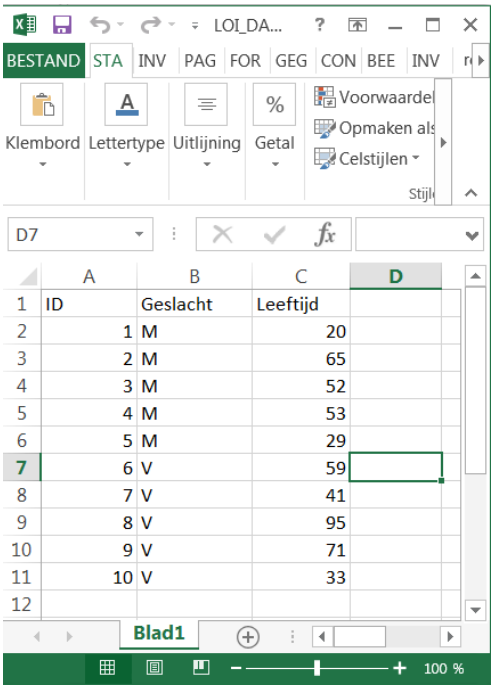
# Data Structuren en Data Management

Nu u **R** en **RStudio** hebt geïnstalleerd, en u de belangrijkste elementen van **RStudio** kent, kunnen we binnen de **RStudio** gaan werken met data. De beste manier om **R** (en andere software) te leren gebruiken is om achter de computer te gaan zitten, de voorbeelden te volgen, en ermee te experimenteren door de voorbeelden iets te veranderen.

## Invoeren van gegevens

Laten we uitgaan van het eerste voorbeeldbestand. De Excel-file kunt u openen vanuit **RStudio**. We zullen straks uitleggen hoe u de data vanuit het Excel bestand kunt inlezen in **R.** Maar eerst gaan we uit van de situatie dat de gegevens nog niet in zo’n bestand staan.

De data zien er als volgt uit. We hebben van 10 personen (5 mannen en 5 vrouwen), de gegevens over leeftijd. De observaties (of cases) staan in de rijen. Omdat we ook een rij hebben met labels voor de kolommen hebben we niet 10 maar 11 rijen. De informatie (ID; geslacht; en leeftijd) is weergegeven in kolommen.




*Afb. 9. Gegevens, in een Excel-bestand.* *Bron: Excel.*

## Vectoren

Elke kolom in het databestand is te beschouwen als een vector met een lengte van 10. We kunnen in **R** een vector maken als volgt.

```
# vectoren
id <- c(1:10) # 1:10 is korter dan 1, 2, 3, ... 10
geslacht <- c("M","M","M","M","M","M","V","V","V","V","V")
leeftijd <- c(20, 65, 52, 53, 29, 59, 41, 95, 71, 33)
```

Er zijn verschillende manieren om de data in te voeren. De **`c()`** –functie is een veelgebruikte manier om dingen te combineren. Voor **`id`** worden de getallen 1 tot 10 in een vector met lengte 10 geplaatst; “1:10” is een korte manier om “1, 2, 3, …, 10” te schrijven. Voor de vector geslacht hebben we een tekst-codering (“M” of “V”) gebruikt, en daarom moeten aanhalingstekens worden gebruikt.

In het script hebben we onder “**`# tips`**” een aantal alternatieven gegeven om hetzelfde te bereiken. Probeer ze uit. En probeer zelf een vector met lengte 10 te maken met gegevens over de lengte van de persoon in cm!

U ziet de drie vectoren in het scherm rechtsboven, onder environment. Als u de objecten x, y en c wilt verwijderen dan kunt u het **`rm()`** commando gebruiken (**`rm`** staat voor remove).

```
rm(c, x, y)
```

Voor diegenen die gewend zijn te werken met (statistische) commerciële pakketten zoals **SPSS** is het begrip vector vaak lastig. Gebruikers van **SPSS** zijn gewend te denken ze in termen van “rechthoekige” databestanden, met cases en variabelen. Die situatie zullen we repliceren in **R,** maar het is vaak niet strikt nodig.

Hoewel we drie ogenschijnlijk losse vectoren hebben gedefinieerd, kunnen we informatie aan elkaar koppelen. Bijvoorbeeld, we kunnen de leeftijden printen van mannen in de dataset, of van de laatste drie personen.

```
> leeftijd[geslacht=="M"]
[1] 20 65 52 53 29
> leeftijd[id[8:10]]
[1] 95 71 33
```

Merk op dat we in het eerste commando gebruikmaken van het dubbel is-gelijk teken ("=="). Dit is in veel programmeertalen gebruikelijk in het geval van het testen van voorwaarden. Het commando, in woorden, zegt "print de leeftijden onder de voorwaarde dat geslacht man is". In het tweede commando staat "8:10" voor 8, 9 en 10. Deze korte notatie wordt veelvuldig gebruikt in **R,** en u zult het nog vaak tegenkomen in de hoofdstukken die volgen.

## Indexeren

De toevoeging "**id[8:10]**" wordt wel aangeduid als indexering. We kijken, in dit geval, naar de elementen uit de **id**-vector met de index 8, 9 of 10. Het is soms efficiënter aan te geven welke elementen u niet wilt gebruiken. Dit kunt u doen met het minusteken ("-"). Om bijvoorbeeld het element 2 weg te laten, of de elementen 2 t/m 4 uit de **id** vector kunt u de volgende commando's gebruiken.

```
> id[-2]
[1]  1  3  4  5  6  7  8  9 10
> id[-(2:4)] # Merk op: -2:4 zal niet werken!!
[1]  1  5  6  7  8  9 10
```

De vector **geslacht** bevat tekst die het geslacht van de persoon aangeeft. Het is goed gebruik om dergelijke "categoriale" variabelen te definiëren als een zogenaamde factor. We kunnen van de oorspronkelijke vector een factor met het commando.

```
> (geslachtFac <- as.factor(geslacht))
 [1] M M M M M V V V V V
Levels: M V
> # geslacht <- as.factor(geslacht) # overschrijven is riskant!
```

Als alternatief zouden we de oude vector kunnen overschrijven. Maar dit brengt een zeker risico met zich: de oude data kunnen verloren gaan, wat kostbaar is als u een fout hebt gemaakt!

Op zich lijkt er weinig verschil te zijn tussen een tekstvariabele zoals **geslacht**, en de daarvan afgeleide facto **geslachtFac**. We zullen verderop in deze module laten zien wat de voordelen zijn van het gebruiken van factors.

## Dataframes

De meeste **R**-gebruikers vinden het toch handig om de vectoren samen te brengen in een data frame ook al is het in veel gevallen niet strikt noodzakelijk – zoals we hierboven hebben laten zien. Het combineren van vectoren binnen een data frame kan alleen als de vectoren van gelijke lengte zijn. Het combineren van vectoren tot een data frame, gebeurt met commando **data.frame()**. Door het gehele commando tussen haakjes te plaatsen wordt het dataframe gemaakt én afgedrukt in de console.

```
> (voorbeeld_01 <- data.frame(id,geslacht,leeftijd))
   id geslacht leeftijd
1   1        M       20
2   2        M       65
3   3        M       52
4   4        M       53
5   5        M       29
6   6        V       59
7   7        V       41
8   8        V       95
9   9        V       71
10 10        V       33
```

Stel dat we de leeftijden van de eerste twee personen niet kennen. Als we een vector **leeftijdMis** maken met een lengte van 8, dan zal **R** een foutmelding geven. Het is immers niet duidelijk hoe de acht leeftijden die we wel kennen, te koppelen zijn aan de 10 personen! De te koppelen vectoren moeten dus van gelijke lengte zijn. Dit betekent dat ontbrekende informatie expliciet moet worden aangegeven in de vector. Deze ontbrekende waarden worden in **R** aangeduid als **NA** (not available). Dit hebben we gedaan in **leeftijdMis2**. Met de functie **head(dataframe, n)** kunnen we de eerste rijen van een data frame afdrukken. De functie **tail()** doet hetzelfde voor de laatste rijen van een data frame. In dit geval hebben we de eerste 5 rijen afgedrukt. Als we niks invullen voor n dan worden de eerste (of voor **tail()** de laatste) 6 rijen afgedrukt.

```
> leeftijdMis <- c(52, 53, 29, 59, 41, 95, 71, 33)
> voorbeeldMis_01 <- data.frame(id,geslacht,leeftijdMis)
Error in data.frame(id, geslacht, leeftijdMis) :
  arguments imply differing number of rows: 10, 8

> leeftijdMis2 <- c(NA, NA, 52, 53, 29, 59, 41, 95, 71, 33)
> voorbeeldMis2_01 <- data.frame(id,geslacht,leeftijdMis2)
> head(voorbeeldMis2_01,5)
  id geslacht leeftijdMis2
1  1        M           NA
2  2        M           NA
3  3        M           52
4  4        M           53
5  5        M           29
```

## Selecteren van observaties, en van variabelen

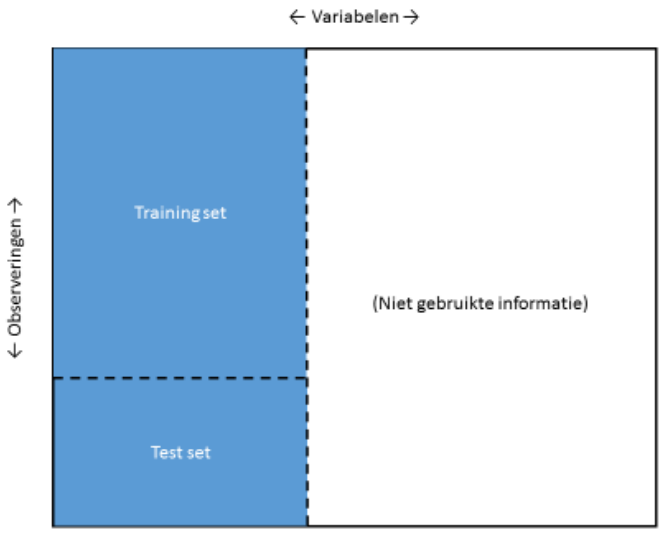
Nu we alle data overzichtelijk bijeen hebben gebracht in een data frame, kunnen we de data gaan analyseren. Een data frame is een logische manier om gegevens te analyseren omdat we, in toegepaste statistiek en in ML, meestal geïnteresseerd zijn in relaties tussen variabelen.

In het volgende hoofdstuk zullen we ingaan op twee soorten modellen, namelijk voorspellende modellen en beschrijvende modellen (en de daaraan gerelateerde begrippen supervised en unsupervised learning). Bij voorspellende modellen gaat het erom de waarde van een doelvariabele te voorspellen uit een set van andere variabelen. Bijvoorbeeld, een bank die leningen verstrekt aan kleine bedrijven, wil weten welke variabelen goede voorspellers zijn van het al of niet terugbetalen van de lening.

Een data frame is eigenlijk een grote bak van data. Uit die bak van gegevens willen we informatie halen, om van te leren. Natuurlijk kan het zijn dat we niet over alle relevante informatie beschikken (bijvoorbeeld, de bank die de leningen verstrekt heeft geen informatie over de integriteit van de aanvrager van de lening). Het zal ook zo zijn dat niet alle gegevens nodig zijn voor het model.

Voor de ML-toepassing zullen we dus vaak een selectie maken uit de ter beschikking staande gegevens. Voor voorspellende modellen heeft het zin om de data te splitsen in een trainingset en een testset. In de trainingset proberen we tot een model te komen dat de relatie tussen de voorspellers en de doelvariabele zo goed mogelijk beschrijft. Maar hoe goed het model is moet blijken uit de toepassing op gegevens die niet in de training zijn gebruikt. We maken dus twee soorten selecties:

- "Verticaal", selecteren we die variabelen die voor de ML-toepassing van belang zijn.
- "Horizontaal", splitsen we observaties in een trainingset en een testset.




*Afb. 10. Het (horizontaal en verticaal) splitsen van de dataset.* Bron: OGN.

In ons voorbeeldbestand zouden we er voor kunnen kiezen om alleen de leeftijden te analyseren, en geslacht achterwege te laten. Vervolgens zouden we de gemiddelde leeftijd kunnen vaststellen in een trainingset die 80% van de observaties (in ons voorbeeld, 8 van de 10 observaties) bevat, en kunnen kijken of de gemiddelde leeftijd in de trainingset een goede voorspeller is van die in de test set (met 2 observaties).

We zullen veelvuldig van training- en testsets gebruikmaken, en in de andere lessen zullen we ook ingaan op de omvang van de training- en de testset. Hier wijzen we er alvast op dat het uiterst belangrijk is hoe de observaties in de training- en de testset worden geselecteerd. Het basisprincipe is dat de selectie van de observaties willekeurig (at random) moet zijn. Als de observaties in het data frame al in willekeurige volgorde staan dan kunnen we simpelweg de eerste 8 observaties als trainingset nemen, en de laatste 2 als testset. Als de observaties op één of andere manier gesorteerd zijn, dan is het beter om een random steekproef te trekken uit het data frame. In het ergste geval zijn de observaties gesorteerd op leeftijd, en in dat geval zijn de leeftijden van de eerste acht observaties in de trainingset een zeer slechte voorspeller van de leeftijden in de testset!

We zien dat in ons voorbeeld de observaties gesorteerd zijn op geslacht: eerst komen de 5 mannen en dan de 5 vrouwen. Als geslacht in het data frame niet gerelateerd is aan leeftijd, dan zouden we nog steeds kunnen aannemen dat de vector leeftijd willekeurig is gesorteerd. Maar stel dat het gaat het om een data set met klanten, en onze mannelijke klanten zijn gemiddeld veel ouder of jonger zijn dan de vrouwelijke klanten. De twee sets zijn dan niet langer willekeurige steekproeven uit het bestand! Als er twijfel is, dan is het aan te bevelen om de observaties op willekeurige wijze te rangschikken, bijvoorbeeld met behulp van willekeurige getallen (random numbers). De veiligste weg is om altijd een willekeurige steekproef te trekken: als het bestand wel gesorteerd is dan moet een willekeurige steekproef. En is het bestand niet gesorteerd, dan doet het geen kwaad!

Bij indexeren maken we gebruik van de twee dimensies van het dataframe. Tussen vierkante haken selecteren we voor de komma de observaties (cases), en achter de komma de variabelen, als volgt:

```
df[selectie_cases,selectie_variabelen]
```

Willen we alle cases behouden, dan laten we **selectie_cases** voor de komma leeg; de komma echter blijft staan! Dit is, in afbeelding 10, een verticale split.

```
df[,selectie_variabelen] # alle cases, en selectie van variabelen
```

Willen we alle variabelen behouden in een horizontale split, dan gebruiken we:

```
df[selectie_cases,]
```

En een triviale manier om alle informatie te behouden is:

```
df[,] # dit is hetzelfde als df
```

Probeer dit zelf met ons voorbeeldbestandje! We hebben enkele voorbeelden opgenomen in ons script. Probeer de commando's en de resultaten te begrijpen, en maak zelf enkele andere selecties. Oefening met het splitsen van bestanden betaalt zichzelf terug!

*Voorbeeld*

We willen de leeftijden analyseren met een trainingset van 8 en een test set van 2 personen. Geslacht nemen we niet mee in de analyse. We sorteren geslacht op willekeurig wijze met willekeurige getallen.

```
> # selecteren van observaties, en van variabelen
> # willekeurig sorteren op leeftijd

> analyseSet <- voorbeeld_01[,-2] # geslacht weglaten
> set.seed(1234)  # voor reproduceerbaarheid van ons voorbeeld
> analyseSet$random <- runif(nrow(voorbeeld_01)); analyseSet
   id leeftijd      random
1   1       20 0.113703411
2   2       65 0.622299405
3   3       52 0.609274733
4   4       53 0.623379442
5   5       29 0.860915384
6   6       59 0.640310605
7   7       41 0.009495756
8   8       95 0.232550506
9   9       71 0.666083758
```

```
10 10       33 0.514251141
> analyseSet <- analyseSet[order(analyseSet$random),]
> analyseSet$random <- NULL      # het random getal is niet meer nodig
> training <- analyseSet[1:8,]   # eerste 8 observaties in training set
> test     <- analyseSet[9:10,]  # laatste 2 in test set
> training; test
   id leeftijd
7   7       41
1   1       20
8   8       95
10 10       33
3   3       52
2   2       65
4   4       53
6   6       59
  id leeftijd
9  9       71
5  5       29
> mean(training$leeftijd)
[1] 52.25
> mean(test$leeftijd)
[1] 50
```

***Toelichting***

1. Het bestandje **analyseSet**, is hetzelfde als ons voorbeeldbestand, maar dan zonder de variabele geslacht.

   In **naamDataFrame[observaties,variabelen]**, staan de variabelen die u wilt (de)selecteren achter de komma. Onthoud goed, ook als u alle observaties wilt behouden is de komma nodig. In **voorbeeld_01[,-2]** zeggen we dus dat we alle observaties willen behouden (voor de komma staat niks), maar de tweede variabele (**geslacht**) deselecteren.

2. Omdat we niet zomaar kunnen aannemen dat de observaties in willekeurig volgorde staan, maken we een willekeurig getal aan met de **runif(n)** functie. De toevalsgetallen zijn getrokken uit een uniforme verdeling van getallen (met heel veel decimalen) tussen 0 en 1. Uniform, in dit verband, wil zeggen dat de kans op een getal tussen 0 en 0.10, net zo groot is als de kans op een getal tussen .10 en 0.20, of tussen 0.60 en 0.70.

3. Tussen haakjes geeft de **n** aan hoeveel willekeurige getallen we willen genereren. In ons geval zijn dat er 10. We zouden dus **runif(10)** kunnen opgeven, maar het is handiger om **R** te vertellen dat **n** gelijk moet zijn aan het aantal rijen in ons data frame; we kunnen daarvoor de functie **nrow(dataframe)** gebruiken.

4. Het **set.seed()** commando zorgt ervoor dat u dezelfde random nummers krijgt als in ons voorbeeld.

5. Vervolgens sorteren we het data frame van laag naar hoog, op het willekeurig getal, in de variabele random. Merk op dat we het oude bestand **analyseSet** overschrijven met het nieuwe, gesorteerde bestand.

6. Eenmaal gesorteerd, hebben we de variabele random niet meer nodig. Het verwijderen van een object (in dit geval een variabele) kan door de toewijzing **NULL**.

7. Ten slotte maken we een trainingset en een testset uit de eerste 8 respectievelijk de laatste 2 observaties van het data frame.

Het is natuurlijk altijd beter om het rekenwerk over te laten aan de computer! In plaats van te moeten onthouden hoeveel observaties er zijn, en te bepalen bij welke observatie de trainingset af te kappen is een kort programmaatje handig.

```
trainAndTest <- function(dataframe, omvang)
{
  Z <- nrow(dataframe)
  x <- as.integer(omvang * z)
  y=x+1
```

```
  cat("De Training set loopt van 1 tot ",x,"\nTest set loopt van ", y," tot ",
z,"\n")
}
```

Het runnen van deze regels creëert de functie **trainAndTest**, die als object in de environment verschijnt. De functie gebruikt als argumenten: de naam van een dataframe (**analyseSet**), en de proportie (0.80, of 80%) van de observaties te gebruiken in de trainingset. U kunt de proportie nu zelf veranderen, in bijvoorbeeld 0.75 of 0.90. Met 0.80 (we laten de nul voor de decimale punt meestal weg: .80) geeft de functie de volgende output:

```
> trainAndTest(analyseSet, .80)
De Training set loopt van 1 tot  8
Test set loopt van  9  tot  10
```

Die getallen kunnen we vervolgens gebruiken bij het aanmaken van de training- en de testset!

```
(train2 <- analyseSet[1:8,])
(test2  <- analyseSet[9:10,])
```

# Verkennen van gegevens

## Beschrijven van de data: basisfuncties

Nu we weten hoe we vanuit een bestand selecties kunnen maken van observaties en variabelen, zijn we klaar voor het echte werk!

De eerste stap is het verkennen van de gegevens. Dat is belangrijk om een beeld te krijgen van de mogelijke waarden van de variabelen, en ook om de kwaliteit van de gegevens te beoordelen. Bijvoorbeeld, als we een bestand hebben met onder andere rapportcijfers van leerlingen, dan zijn we geïnteresseerd in de gemiddelde scores, of in de scores van jongens en meisjes. Maar ook willen we checken dat de cijfers liggen tussen 1 en 10. Een rapportcijfer van 100 kan een (type)fout zijn in de registratie. We moeten iets met onjuiste waarden doen omdat een cijfer van 100 (zeker als de groep klein is) het gemiddelde ten onrechte zal optrekken.

**R** biedt verschillende manieren om een overzicht te krijgen van de gegevens. De **summary()** functie is een veelgebruikte functie. De output van **summary()** is afhankelijk van het soort object dat tussen de haakjes staat. Als we de functie loslaten op een data frame dan krijgen we een overzicht van de waarden van de variabelen. Voor numeriek variabelen zoals leeftijd, zien we het minimum (de jongste persoon in het bestand is 20), het maximum, de gemiddelde leeftijd (mean, is 51.8) en de mediaan.

Een statistisch overzicht van de variabele **id** (hier een volgnummer van 1 tot 10) is niet zinnig. We hadden dit deel achterwege kunnen laten door **id** te deselecteren met **voorbeeld_01[,-1]**. Merk op dat de **summary()** functie slim is: voor de variabele **geslacht** worden de voorkomens van "M" en "V" geteld (het berekenen van een gemiddelde waarde is niet mogelijk).

```
> summary(voorbeeld_01)
       id        geslacht    leeftijd
 Min.   : 1.00   M:5      Min.   :20.0
 1st Qu.: 3.25   V:5      1st Qu.:35.0
 Median : 5.50            Median :52.5
 Mean   : 5.50            Mean   :51.8
 3rd Qu.: 7.75            3rd Qu.:63.5
 Max.   :10.00            Max.   :95.0
```

### Het rekenkundig gemiddelde

Het (rekenkundig) gemiddelde zal u ongetwijfeld bekend zijn, het is de som van alle leeftijden gedeeld door het aantal personen. De mediaan is net als het gemiddelde een maatstaf voor de centrale tendentie (of het midden) van de verdeling.

Als we **R** als rekenmachine gebruiken, dan is het gemiddelde te berekenen als:

```
> gemiddelde <- (20+65+52+53+29+59+41+95+71+33)/10; gemiddelde
[1] 51.8
> mean(voorbeeld_01$leeftijd)
[1] 51.8
```

Als u alleen maar geïnteresseerd bent in het gemiddelde dan kunt u gebruikmaken van de **mean()** functie. We verwijzen naar een variabele in een dataframe, met de formulering **dataframe$variabele**; in dit geval **voorbeeld_01$leeftijd**.

## De mediaan

De mediaan is lastiger zelf te bepalen omdat u eerst de variabele moet sorteren. De mediaan is dan die waarde waar de helft van de verdeling onder ligt en de andere helft boven. In een reeks met een even hoeveelheid getallen (zoals 10, in ons voorbeeld), ligt de mediaan tussen de onderste en de bovenste 5 in. We nemen dan het gemiddelde van de $5^e$ en $6^e$ waarneming (52+53)/2 = 52.5. De **median()** functie geeft direct het resultaat.

Een voordeel van de mediaan is dat die minder gevoelig is voor uitschieters. Ook al zou de oudste persoon 115 jaar oud zijn in plaats van 95, dan zou de mediaan niet veranderen (maar het rekenkundig gemiddelde wel!). Als het rekenkundig gemiddelde hoger (lager) is dan de mediaan, dan duidt dat erop dat de verdeling niet symmetrisch is, maar dat er uitschieters zijn naar boven (beneden).

```
> voorbeeld_01<-voorbeeld_01[order(voorbeeld_01$leeftijd),]; voorbeeld_01[3]
   leeftijd
1        20
5        29
10       33
7        41
3        52
4        53
6        59
2        65
9        71
8        95
> median(voorbeeld_01$leeftijd)
[1] 52.5
```

## De structuur van de data set

Een ander handig commando is een overzicht van de structuur van de data, met **str(dataframe)**.

```
> str(voorbeeld_01)
'data.frame': 10 obs. of 3 variables:
 $ id       : int  1 5 10 7 3 4 6 2 9 8
 $ geslacht: Factor w/ 2 levels "M","V": 1 1 2 2 1 1 2 1 2 2
 $ leeftijd: num  20 29 33 41 52 53 59 65 71 95
```

De functie geeft als output het aantal observaties (10) en het aantal variabelen (3). Voor iedere variabele wordt weergegeven het type variabele, en de waarden van de eerste observaties (alleen omdat het aantal observaties gering is in ons voorbeeld, krijgen we alle waarden te zien). **R** maakt zelf een factor variabele van **geslacht**, en de **str()** functie geeft weer dat de factor twee levels (twee mogelijke waarden) heeft, namelijk “M” en “V”. Hoewel ons databestand inderdaad “M” en “V” bevat voor geslacht, zet **R** dit om in coderingen (1 en 2), waarbij de codering de alfabetische volgorde volgt (“M” wordt 1, want komt in het alfabet voor “V”). Sommigen vinden dit verwarrend. Voor hen zijn er gelukkig methoden om factoren te omzeilen. Voor onze toepassingen maakt het meestal niet veel uit.

Voor een snel overzicht van enkele observaties in het data frame, kan gebruik worden gemaakt van de **head()** en **tail()** functies. Als geen getal wordt ingegeven dan worden de variabelen van de eerste of laatste 6 observaties afgedrukt. Variabelen kunnen worden geselecteerd met de gebruikelijke syntax.

Enkele voorbeelden. Voor het overzicht sorteren we eerst het data frame op **id**.

```
> voorbeeld_01<-voorbeeld_01[order(voorbeeld_01$id),]
> head(voorbeeld_01)        # eerste 6 observaties
  id geslacht leeftijd
1  1        M       20
2  2        M       65
3  3        M       52
4  4        M       53
5  5        M       29
6  6        V       59
```

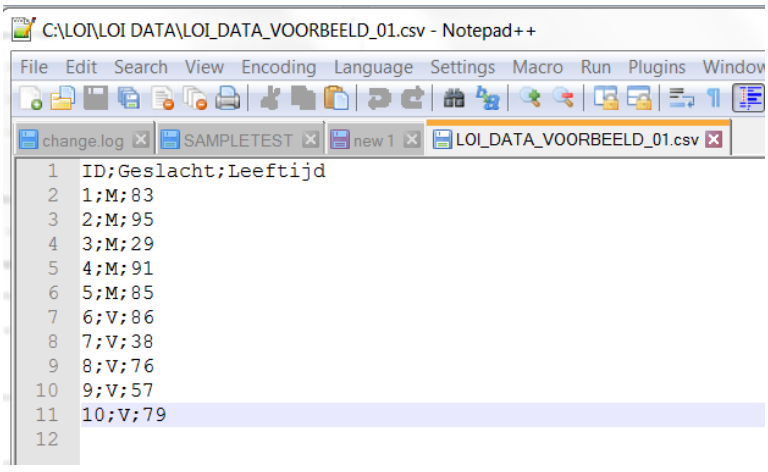
```
> tail(voorbeeld_01, 3)      # laatste 3 observaties
   id geslacht leeftijd
8   8        V       95
9   9        V       71
10 10        V       33
> head(voorbeeld_01[-1], 4) # eerste 4 observaties; variabele id niet
  geslacht leeftijd
1        M       20
2        M       65
3        M       52
4        M       53
> head(voorbeeld_01[,c("id","leeftijd")])
  id leeftijd
1  1       20
2  2       65
3  3       52
4  4       53
5  5       29
6  6       59
```

## Inlezen van gegevens

Voor de basiscommando's die we tot dusver hebben behandeld, hebben we gebruiktgemaakt van een simpel voorbeeldbestandje, met weinig observaties en weinig variabelen. Voor het illustreren van het exploreren van gegevens en vooral ook van relaties tussen variabelen met **R,** zullen we gebruik maken van een groter bestand. Maar voordat we dat doen is het goed nog even terug te komen op het inlezen van "externe" gegevens.

Databestanden zijn er in vele formats. Een veelgebruikt format is het **csv** (comma separated value) format. Een **csv**-bestand kan worden aangemaakt met bijvoorbeeld **Excel**. Echter, afhankelijk van de instellingen van **Excel** is het scheidingsteken soms niet een komma maar een puntkomma. De functie **`read.csv()`** in **R** is gelukkig flexibel genoeg om daarmee om te gaan. Check dus altijd welke scheidingsteken gebruikt is!

In ons voorbeeld hebben we het in **Excel** aangemaakte bestand bewaard als een **csv**-bestand. Met een krachtige editor zoals **Notepad++** ziet u snel hoe zo'n bestand er uitziet en welk scheidingsteken is gebruikt. In ons geval is dat dus ";" (ook al hebben we het opgeslagen als comma separated!). Als u toch een komma wilt, dan kunt u in Notepad++ alle ";" vervangen door ",". Maar handiger is het om in de **`read.csv()`** functie aan te geven dat het scheidingsteken niet "," is (de default) maar ";".




*Afb. 11. Een csv-bestand, geopend in Notepad++.* *Bron: Notepad.*

Het **R**-commando luidt als volgt. De toevoeging **`sep=";"`** geeft aan dat de puntkomma als scheidingsteken (separator) is gebruikt. De toevoeging **`header=TRUE`** geeft opdracht om de eerste regel van het bestand te gebruiken voor de labels van de variabelen.

```
> vb_csv <- read.csv("ML2_VOORBEELD.csv", sep=";", header=TRUE)
> head(vb_csv, 3)
  ID Geslacht Leeftijd
1  1        M       83
2  2        M       95
3  3        M       29
```

## Het verkennen van gegevens: een uitgebreid voorbeeld

Voor een overzicht van manieren om data verkennen maken we gebruik van het bestand **vervreemding.csv**. Het bestand bevat de volgende gegevens:

- Een volgnummer (**id**) van 1 tot 100, voor de 100 personen in de steekproef.
- **vervreemding**: Een score van 1 tot 10 die iemands vervreemding van de maatschappij weergeeft. De score is gebaseerd op de uitkomsten van een enquête met op wetenschappelijk onderzoek gebaseerde vragen die een indicatie geven van vervreemding.
- **inkomen** en **spaar**: Gegevens over het inkomen en spaartegoed (in €) van het huishouden waartoe de persoon behoort.
- De regio waar iemand woont (**regio** en **regioFac**). In **regio** is een numerieke codering gebruikt (1 voor Noordwest; en 2 voor Zuidoost). In **regioFac** staat dezelfde informatie maar dan als tekst.
- De religie van de persoon. Een onderscheid is gemaakt tussen protestant, rooms-katholiek en anders. De variabele **religie** bevat codes 1 t/m 3, en **religieFac** bevat tekst.

Na het inlezen van de gegevens is het altijd goed een overzicht te maken met de functies die we eerder hebben besproken: **str()**, **head()** en **summary()**.

```
> str(vv)
'data.frame':    100 obs. of 8 variables:
 $ id          : int  35 10 3 40 89 44 6 25 92 39 ...
 $ vervreemding: int  9 6 6 1 2 5 4 3 5 8 ...
 $ inkomen     : num  23571 49286 50000 84286 62143 ...
 $ spaar       : num  15675 35423 36965 86936 50433 ...
 $ regio       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ regioFac    : Factor w/ 2 levels "NW","ZO": 1 1 1 1 1 1 1 1 1 1 ...
 $ religie     : int  1 1 3 3 1 1 1 1 1 3 ...
 $ religieFac  : Factor w/ 3 levels "Anders","Protestant",..: 2 2 1 1 2 2 2 2
> head(vv)
  id vervreemding  inkomen    spaar regio regioFac religie religieFac
1 35            9 23571.43 15674.72     1       NW       1 Protestant
2 10            6 49285.71 35423.39     1       NW       1 Protestant
3  3            6 50000.00 36965.44     1       NW       3     Anders
...
> summary(vv)
       id          vervreemding      inkomen          spaar             regio     regioFac    religie
 Min.   :  1.00   Min.   : 1.00   Min.   :    0   Min.   :   741.1   Min.   :1.0   NW:50    Min.   :1.00
 1st Qu.: 25.75   1st Qu.: 3.00   1st Qu.:21071   1st Qu.: 18340.7   1st Qu.:1.0   ZO:50    1st Qu.:1.00
 Median : 50.50   Median : 5.00   Median :42500   Median : 30918.0   Median :1.5            Median :2.00
 Mean   : 50.50   Mean   : 5.56   Mean   :41479   Mean   : 35665.9   Mean   :1.5            Mean   :1.92
 3rd Qu.: 75.25   3rd Qu.: 8.00   3rd Qu.:62857   3rd Qu.: 50869.6   3rd Qu.:2.0            3rd Qu.:3.00
 Max.   :100.00   Max.   :10.00   Max.   :94286   Max.   :119267.8   Max.   :2.0            Max.   :3.00
      religieFac
 Anders    :29
 Protestant:37
 RK        :34
```

**Zelf-test:**

1. Hoeveel personen telt het bestand?
2. Wat is het gemiddelde inkomen van deze groep personen?
3. Wat is het mediane spaartegoed? Welke conclusie kunt u trekken uit het gegeven dat het gemiddelde spaartegoed hoger is dan het mediane spaartegoed?
4. Hoeveel personen wonen in het zuidoosten van het land?

## Spreiding van de gegevens: bereik, interquartile range

De basisfuncties geven op een compacte manier al een schat van informatie over de inhoud van het bestand. Over het inkomen kunnen we zeggen dat het "midden" van de verdeling rond de €42.000 ligt. De spreiding is echter vrij groot: er zijn personen die helemaal niets verdienen, terwijl het hoogst gemeten inkomen ruim €94.000 is. Het bereik (range) van inkomen, gedefinieerd als de hoogste minus de laagste waarde is dus ook ruim €94.000. Omdat het bereik gevoelig is voor uitschieters naar boven en naar beneden, gebruiken we vaak de Interquartile Range (IQR) die het verschil meet tussen het inkomen aan de onderkant waaronder 25% van de groep zich bevindt en het inkomen aan de bovenkant waarboven 25% zich bevindt. Uit de **`summary()`** informatie lezen we af dat 25% minder verdient dan € 21,071, en 25% meer dan € 62,857. De IQR is het verschil tussen die twee: € 41,786. Al deze informatie is gemakkelijk op te vragen in **R**:

```
> mean(vv$inkomen)
[1] 41478.57
> median(vv$inkomen)
[1] 42500
> range(vv$inkomen)
[1]     0.00 94285.71
> diff(range(vv$inkomen))
[1] 94285.71
> IQR(vv$inkomen)
[1] 41785.72
```

## De verdeling van de gegevens grafisch weergegeven: boxplot

Voor een gedetailleerde beeld van de verdeling van inkomens over personen kunnen we gebruikmaken van twee soorten grafieken: de boxplot en het histogram.

Boxplots van inkomen en spaartegoed kunnen worden verkregen met de commando's:

```
boxplot(vv$inkomen, main="Boxplot van inkomen", ylab="Inkomen in Euro")
boxplot(vv$spaar, main="Boxplot van spaartegoed", ylab="Spaartegoed in Euro")
```

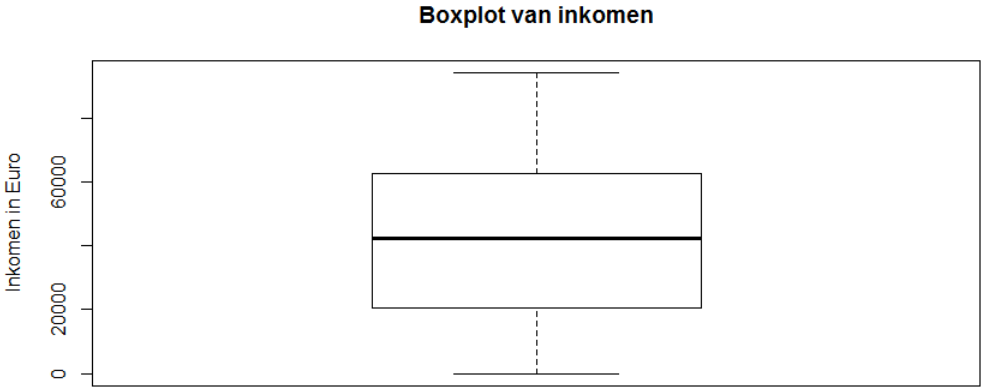
Zeker als u de grafiek wilt gebruiken in een rapport bestemd voor anderen, is het verstandig de titel van de grafiek en van de verticale as in te stellen met de **`main`** en **`ylab`** opties.

De grafieken verschijnen nu rechtsonder, onder de "plots" tab. In het scherm is een optie om de grafiek te exporteren, bijvoorbeeld naar het klembord (clipboard). In het klembord kunt u de lengte en de breedte aanpassen. Met "copy plot" kan een kopie worden gemaakt die rechtstreeks in bijvoorbeeld uw rapport kan worden geplakt.

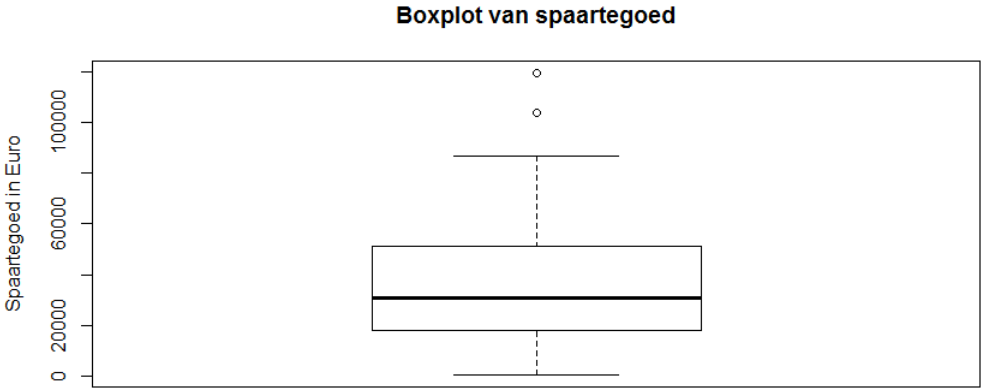
De boxplot (voluit ook wel box-and-whiskers plot genoemd), geeft op een compact manier informatie over:

- Het midden van de verdeling: de dikke streep in de rechthoek (box) geeft de mediaan aan van de verdeling (dus niet zoals velen denken het gemiddelde!).
- De 25%- en 75% kwartielen van de verdeling (de onderkant en de bovenkant van de box).
- De IQR: de lengte van de box.
- Het minimum, maximum en bereik: de streep aan de onderste whisker geeft het minimum, weer. De streep aan de bovenste whisker geeft het maximum weer. De afstand tussen de twee is het bereik. Dit geldt niet als er uitschieters zijn in de data!

- De vorm van de verdeling: omdat de whiskers aan de bovenkant iets groter zijn dan aan de onderkant is de verdeling ietwat scheef. Dat is ook wel logisch omdat inkomen en spaartegoeden aan de onderkant begrensd zijn, bij nul, maar aan de bovenkant niet begrensd zijn.
- Eventuele uitschieters. De lengte van de whiskers is hooguit 1.5 keer de IQR. Waarden daarbuiten worden getoond als cirkels. In de boxplot van inkomen komen geen uitschieters voor; in de boxplot van spaartegoed komen twee uitschieters voor aan de bovenkant van de verdeling.




*Afb. 12. Boxplot inkomen.* *Bron: OGN.*

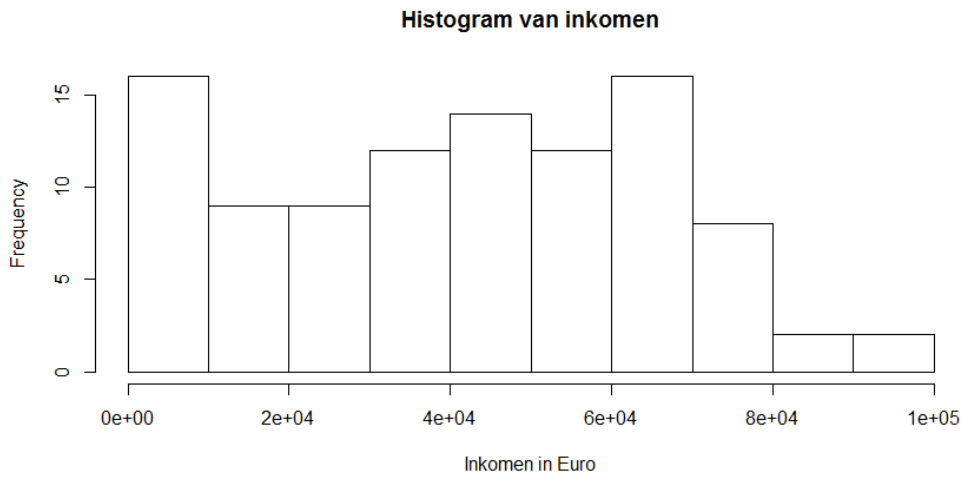



*Afb. 13. Boxplot van spaartegoed.* *Bron: OGN.*

Let op: de grafieken worden geopend in een scherm dat u ook weer moet sluiten! Als u vergeet af te sluiten dan kan het gebeuren dat u een nieuwe grafiek wilt maken maar er gebeurt ogenschijnlijk niets! Dit kunt u verhelpen door de **`plot()`** vooraf te laten gaan door het commando **`dev.off()`** dat u op diverse plaatsen in het script tegenkomt.

## De verdeling van de gegevens grafisch weergegeven: histogram

Een andere manier om de verdeling weer te geven is het histogram. De syntax van deze commando's is vrijwel gelijk aan die van boxplot; het enige verschil is dat de waarden van de variabele nu niet op de verticale as (de y-as) staan maar op de horizontale as (de x-as); in plaats van **`ylab`** gebruiken we daarom **`xlab`**, voor de titel van de as.

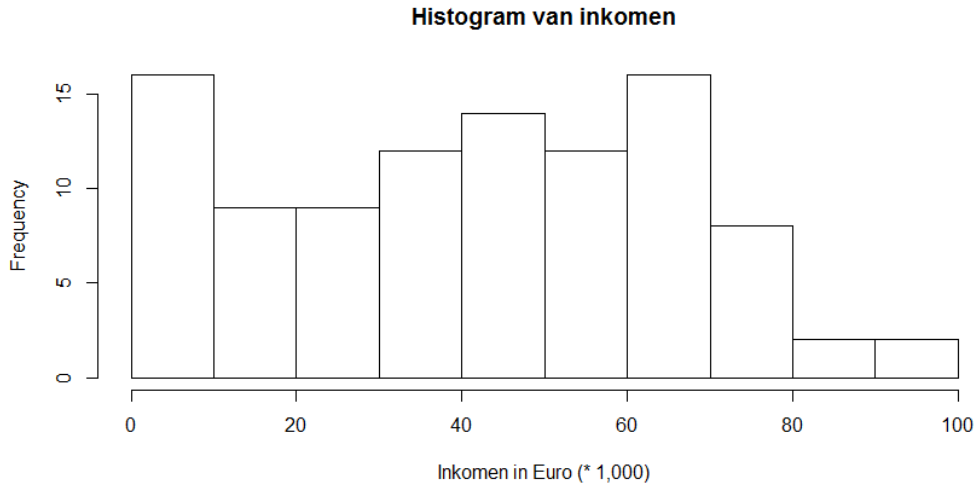



*Afb. 14. Histogram van inkomen.* *Bron: OGN.*

In de grafiek ziet u dat de waarden op de x-as in de zogenaamde wetenschappelijke notatie staan weergegeven. In die notatie betekent 2e+04, bijvoorbeeld, 2 * $10^4$, ofwel 2*10,000 = 20,000. Voor de lezer van het rapport is de wetenschappelijke notatie niet erg gebruikersvriendelijk. Een manier om de gegevens fraaier te presenteren, is het aanmaken van een nieuwe variabele die het inkomen weergeeft in duizenden euro's. In de **`round()`** functie geeft het getal 3 het aantal cijfers achter de decimale punt aan (oftewel, de nauwkeurigheid).

```
vv$ink_1000   <- round(vv$inkomen/1000, 3)
hist(vv$ink_1000, main="Histogram van inkomen", xlab="Inkomen in Euro (*
1,000)")
```

```
> head(vv[,c("id","inkomen","ink_1000","spaar", "spaar_1000")])
  id  inkomen ink_1000    spaar spaar_1000
1 35 23571.43   23.571 15674.72     15.675
2 10 49285.71   49.286 35423.39     35.423
3  3 50000.00   50.000 36965.44     36.965
4 40 84285.71   84.286 86936.47     86.936
5 89 62142.86   62.143 50432.92     50.433
6 44 56428.57   56.429 50121.11     50.121
```




*Afb. 15. Histogram van inkomen (met aangepaste schaal).* *Bron: OGN.*

De verdeling van inkomen geeft pieken aan tussen de dertig- en de zeventigduizend euro. Een kleiner aantal personen verdient inkomens hoger dan tachtig- of lager dan dertigduizend euro, maar tussen 0 en tienduizend is er weer een piek (waarschijnlijk door de groep van mensen zonder inkomen, bijvoorbeeld jongeren, ouderen en werklozen).

## Verdeling van categoriale variabelen: tabellen

Voor een overzicht van scores op de 10-puntsschaal voor vervreemding, kunnen we gebruik maken van de **table()** en **prop.table()** functies. In de **table()** functie komt tussen haakjes de naam van een variabele; in de **prop.table()** functie komt tussen de haakjes een tabel (dat wil zeggen, een object dat is gemaakt met de **table()** functie). Het is overzichtelijk om eerst een tabel aan te maken en te bewaren als een object. Dit kan als volgt:

```
> tabel1 <- table(vv$vervreemding); tabel1
 1  2  3  4  5  6  7  8  9 10
10  9  8 13 13 10  5  8 12 12

> prop.table(tabel1)
   1    2    3    4    5    6    7    8    9   10
0.10 0.09 0.08 0.13 0.13 0.10 0.05 0.08 0.12 0.12

> cbind(prop.table(tabel1))
   [,1]
1  0.10
2  0.09
3  0.08
4  0.13
5  0.13
6  0.10
7  0.05
8  0.08
9  0.12
10 0.12
```

De output van de **table()** functie is een rij met de voorkomende waarden (1 t/m 10), gevolgd door een rij met het aantal keren dat die waarde voorkomt (de frequentie; 10 personen scoren een 1; 13 personen scoren een 5; enzovoorts). De **prop.table()** functie geeft proporties in plaats van frequenties. Er zijn diverse packages die beschrijvende informatie geven. Soms is het handig om zelf een functie te schrijven. Hieronder is een functie genaamd **CFR()** geschreven die als argumenten heeft een variabele en het aantal decimalen in de output (zeg, 2). De output lijkt veel op de output die SPSS geeft. Zelfgeschreven functies zijn objecten in **R** en moeten dus elke keer als ze nodig zijn gerund worden! Het schrijven van functies valt buiten het bestek van deze module.

```
Frequencies (absolute/relative/cumulative)
LOI_Machine Learning/2016

    f cum.f rel.f cum.rel.f
1  10    10  0.10      0.10
2   9    19  0.09      0.19
3   8    27  0.08      0.27
4  13    40  0.13      0.40
5  13    53  0.13      0.53
6  10    63  0.10      0.63
7   5    68  0.05      0.68
8   8    76  0.08      0.76
9  12    88  0.12      0.88
10 12   100  0.12      1.00
```

Met de **CFR()** functie kunnen we snel zien dat (i) 13 personen een 5 scoren; (ii) 53% van de personen een 5 of lager scoren; (iii) er in totaal 100 personen een score hebben.

We zijn vooral geïnteresseerd in relaties tussen de variabelen. Voor relaties tussen nominale (of categorale) variabelen zoals religie en regio kunnen we ook gebruik maken van **table()** functies, maar dan met twee of zelfs meer variabelen. We beperken ons tot relaties tussen twee variabelen. Een tabel van religie naar regio kunnen we als volgt maken. De functie **table()** geeft frequenties; en **prop.table** geeft proporties. De toevoegingen 1 en 2 in de

**prop.table()** functie vragen om rij- of kolompercentages. De **round()** functie rondt af in 2 decimalen. We lezen af dat:

- Van de rooms-katholieken 79% in het zuidoosten woont.
- In het zuidoosten 54% van de personen rooms-katholiek is.

```
> table2 <- table(religieFac, regioFac); table2
            regioFac
religieFac   NW ZO
  Anders     16 13
  Protestant 27 10
  RK          7 27
> round(prop.table(table2, 1), 2)
            regioFac
religieFac     NW   ZO
  Anders     0.55 0.45
  Protestant 0.73 0.27
  RK         0.21 0.79
> round(prop.table(table2, 2), 2)
            regioFac
religieFac     NW   ZO
  Anders     0.32 0.26
  Protestant 0.54 0.20
  RK         0.14 0.54
```

## Analyse van tabellen met het gmodels package

Een meer overzichtelijke tabel kan worden gemaakt met de **CrossTable()** functie uit het package **gmodels**. Vergeet niet eerst het package te installeren, en vervolgens aan te roepen met **library(gmodels)**! De output van de **CrossTable()** functie lijkt erg op de output van SPSS.

In het commando hebben we ook om de zogenoemde chi-square test gevraagd. Er wordt getest of de samenhang tussen de twee variabelen al of niet toevallig is.

De redenering is als volgt. Als er geen samenhang zou zijn dan zouden we verwachten dat in beide regio's verhoudingsgewijs evenveel protestanten als rooms-katholieken voorkomen. Maar in regio 1 (NW) is 54% protestant en 14% katholiek, en in regio 2 is dat 20% en 54%. Hoe groter de verschillen zijn, des te groter wordt de chi-square uitkomst. Er kan statistisch berekend worden dat de kans op een chi-square van 19.89 voor een 3*2 tabel heel erg klein is ($4.81*10^{-5}$, oftewel 0.00481%). We concluderen dat het verband statistisch significant (betekenisvol) is: het kan bijna geen toeval zijn!

```
> library(gmodels) # installeer gmodels als u dat nog niet gedaan hebt!
> # install.packages("gmodels")
> CrossTable(religie, regio, prop.chisq=FALSE, chisq=TRUE, format="SPSS")


   Cell Contents
|-------------------------|
|                   Count |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|
```

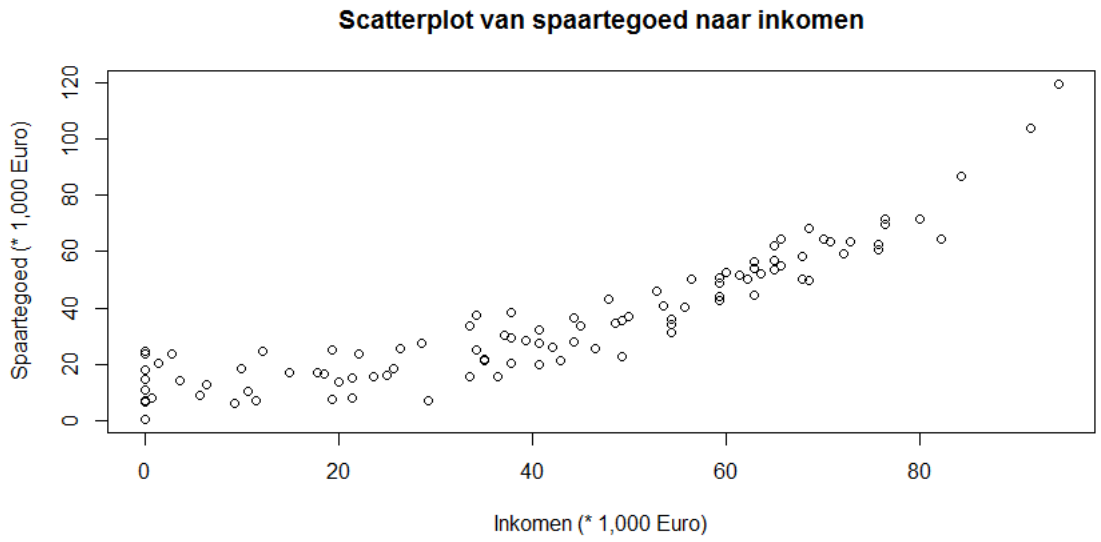
```
Total Observations in Table:  100

             | regio
     religie |         1  |         2  | Row Total |
-------------|------------|------------|-----------|
           1 |        27  |        10  |        37 |
             |    72.973% |    27.027% |    37.000% |
             |    54.000% |    20.000% |           |
             |    27.000% |    10.000% |           |
-------------|------------|------------|-----------|
           2 |         7  |        27  |        34 |
             |    20.588% |    79.412% |    34.000% |
             |    14.000% |    54.000% |           |
             |     7.000% |    27.000% |           |
-------------|------------|------------|-----------|
           3 |        16  |        13  |        29 |
             |    55.172% |    44.828% |    29.000% |
             |    32.000% |    26.000% |           |
             |    16.000% |    13.000% |           |
-------------|------------|------------|-----------|
Column Total |        50  |        50  |       100 |
             |    50.000% |    50.000% |           |
-------------|------------|------------|-----------|

Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  19.88586     d.f. =  2     p =  4.806623e-05
```

## Verkennen van verbanden tussen numerieke variabelen

Voor relaties tussen numerieke variabelen kunnen we geen tabellen gebruiken, omdat elke waarde (elk inkomen; elk spaartegoed) vermoedelijk uniek is. We kunnen wel de inkomens indelen in categorieën (bijvoorbeeld van 0 tot 10.000 ; van 20.000 tot 30.000 ; enzovoorts), maar daarmee verliezen we informatie, en het is bovendien niet nodig. Het verband tussen twee numeriek variabelen zoals inkomen en spaartegoed kan inzichtelijk worden gemaakt met een spreidingsdiagram (of scatterplot).

Als we aannemen dat het spaartegoed een functie is van inkomen, dan kunnen we een plot maken met spaartegoed op de verticale as (de y-as), en inkomen op de horizontale (x-)as. De grafiek laat zien dat er een positief verband is, zoals verwacht, tussen de twee variabelen: hoe hoger het inkomen des te groter het spaartegoed. De grafiek laat ook zien dat het spaartegoed niet lineair maar exponentieel toeneemt. De stijging is eerst geleidelijk maar neemt toe bij hogere inkomens (vanaf ongeveer 40.000).

```
plot(x=ink_1000, y=spaar_1000, main="Scatterplot van spaartegoed naar inkomen",
     xlab="Inkomen (* 1,000 Euro)", ylab="Spaartegoed (* 1,000 Euro)")
```



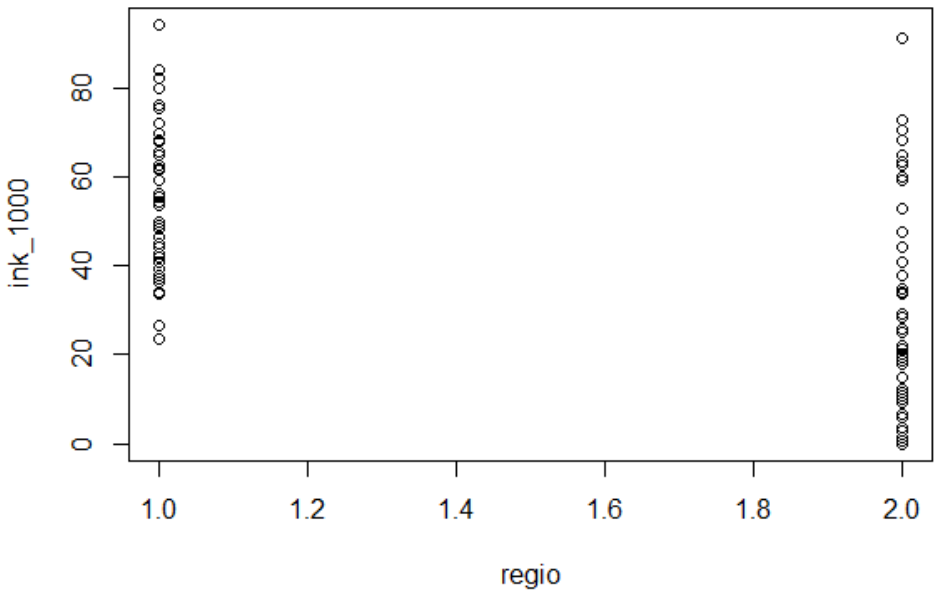

*Afb. 16. Scatterplot: verband tussen twee numerieke variabelen.* Bron: OGN.

## Verkennen van verbanden tussen numerieke en categoriale variabelen

We hebben geleerd om relaties te verkennen tussen twee categoriale (of: nominale) variabelen (met tabellen), en tussen twee numerieke variabelen.

Maar wat nu als we de relatie willen onderzoeken tussen een numerieke variabele (zoals inkomen) en een nominale variabele (zoals regio)? Zeker als de nominale variabele niet al te veel categorieën telt kunnen we nog steeds gebruikmaken van een scatterplot. We gebruiken dan een iets andere notatie: inkomen ~ regio ("Inkomen wordt verklaard uit regio"). De tilde ("~") als aanduiding voor "wordt verklaard uit" zullen we vaker tegenkomen bij de ML-toepassingen in volgende hoofdstukken!

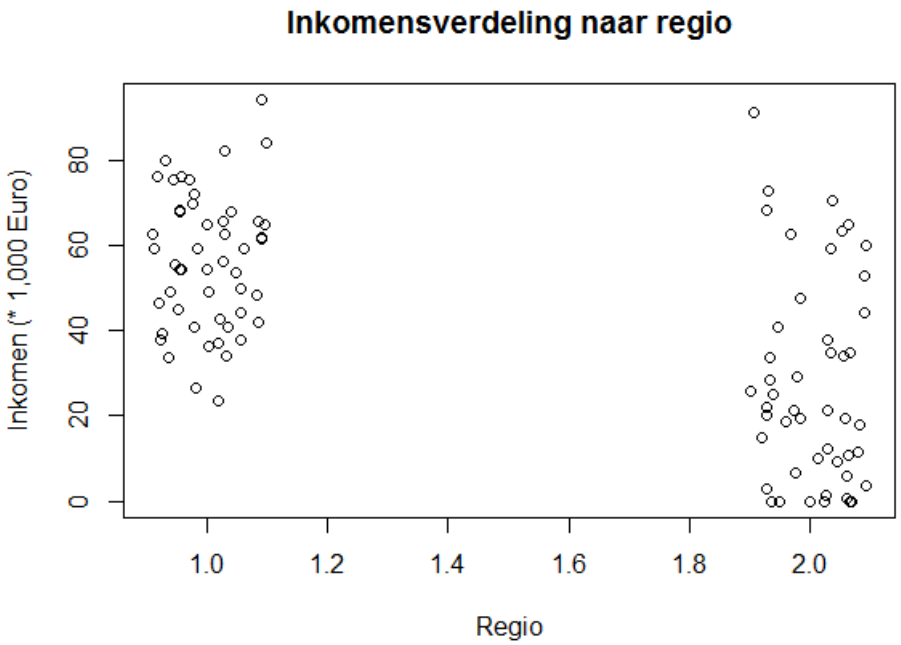
```
plot(ink_1000 ~ regio)
```




*Afb. 17. Scatterplot, met een numerieke en een categoriale variabele.* *Bron: OGN.*

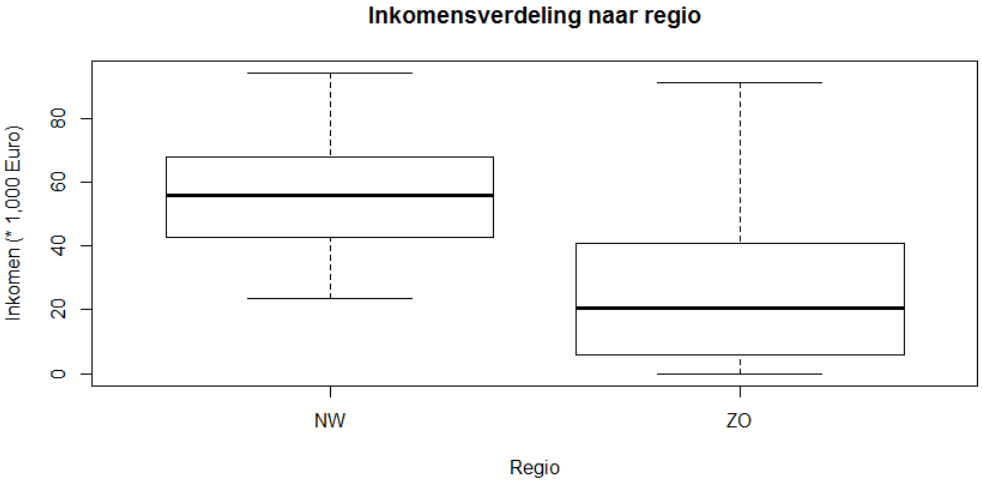
U ziet waarschijnlijk het nadeel van deze plot. Omdat regio maar twee waarden kan aannemen (1 of 2) worden veel punten over elkaar geprint. Zouden we heel veel observaties hebben, en zouden alle inkomens voorkomen in beide regio's, dan resulteert uiteindelijk niets meer dan twee verticale lijnen die weinig informatie geven over de relatie. Om optimaal gebruik te maken van de grafische mogelijkheden in **R** gaan we een beetje smokkelen: we trekken de punten uit elkaar door een kleine random (willekeurige) afwijking aan regio toe te voegen. We noemen deze willekeurige afwijkingen jitter. U kunt de hoeveelheid jitter instellen, en het is een kwestie van trial and error om te komen tot de beste presentatie. In dit geval geeft een jitter van 0.5 een goed resultaat. We zien nu duidelijk(er) dat de inkomensverdeling in regio 1 (noordwest) hoger ligt dan die in regio 2 (zuidoost).

```
plot(ink_1000 ~ jitter(regio,.5), main="Inkomensverdeling naar regio",
xlab="Regio", ylab="Inkomen (* 1,000 Euro)")
```




*Afb.18. Scatterplot, met een numerieke en een categoriale variabele, en "jitter".* *Bron: OGN.*

We hebben hier gekozen voor de numeriek gecodeerde variabele **`regio`**. Als we in hetzelfde commando de factor **`regioFac`** gebruiken, dan geeft **R** een boxplot voor de twee regio's, en dit is natuurlijk een uitstekend alternatief!




*Afb. 19. Boxplot, voor een numerieke en een categoriale variabele.* *Bron: OGN.*