

Programmeren en data analyseren met R

Een praktische kennismaking om de R in de vingers te krijgen

Arie Twigt

januari 2017

Abstract

R is de open source statistische programmeertaal die de laatste jaren razend populair is voor data gerelateerde projecten. Door de uitgebreide mogelijkheden om data te analyseren, visualiseren, voorspellingsmodellen te ontwikkelen en andere databewerkingen toe te passen, geldt R als de facto (samen met Python) in de Data Science wereld voor data analyse. Het doel van dit boek is om de lezer een goed begrip te laten krijgen van de basis en de structuur van R. Door deze kennis moet de lezer uiteindelijk in staat zijn om zelf analyses kunnen uit te voeren.

Contents

1	Voorwoord	6
2	Inleiding	7
2.1	De programmeertaal R	7
2.2	De structuur van R	8
2.3	Data bewerken	8
2.4	Data analyseren	9
2.5	Geen big data	9
2.6	Geen uitgebreide statistiek of econometrie	9
2.7	Wat je leert	9
2.8	Wat je eraan hebt	10
2.9	Samenvattingen en oefeningen	10
2.10	Termen	10
2.11	Legenda	11
2.12	Installatie software en bestanden	12
2.13	R en RStudio	12
2.14	Datasets	16
2.15	Referenties	16
3	De basics van de R-syntax	17
3.1	Variabelen aanmaken in R	18
3.2	Verschillende klassen in R	21
3.3	Logical waarden en operators	24
3.4	Samenvatting	25
3.5	Oefeningen	25

4	Reken- en statistische functies in R	26
4.1	Eenvoudige rekensommen	27
4.2	Simpele statistische functies gebruiken in R	28
4.3	Samenvatting	32
4.4	Oefeningen	32
5	Data analyseren en bewerken in R	34
5.1	Ingebouwde dataset laden in R	35
5.2	Eenvoudige manieren om de dataset te verkennen	35
5.3	Eenvoudige statistische calculaties uitvoeren in R	37
5.4	Samenvatting	39
5.5	Oefeningen	40
6	Data importeren en exporteren in R	41
6.1	Een ingebouwde dataset openen in R	42
6.2	Een lokale dataset importeren in R	43
6.3	Een online dataset in R importeren	46
6.4	Variabelen van de dataset hernoemen	50
6.5	Omgaan met lege waarden (NA's)	52
6.6	De <code>attach()</code> functie gebruiken	54
6.7	Een data frame of matrix exporteren	55
6.8	Samenvatting	55
6.9	Oefeningen	56
7	Data frame bewerkingen	57
7.1	De “mtcars” dataset	58
7.2	Rijen en kolommen van een data frame selecteren	58
7.3	Kolommen uit een data frame selecteren op basis van de naam	60
7.4	Filters toepassen op data frames	60
7.5	Berekende kolommen toevoegen	61
7.6	Kolommen met een bepaalde waarde selecteren	61
7.7	Kolomnamen van een data frame veranderen	62
7.8	Samenvatting	63
7.9	Oefeningen	63

8	Werken met lijsten en matrixen in R	64
8.1	Handmatig een matrix maken	65
8.2	Lijsten (lists) maken in R	66
8.3	Een matrix samenstellen op basis van lijsten	68
8.4	Data uit de matrix selecteren	69
8.5	Calculaties maken met matrixen	69
8.6	Samenvatting	71
8.7	Oefeningen	72
9	Het <i>factor</i> datatype	73
9.1	Het “factor”-datatype in R	74
9.2	Een data frame plotten	75
9.3	Het factor-datatype definiëren	76
9.4	Data in de juiste volgorde plotten met factors	77
9.5	Handig gebruik maken van factor vectoren	79
9.6	Een data frame sorteren rangschikken	80
9.7	Samenvatting	83
9.8	Oefeningen	83
10	Eenvoudige grafieken maken in R	84
10.1	De data frame voorbereiden	85
10.2	Een eenvoudige grafiek maken in R	85
10.3	Staafdiagrammen en histogrammen	90
10.4	Een boxplot maken in R	93
10.5	Samenvatting	95
10.6	Oefeningen	95
11	Case 1: Transactiedata analyseren	97
11.1	De CDNOW-dataset importeren	98
11.2	De eigenschappen van de dataset bekijken	99
11.3	De dataset de juiste vorm geven	99
11.4	Spelenderwijs de data verkennen	101
11.5	Totale uitgave per klant berekenen	111
11.6	Klanten Segmenteren	112
11.7	Klantnummers verzamelen per klantgroep	114
11.8	Informatie uit transactiedata presenteren	114
11.9	Informatie presenteren over de perioden	115

12	Correlaties, functies en loops in R	117
12.1	De definitie van een correlatie	118
12.2	Correlaties berekenen	118
12.3	Correlatiematrix maken	119
12.4	De correlatiematrix maken	131
12.5	Samenvatting	131
12.6	Oefeningen	132
13	Lineair regressiemodel	133
13.1	De dataset	134
13.2	Een lineair regressie model maken met de <code>lm()</code> functie	134
13.3	De regressieanalyse uitvoeren	134
13.4	Meervoudige Regressieanalyse	137
13.5	Categorische variabelen	138
13.6	Samenvatting	141
13.7	Oefeningen	141
14	Uitgebreide visualisaties maken in R met ggplot	142
14.1	Het ggplot2 package installeren	143
14.2	Visualiseren in R met ggplot	143
14.3	De <i>kleur</i> dimensie	144
14.4	De <i>grootte</i> dimensie	146
14.5	Voeg een extra dimensie toe op basis van vorm in <i>ggplot</i>	147
14.6	Andere visualisaties in ggplot	148
14.7	Samenvatting	150
14.8	Oefeningen	151
15	Case 2: Open data ontsluiten	152
15.1	Importeren van de dataset via de API van <i>Open data overheid</i>	152
15.2	De API met JSON-data ontsluiten met het <i>jsonlite</i> package	153
15.3	De dataset bewerken	153
15.4	Extra, berekende, kolommen toevoegen aan de data frame	154
15.5	Prijsverschillen per dag	155
15.6	Namen van de weekdays weergeven	155
15.7	Weekdagen in goede volgorde zetten	156
15.8	Weergeven of het een weekday of een dag in het weekend was	156
15.9	Visualisaties in ggplot	157

16 Antwoorden voor oefeningen	173
16.1 De basics van de R-syntax	173
16.2 Reken- en statistische functies in R	173
16.3 Data analyseren en bewerken in R	174
16.4 Data importeren en exporteren in R	176
16.5 Data frame bewerkingen	176
16.6 Werken met lijsten en matrixen in R	178
16.7 Handig gebruik maken van het “factor” datatype	178
16.8 Correlaties, functies en loops in R	180
16.9 Lineaire regressiemodel	181
16.10 Uitgebreide visualisaties maken in R met ggplot	182
17 Jouw data avontuur is nog maar net begonnen...	185
17.1 Experimenteer!	185
17.2 Andere platformen om R te leren	185
17.3 Laat je inspireren en join the club	185
18 We houden contact	186
19 Referenties en links	187
20 Over de Auteur	189

1 Voorwoord

Allereerst, bedankt dat je dit boek erbij pakt om kennis te maken met de open source statistische programmeertaal R. Daarbij wil ik gelijk mijn vrouw bedanken die voor de illustraties heeft gezorgd in dit boek. Ze zijn handmatig met potloden op tekenpapier gemaakt en daarna met Photoshop bewerkt. Erg bedankt!

Ook wil ik mijn Sogeti collega Dylan van Riel bedanken voor de uitgebreide review van dit boek. Dylan leidt de Technology & Tooling groep op het gebied van Analytics binnen Sogeti, waar ook de programmeertalen R en Python worden behandeld. Dankzij zijn tips en scherpe review durf ik dit boek met een gerust hart uit te geven.

R is de afgelopen jaren mijn stokpaardje geworden. Tijdens mijn studie *Business Administration - Information & Knowledge Management* op de Vrije Universiteit Amsterdam heb ik het vak *Business Intelligence* gekregen en kwamen er onderwerpen als **predictive analytics** (voorspellingsmodellen) en uitgebreide data-analyse aan bod. Omdat ik tussen het bestuderen van papers ook graag de praktijk in wilde duiken, ben ik hiermee naar de professor gegaan, Frans Feldberg, die mij adviseerde om eens een kijkje te nemen naar de statistische programmeertaal **R**. “R? Hoe spel je dat?” “Gewoon de letter R”. Uiteindelijk is dit het gesprek dat ik nu heel vaak voer met mensen aan wie ik uitleg geef over R. Zoek voor de gein maar eens op waarom de programmeertaal “R” heet. Uiteindelijk ben ik er zo enthousiast over geworden dat ik er bijna dagelijks mee experimenteerde. Uit mijn enthousiasme heb ik daar uiteindelijk een handleiding van geschreven. Dat is niet deze handleiding, maar de handleiding die ik in 2013 geschreven heb: Handleiding R. Deze handleiding is door de komst van dit boek achterhaald.

Afgelopen jaren heb ik bij verschillende klanten oplossingen met R geïmplementeerd of de klant laten kennismaken met R. Hierbij heb ik gezien dat R over het algemeen positief wordt ontvangen en zeer toepasbaar is in het bedrijfsleven. Voorbeelden zijn klanten die voor processen met complexe berekeningen overstappen van pakketten als MatLab, SPSS of Excel naar R. Het is makkelijk te implementeren, de mogelijkheden zijn eindeloos en het is gratis. Het feit dat R open source is, zorgt af en toe voor een licht drempeltje bij klanten. Gelukkig heeft Microsoft sinds 2015 R in verschillende software geïntegreerd, bijvoorbeeld op **Microsoft Azure**, **Microsoft Visual Studio** (met *R tools for Visual Studio*) en **Microsoft SQL Server Management Studio 2016**. Ook heeft Microsoft een eigen versie van R, **Microsoft R open**. Er staat je dus een mooie toekomst te wachten als je R onder knie hebt.

Nogmaals, wil ik je erg bedanken dat je dit boek erbij hebt gepakt. Met de opbouw en alle instructies wil ik de denkwijze die ik heb en de methoden die ik gebruik voor de basics naar jou overbrengen. Natuurlijk zijn er mensen die dit op een andere manier aanpakken, dit zal je naar verloop van tijd zelf ook doen als je het aardig onder de knie krijgt. In ieder geval zal dit boek je helpen met het onder de knie krijgen van de basics en zal je alle aspecten bezitten om uitgebreid data te kunnen analyseren met R. Ik ben er van overtuigd dat je net zoveel plezier mee zult hebben dit boek te lezen en de instructies uit te voeren, als het plezier dat ik heb ervaren tijdens het samenstellen van dit boek.

2 Inleiding

2.1 De programmeertaal R

R wordt over het algemeen een open source statistische programmeertaal genoemd. Deze beschrijving is een hele mond vol, maar bevat wel de belangrijkste aspecten van deze programmeertaal:

- Open source;
- Statistisch;
- Programmeertaal.

2.1.1 Open source

R is een open source programmeertaal, dat betekent dat niemand, geen organisatie en geen persoon, de eigenaar is van R en je het dus overal gratis voor kunt gebruiken. Daarbij geeft open source aan dat de broncode waarmee de programmeertaal is opgebouwd beschikbaar is. Je kunt bijvoorbeeld bekijken met welke broncode de functies van R zijn opgebouwd. Een ander belangrijk aspect van open source in dit geval is dat je alle vrijheid hebt om onderdelen aan te passen en er extra mogelijkheden aan kunt toevoegen. De mogelijkheden hierin zijn eindeloos, in die zin dat mogelijkheden niet bewust zijn afgesloten omdat R niet is dichtgetimmerd.

Een belangrijk voorbeeld hiervan zijn de verschillende *packages* die beschikbaar zijn in R. *Packages* zijn verzamelingen van extra functies die geschreven zijn door andere ontwikkelaars en over het algemeen vrij beschikbaar worden gesteld. Deze packages maken de programmeertaal zeer krachtig, flexibel en toepasbaar voor verschillende werkzaamheden, sectoren en branches. Zo zijn er bijvoorbeeld packages die het makkelijk maken om *Twitter* data te importeren (het **twitter** package), packages die formules van financiële calculaties bevatten (het **FinCal** package), packages waarmee je analytics kunt uitvoeren door Google Analytics data te importeren in R (het officiële **RGoogleAnalytics** package van Google) en packages om op een prachtige manier data te visualiseren (bijvoorbeeld het **ggplot2** package). Dit maakt R een Zwitsers zakmes als het gaat om werken met data en is dit voor een groot deel te danken aan het feit dat het een open source programmeertaal is.

Packages zijn goed gedocumenteerd en worden beheerd in de Comprehensive R Archive Network (CRAN). Dit is ook de plek waar over het algemeen de belangrijkste packages staan en worden beheerd door de community van R. Deze packages hebben aan een aantal belangrijke criteria voldaan. De documentaties zijn over het algemeen te vinden in een overzichtelijk pdf-bestand. In deze documentatie kun je alle functies van het package bekijken, contactgegevens vinden van de ontwikkelaar en met voorbeelden zien hoe functies van het package worden toegepast. Naast CRAN, zijn er ook steeds meer packages te vinden op **Github**. Deze werken gewoon op dezelfde manier als packages die op CRAN staan, de installatie is alleen iets anders. In dit boek wordt er ook verschillende keren gebruik gemaakt van packages. Je zult zien dat packages zeer eenvoudig te installeren en te gebruiken zijn.

2.1.2 Statistisch

R wordt een statistische programmeertaal genoemd omdat het standaard een uitgebreide collectie van statistische functies bevat. Dit betekent absoluut niet dat R beperkt wordt tot het vakgebied statistiek of dat het minder handig is voor andere vakgebieden. De reden dat R deze statistische functies bevat is omdat het oorspronkelijk ontwikkeld is om (statistisch) onderzoek mee uit te voeren. Je kunt het hierbij vergelijken met softwarepakketten als *SPSS* en *MatLab*. Je zou op dezelfde manier kwantitatief onderzoek kunnen doen of je scriptie kunnen schrijven in R. Statistische toetsen, visualisaties of formules zijn zeer eenvoudig met functies

toe te passen in R zonder dat je daar een speciaal *package* voor hoeft te installeren. Deze functies maken het data analyseren veel makkelijker. Naast de ingebakken statistische functies bevat R namelijk een breed scala aan functies om databewerkingen of analyses te doen. Bij meer uitgebreide analyses kunnen bepaalde statistische handelingen van toepassing komen. In andere programmeertalen zal je dan de formules van deze statische handelingen handmatig moeten programmeren, dit hoeft in R echter niet om dat zo goed als elke statische functie in R zit. Dit zorgt ervoor dat je een extreem krachtige tool hebt om data te analyseren, zelfs als het een stuk ingewikkelder wordt. Daarom kun je het statische aspect van R eerder zien als een waardevolle toevoeging dan een beperking.

2.1.3 Programmeertaal

Het feit dat R een programmeertaal is, zorgt voor veel flexibiliteit. Het werk dat je gedaan hebt in R is in de meeste gevallen een reeks aan commando's die samengevoegd worden tot een script. In een script worden de commando's van boven naar beneden uitgevoerd, net zoals bij andere script-gebaseerde programmeertalen (*scripting languages*). Een script in R wordt een R-script genoemd en heeft de `.r` extensie (bijvoorbeeld: `analyse.r`). Dit script kun je overal mee naartoe nemen en in verschillende systemen toepassen. Zo kun je bijvoorbeeld R code toepassen om data uit een database aan te passen, verschillende databases zijn namelijk in staat om R-code uit te voeren (bijvoorbeeld **Oracle** en **Microsoft SQL Server (vanaf 2016)**). Ook op Big Data platformen en software zoals **Hadoop** en **Spark** kun je R code schrijven om data te bewerken.

Het werk dat je maakt in R is dus op veel verschillende plekken toepasbaar en kun je eenvoudig meenemen en delen. Als je nu bijvoorbeeld Excel neemt, iets dat in Excel gemaakt is blijft in Excel. De logica en code die je in Excel gebouwd hebt kun je niet overhevelen naar een ander systeem (hooguit kun je de data delen), deze zitten namelijk in het Excel-bestand gebakken. Functies en handelingen die bouwt in R, kunnen op veel meer plekken gebruikt worden. Dit geldt trouwens voor de meeste programmeertalen, de code is los te trekken van de omgeving waar het in is gebouwd: *“Code once, use everywhere.”*.

Programmeren klinkt voor sommigen misschien heel eng en code kan er soms heel ingewikkeld uitzien. Je moet je echter hierdoor niet laten afschrikken. Het is natuurlijk nodig dat je de syntax van R begrijpt en weet wanneer je welke code moet toepassen. Hier zit een leercurve in die bij R soms als vrij stijl wordt ervaren. Echter kun je deze overwinnen met genoeg oefening waardoor je een beter begrip krijgt van de syntax en de structuur waardoor je voor jezelf een eigen manier zal vinden om R goed onder de knie te krijgen. Als je hebt gewerkt met formules in Excel, kun je sommige onderdelen van die kennis meenemen om R leren te begrijpen. Het is daarbij ook een verschil in mindset: R is geen software waarmee je op knopjes drukt en kan klikken en slepen, maar een programmeertaal waarmee aan de hand van commando's een reeks aan handelingen geeft als opdracht om door R uit te laten voeren.

R is flexibel door de vele mogelijkheden, de vele verschillende datatypes en de vele datastructuren. Maar deze flexibiliteit kan complex en ingewikkeld overkomen als je er zomaar in duikt en begint met programmeren zonder enig begrip van de structuur van R. Daarom worden er in dit boek stapje voor stapje de belangrijkste structuren en datatypes in R uitgebreid behandeld.

2.2 De structuur van R

Het is de bedoeling dat je na het lezen en het volgen van de instructies in dit boek programmeren en data analyseren met R goed onder de knie hebt gekregen. Daarom beginnen we bij het begin, namelijk de bouwstenen van de structuur in R. Zonder goede kennis van de structuur in R loop je het gevaar dat je tijdens het data analyseren in de knoop komt, bijvoorbeeld door dingen te willen doen die qua structuur helemaal niet kunnen.

2.3 Data bewerken

In bijna elk data-analyse proces moet de data eerst worden bewerkt voordat er mee geanalyseerd kan worden. De instructies geven je goede voorbeelden hoe je dit in R kunt doen. In veel gevallen kan data

na het ondergaan van een aantal bewerkingen goed geanalyseerd worden. R heeft een zeer breed scala aan mogelijkheden om data te bewerken. De bewerkingen in dit boek zijn qua niveau logisch opgebouwd van eenvoudig naar geavanceerd. Daarbij zijn veel instructies en voorbeelden van bewerkingen die ik tijdens projecten bij klanten ben tegengekomen. Ik zal je dus geen nutteloze trucjes of handelingen leren, in ieder geval zo weinig mogelijk.

2.4 Data analyseren

Er zijn eindeloos veel mogelijkheden om data uitgebreid te analyseren. Zowel visueel of bijvoorbeeld statistisch. Ik heb nooit de illusie gehad om alle mogelijkheden in één boek te stoppen. Echter zit er wel een rode lijn in dit boek op het gebied van data analyseren. Onderwerpen zijn bijvoorbeeld analyses door te *slicen en dicen* zoals de Business Intelligence wereld dat noemt, statistische functies, visualisaties en speciale R-functies. Ook deze heb ik onder andere toegepast bij klanten en hebben waardevolle inzichten opgeleverd. Dit kan voor jou, of voor het bedrijf waar je voor werkt, zeer waardevol zijn.

2.5 Geen big data

Dit boek behandelt niet het onderwerp *Big Data*. Dit is data die te groot is om door een enkele CPU op je machine bewerkt te kunnen worden. Hiervoor heb je geavanceerde software nodig om meerdere computers aan elkaar te koppelen voor meer rekenkracht. Bekende software-omgevingen die dit doen zijn bijvoorbeeld **Apache Hadoop** en **Apache Spark**. Tegenwoordig ook verschillende Microsoft-producten zoals Microsoft R server of Microsoft **Azure Batch Services**. Het leuke aan deze software is dat je er R in kunt programmeren. Hierdoor wordt het mogelijk om R-bewerkingen te doen op gigantische datasets, datasets met tientallen of zelfs honderden miljoenen aan rijen aan data. Als je R onder de knie hebt, kun je altijd naar de websites gaan die deze *Big Data* oplossingen aanbieden. Je zult de R code herkennen en daardoor de stap kunnen maken naar *Big Data*.

In het artikel Microsoft Azure Batch: Doorlooptijden terugbrengen van dagen naar uren, leg ik uit hoe Microsoft Azure Batch Services in zijn werk gaat en het kunt gebruiken om meer rekenkracht te hebben voor R calculaties.

2.6 Geen uitgebreide statistiek of econometrie

Naast *Big Data* wordt er in dit boek ook geen uitgebreide uitleg gegeven over geavanceerde wiskundige modellen. Je krijgt echter wel voldoende informatie over de statistische functies die je kunt toepassen in R. Bijvoorbeeld het onderwerp *Lineaire regressieanalyse* in dit boek legt uitgebreid de statische betekenis en onderbouwing van de output van de functie uit. Hiermee kun je klanten goed uitleggen wat je toepast en wat de uitkomsten daarvan zijn. Een lineaire regressie is een zeer populair voorspellingsmodel, maar slechts een van de vele mogelijkheden van voorspellende analyses. Ondanks dat is het een mooi begin en zal goede kennis van R die overgebracht wordt in dit boek je helpen als je naar meer uitgebreide wiskundige of statistische modellen wilt kijken.

2.7 Wat je leert

Dit boek kan jouw eerste kennismaking met R zijn. Als je al vaker met R hebt gewerkt, is dit boek alsnog een goede toevoeging aan jouw R kennis. De opbouw en de structuur van R worden namelijk op een zo duidelijk en compleet mogelijke manier uitgelegd. In dit boek wordt er zoveel mogelijk een afweging gemaakt om de basics uit te leggen zonder dat het geestdodend eenvoudig en saai is. Daarbij zijn onderwerpen die te diep gaan en over het algemeen voor werkzaamheden niet relevant zijn, weggelaten. Het is de bedoeling dat je aan het einde van dit boek de belangrijkste onderdelen van de R structuur onder de knie hebt en de

eigenschappen van R-objecten kent. Met deze kennis zal je aan het einde van dit boek R goed in de vingers hebben en snelheid opbouwen bij het programmeren in R. Daarbij is dit boek een mooi naslagwerk waarmee je verschillende commands kunt opzoeken. Je hoeft er namelijk niet naar te streven om de hele syntax uit je hoofd te leren, zelfs de beste programmeurs gebruiken Google.

2.8 Wat je eraan hebt

R is een waardevolle toevoeging aan je skillset en CV. Vooral bij vacatures voor een rol als bijvoorbeeld Data Scientist staat R op het lijstje van vereiste skills. Andere tools als bijvoorbeeld Python kom je ook vaak tegen. Python is ook een uitstekende tool om data te analyseren. Van oorsprong is Python meer een echte ontwikkeltaal en geeft het daarom meer mogelijkheden bij het ontwikkelen van data gedreven applicaties. Alhoewel je met R ook applicaties kunt bouwen, zou je grofweg kunnen zeggen dat je voor de beste ervaring bij het uitgebreid analyseren van data kiest voor R en als je iets wilt ontwikkelen je kiest voor echte ontwikkelten zoals Python, Java of Scala. Daar staat wel tegenover dat steeds meer gerenommeerde softwarepakketen kiezen voor de integratie met R zoals bijvoorbeeld Microsoft dat doet. Zelf heb ik bijvoorbeeld een opdracht gedaan met R bij een grote verzekeraar die over het algemeen een Microsoft IT architectuur heeft. Omdat Microsoft Visual Studio en Microsoft SQL Server 2016 een uitgebreide R plugin heeft, was R de taal waarmee de uitgebreide calculaties geprogrammeerd werden. Om deze redenen is programmeren in R een waardevolle skillset waar je bij bedrijven en in de toekomst veel aan zult hebben.

*Met R kun je ook Webapplicaties en Dashboards bouwen. Zelf heb ik op dit moment bijvoorbeeld de webapplicatie **TweetExplorer** tweetexplorer.com geschreven, waarmee Tweets eenvoudig geanalyseerd kunnen worden. Data gedreven webapplicaties kun je volledig in R schrijven in combinatie met een beetje HTML, CSS en Javascript.*

2.9 Samenvattingen en oefeningen

Ieder hoofdstuk eindigt met een samenvatting en oefeningen. Omdat het een boek is dat je kunt volgen door instructies uit te voeren in jouw eigen R-sessie, kan het afronden van een hoofdstuk langer duren dan als je het alleen leest. Een samenvatting geeft een mooie afronding van het hoofdstuk en vat mooi samen welke onderwerpen er in het hoofdstuk zijn besproken. Naast de samenvattingen heeft ieder hoofdstuk ook twee of drie oefeningen. Deze oefeningen zijn gemaakt om je los van de instructies stukken R code te schrijven waardoor je wordt gedwongen je eigen opgedane kennis toe te passen. Hiervoor kun hulp vinden in het betreffende hoofdstuk, maar voor sommige vragen moet je zelf een creatieve manier vinden of Google raadplegen. Hierdoor wordt je nog beter getraind en ervaar je de pijn die je tijdens projecten ook kan hebben als je naar een bepaalde oplossing zoekt. Mocht je er echt niet uit komen, kun je de antwoorden van de oefeningen vinden aan het einde van dit boek in het hoofdstuk *Antwoorden voor oefeningen*.

2.10 Termen

2.10.1 Variabele

Een variabele is een object dat een bepaalde waarde bevat. Hierbij kan een variabele alleenstaand zijn of onderdeel maken van een groter object zoals een data frame. In dat geval is een variabele een kolom in een data frame.

2.10.2 Object

Een object is een entiteit in R. Er zijn verschillende typen objecten in R zoals data frames (tabellen), matrixen, lijsten (lists) en vectoren. Ieder type object vervult zijn eigen functie tijdens een R-sessie. Hierbij verschillen de objecten qua mogelijkheden om deze te bewerken.

2.10.3 Working Directory

De *Working Directory* is een belangrijk onderdeel in R. Het is namelijk de map op jouw machine waar de R-sessie op zoek gaat naar bestanden als deze geïmporteerd moeten worden of waar R bestanden naar exporteert, bijvoorbeeld databestanden of ander r scripts. In programmeertermen kun je zeggen dat de *Working Directory* een ander woord is voor de *root directory* van de huidige R-sessie. Je kunt het `getwd()` command gebruiken om te kijken wat de *working directory* voor de huidige R-sessie is. Met het `setwd()` command kun je de *working directory* veranderen.

2.10.4 Project

Een project is eigenlijk geen R entiteit. Het is een object dat aangemaakt wordt door RStudio en kan daarom ook alleen door RStudio gebruikt worden. Een project is een handige *wrapper* voor de gehele omgeving van de R-sessie. Als je bijvoorbeeld bezig bent met een bepaalde analyse en je hebt hiervoor verschillende bestanden geopend, waarden aangemaakt en een working directory ingesteld, wordt dit allemaal bijgehouden en teruggehaald zodra je het project weer opent.

2.10.5 Functie

Een functie is een reeks aan handelingen en bewerkingen op een waarde in R. Net zoals bij andere programmeertalen zijn functies een van de belangrijkste bouwsetenen van R. R komt standaard met een breed scala aan functies waarmee je uitgebreid data kunt analyseren. Daarbij kun je packages installeren die andere functies bevatten die R niet heeft. Deze packages zijn verzamelingen van functies die andere personen gemaakt hebben en beschikbaar stellen. Ten slotte kun je zelf functies schrijven en jouw verzameling aan functies samenbrengen onder een package welke je eventueel beschikbaar stelt voor andere personen. Op deze manier is de cirkel mooi rond en blijft de programmeertaal R bloeien.

2.10.6 Dataset

Een dataset is een verzameling gegevens die in een R-sessie kan worden ingeladen. Dit kan in de vorm van een tabel, matrix, of lijst zijn. Op deze datasets kun je bewerken of gebruiken om er verschillende functies op toe te passen. Door dit op verschillende manieren te doen, kun je verschillende inzichten uit een dataset halen.

2.11 Legenda

2.11.1 Commentaar

In R is het mogelijk om commentaar te geven in het script. Deze regels worden door R overgeslagen tijdens het verwerken van de code. Commentaar is een handig middel om aan te geven wat er gebeurt bij verschillende plekken in het script, voor jezelf en ook voor anderen die het script lezen.

```
# dit is commentaar
```

2.11.2 Output uit de R-console

In sommige voorbeelden geeft R output terug. Deze output wordt aangegeven met `##`.

```
print("Dit is output :)")
```

```
## [1] "Dit is output :)"
```

2.12 Installatie software en bestanden

2.12.1 R installeren

Op officiële website van R, **The Comprehensive R Archive Network**(CRAN), kun je R downloaden. Zoals je kunt zien is R voor verschillende besturingssystemen beschikbaar.

Voor Windows, kies je voor *Download R for Windows*. Op de volgende pagina kies je voor *install R for the first time*. Hiermee download je de normale *base* versie van R, de standaard R met alle standaard functies. Simpel gezegd, “hiermee download je R.” Hierna volgt een wizard waarmee R wordt geïnstalleerd. Kies zo veel mogelijk voor de al standaard keuzes. Pas eventueel aan welke snelkoppelingen je wel en niet wilt hebben.

Als je een Mac OS gebruiker bent, kies je voor *Download R for (Mac) OS X*. Op de je komt hierdoor op een pagina waar verschillende download packages beschikbaar zijn, kies voor de meest recente versie. Als het package eenmaal gedownload is, kun je de eenvoudige stappen van het installatieproces volgen. De meest gebruiksvriendelijke keuzes zijn al door de installatie wizard geselecteerd.

2.12.2 RStudio installeren

Om RStudio te installeren ga je naar <https://www.rstudio.com/products/rstudio/#Desktop>. Kies voor de *Open Source Edition* door op **Download RStudio Desktop** te klikken. Hierdoor kom je op de pagina met alle beschikbare versies van RStudio voor verschillende platformen. Kies voor het platform waar je op werkt. Volg daarna de instructies van de wizard. Kies ook hier zoveel mogelijk voor de standaard mogelijkheden.

2.13 R en RStudio

Ik adviseer je om RStudio pas te installeren als R succesvol is geïnstalleerd. Het is gewoon mogelijk om RStudio te installeren zonder dat je R geïnstalleerd hebt, alleen werkt het pas als R ook geïnstalleerd is. R draait namelijk onder de motorkap van RStudio. Als je RStudio installeert nadat je R hebt geïnstalleerd, zoekt RStudio automatisch naar de locatie van R op op jouw machine. Als dit andersom doet, is er de kans dat je dit later in RStudio zelf moet configureren.

RStudio is een zeer uitgebreide en zeer gebruiksvriendelijke ontwikkelomgeving voor R. Het geeft je als gebruiker een veel beter beeld van waar je mee bezig bent tijdens een R-sessie. Daarbij heeft RStudio veel hulpmiddelen die je kunt gebruiken door er op te klikken, terwijl je daar voor in de normale R commands voor zou moeten gebruiken. Ik adviseer je daarom om RStudio te gebruiken. Mocht je toch liever de normale R-omgeving gebruiken, staat niets je daarvoor in de weg. Ook dan kun je alle instructies in dit boek probleemloos volgen.

2.13.1 De R-console

De R-console is een venster waarin je commands kunt invoeren. Het invoeren van commands is wat we in dit geval het *programmeren* noemen. Eigenlijk spreken we pas van programmeren als we een script bouwen dat uit een verzameling commands bestaat en dit door een systeem wordt uitgevoerd. Aangezien een script bestaat uit commands en je van de commands in dit boek ook uiteindelijk je eigen script kan maken, spreken we in dit boek uiteindelijk toch van programmeren.

De R-console reageert op ieder commando dat we invoeren. Grofweg doet de R-console twee dingen: opslaan en reageren. Dit wordt een *Read Evaluate Print Loop* (REPL) genoemd. Deze R-console is te zien in **Figure 1**. Als we een waarde aan een nieuwe variabele toewijzen, slaat R deze op en geeft hij pas deze waarde terug als we deze variabele weer oproepen. Als we echter een functie gebruiken of zelf een waarde invoeren, geeft R een reactie. Deze reactie is dezelfde waarde, of de waarde na de toepassing van een functie. In de onderstaande schermafbeelding geven we de waarde "Hello World!" aan de `print()` functie. R past de functie toe op deze waarde en geeft het resultaat terug. We gaan hier veel dieper op in tijdens de hoofdstukken in dit boek.

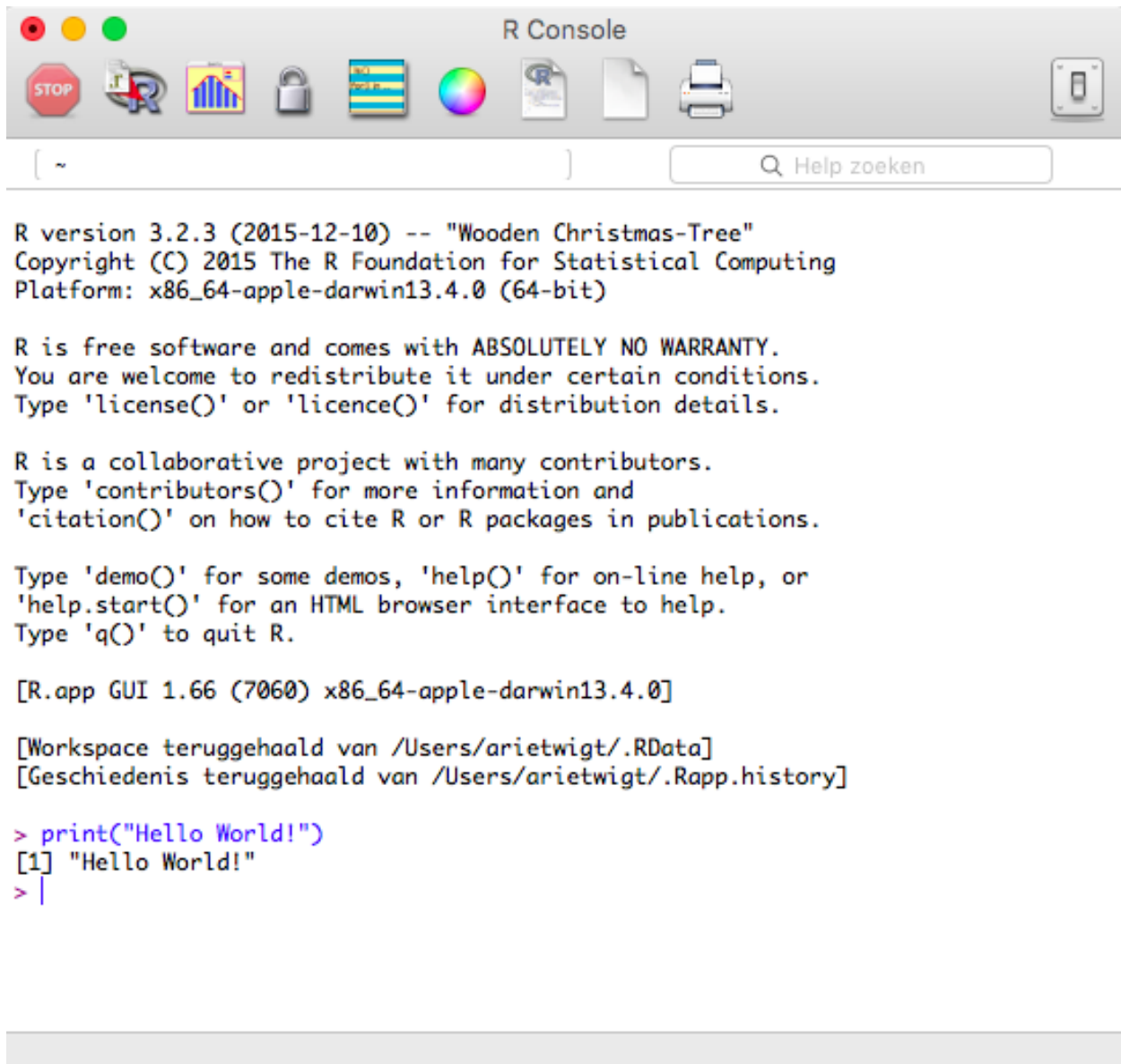


Figure 1: De R-console

2.13.2 De R Studio omgeving

In **Figure 2** zie je de **RStudio IDE**. *IDE* staat voor *Intelligent Development Environment* en dat beschrijft ook precies de toegevoegde waarde van RStudio ten opzichte van de standaard R-console. De RStudio is een stuk uitgebreider dan de R-console, maar zeker niet ingewikkeld. De extra vensters zijn namelijk visuele hulpmiddelen die eigenlijk verstopt zijn bij de normale R-console. Het venster links onderin herken je waarschijnlijk, het is namelijk exact dezelfde R-console. Deze werkt **exact** hetzelfde als de normale R-console. Een toevoeging in RStudio is dat hier de R-console *self complete* toepast op de commando's. Als je bijvoorbeeld een functie invoert of een object oproept, geeft de R-console een hint voor het commando dat je bedoelt.

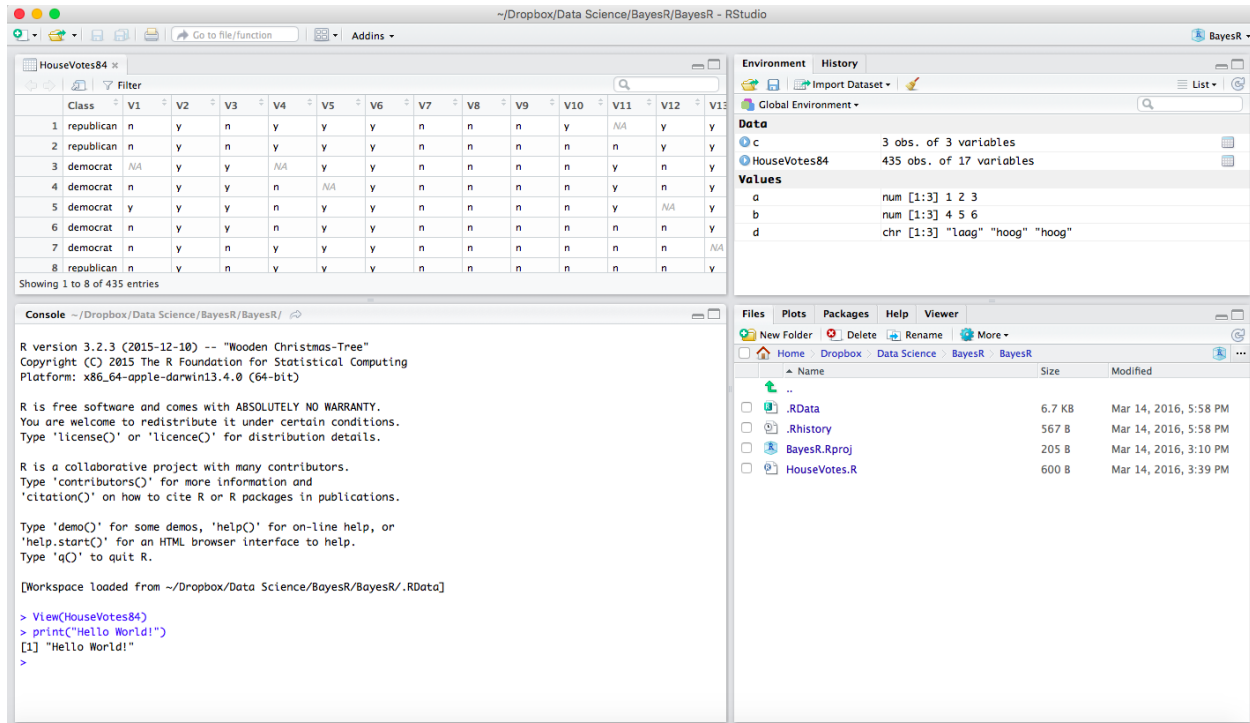


Figure 2: De R-console

Links bovenin zie je een dataset. Omdat je logischerwijs met datasets werkt in R, wil je af en toe weten hoe de dataset er uit ziet. Deze datasets worden dan op dit venster weergegeven. Daarbij zijn er handige mogelijkheden om data te sorteren en te selecteren. Dit venster is ook de plek waar jouw R scripts verschijnen. Met het knopje links bovenin (het blaadje met het plusje) kun je een nieuw R script toevoegen. **Ik adviseer je sterk om een R script te gebruiken.** Een klein voorbeeld van een script kun je vinden in **Figure 3**. Zo heb je altijd een overzicht van de commando's die in de R-sessie worden uitgevoerd en belangrijker nog, je kunt het opslaan als bestand en is daarmee een eindproduct van de R-sessie.

In een script kun je de commando's intypen en laten uitvoeren in de console. Dit doe je door de cursor aan het begin van de regel van het command te plaatsen en **Ctrl + Enter** of **Cmd + Enter** (Mac) in te gebruiken. Je kunt ook de hele regel selecteren en **Ctrl + Enter** of **Cmd + Enter** gebruiken om een commando uit te voeren.

Rechts bovenin heb je het overzicht van de objecten in de huidige R-sessie. Dit zijn bijvoorbeeld variabelen met de toegewezen waarden, functies en datasets. Daarbij heb je een tab met *history*, hiermee krijg je een handig overzicht van de commando's die je in deze R-sessie hebt uitgevoerd.

Het venster rechts onderin bevat de meeste tabs van alle vensters. De *Files* tab is de verkenner waarmee je door de bestanden en mappen kunt bladeren. Het *Plots* tab is de plek waar de grafieken en andere visualisaties

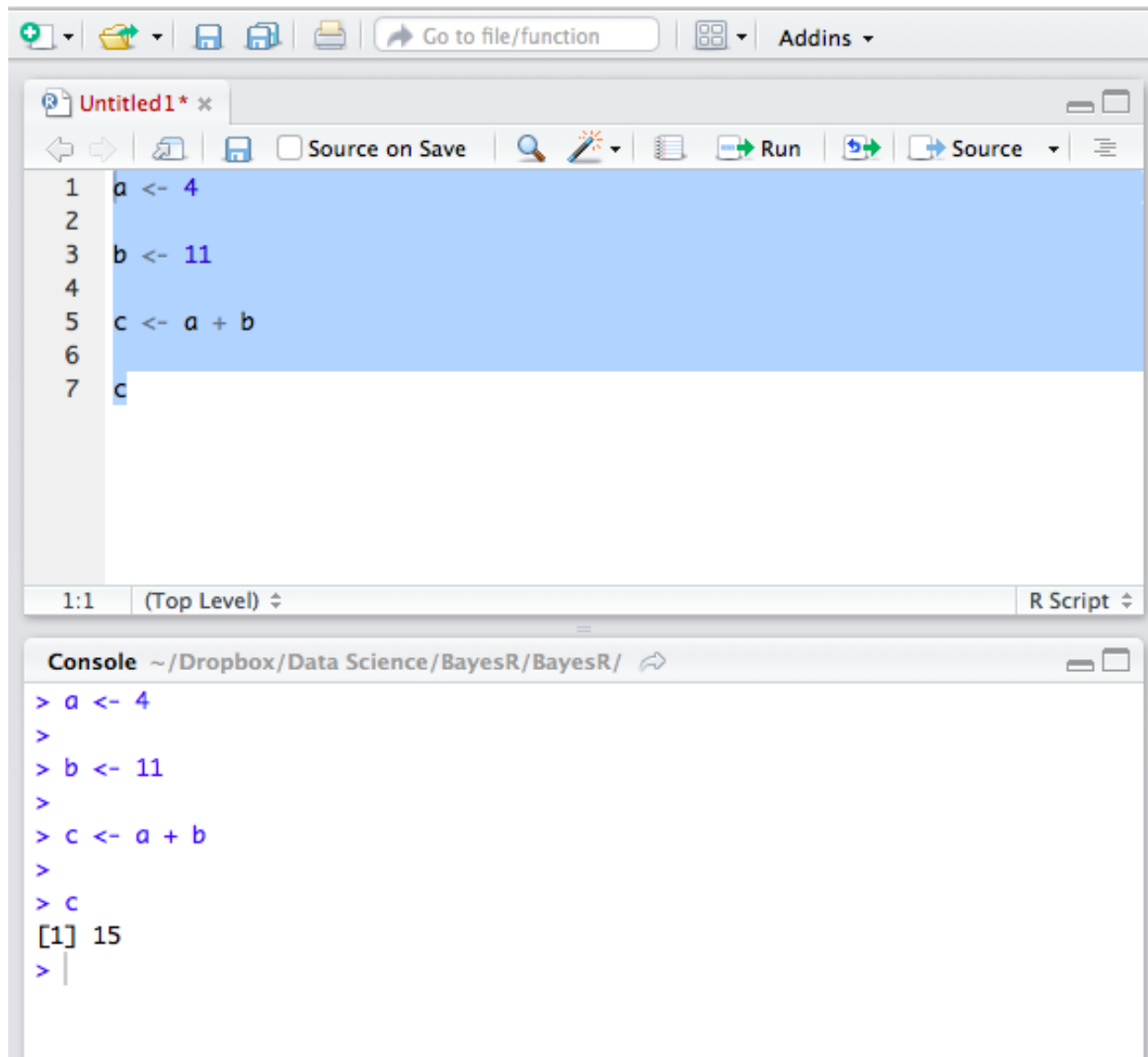


Figure 3: Script in R

worden weergegeven. Dit tab opent zich automatisch zodra je in R een commando invoert om een visualisatie weer te geven. Het *packages* tab geeft een overzicht van de gedownloadte en geopende packages in de huidige omgeving (jouw account op de laptop of computer). Packages zijn modules gemaakt door ontwikkelaars om extra functies toe te voegen in R, deze zullen we ook tegenkomen in dit boek. Het *Viewer* venster is iets nieuwer dan het *Plots* venster en neemt de interactieve visualisaties zoals kaarten voor zijn rekening.

Deze vensters in RStudio geven samen een uitstekende ervaring voor het data analyseren met R. Ongetwijfeld zal je dit zelf ervaren. De zojuist behandelde onderdelen zijn slechts het topje van de ijsberg qua features in RStudio. Ongetwijfeld zal je zelf nog veel meer handige functies bevatten. Voor het verdere verloop van dit boek zijn de belangrijkste in ieder geval behandeld.

2.14 Datasets

Bij sommige hoofdstukken in dit boek wordt bij de voorbeelden gebruik gemaakt van databestanden. De meeste databestanden zijn al in R beschikbaar, in dit boek wordt daarbij uitgelegd hoe je deze gebruikt. Sommige databestanden worden in R geïmporteerd. Dit zijn de volgende databestanden welke op mijn blog kunt downloaden:

Databestanden:

- **Consumentenprijzen.csv**
- **Projecten.csv**
- **Projecten2.csv**

Voor een overzicht van alle datasets (ook van de oude tutorial) kun je naar de pagina van mijn oude R cursus gaan op <http://r-handleiding.blogspot.nl/p/voorbeeld-databestanden-downloaden.html>.

2.15 Referenties

Op sommige plekken in dit boek wordt er doorverwezen naar een bron. Dit zijn bronnen waar een bepaald onderwerp in dit boek extra wordt besproken. Het is nuttig om deze bronnen na te gaan als je, naast de rode lijn in dit boek, dieper wilt ingaan over een bepaald onderwerp. De bronnen zijn genummerd. Voor de lezers die de PDF-versie van dit boek hebben, kunnen ze eenvoudig klikken op de titel om naar de oorspronkelijke bron te gaan. Lezers van de hardcopy versie van dit boek kunnen door middel van de nummering van bronnen het betreffende nummer opzoeken in het hoofdstuk **Referenties** dat zich aan het einde van dit boek bevindt. Achter de nummers van de bronnen staat informatie om de bron te raadplegen.

3 De basics van de R-syntax

In dit hoofdstuk beginnen we bij de basics van R. Namelijk het aanmaken van objecten en hier waarden aan toewijzen. Je zult zien dat R voornamelijk uit vectoren staat die een enkele of meerdere waarden bevatten. Dit kunnen getallen of andere datatypen zijn. Op deze vectoren worden toepassingen gedaan, bijvoorbeeld berekeningen of functies. Eigenlijk is dat grofweg wat je continu doet in R, waarden toewijzen aan vectoren en bewerkingen toepassen met bijvoorbeeld functies. Veel succes met het schrijven van je eerste stukjes R code!



Figure 4:

3.1 Variabelen aanmaken in R

R werkt over het algemeen volgens dezelfde principes als andere programmeertalen. Programmeren in R is vooral (1) variabelen aanmaken en definiëren en (2) functies toepassen op deze variabelen. Als we bijvoorbeeld een variabele willen aanmaken die de naam "Peter" bevat, doen we dat als volgt:

```
# Aanmaken van variabelen doe je met <- of met =
naam = "Peter"

# of
naam <- "Peter"
```

Je kunt = of <- gebruiken om waarden toe te wijzen in R. Deze functies doen exact hetzelfde, kies daarom wat jouw voorkeur heeft. Over het algemeen wordt er vaak voor <- gekozen in R. Veel mensen hebben de vraag waarom deze twee varianten bestaan als ze toch hetzelfde doen. Een artikel op Revolution Analytics heeft hier een mooie en korte uiteenzetting voor [1] Use = or <- for assignment?, (Revolution Analytics, 2008)

Als we een variabele `leeftijd` willen aanmaken en daar het getal `34` aan willen toewijzen, gaat dit op dezelfde manier. Nadat de variabele is aangemaakt, kunnen we de waarde die eraan is toegewezen terugvinden door simpelweg de variabele `leeftijd` weer in te voeren. Het is namelijk één van de principes van programmeertalen om een toegewezen waarde te onthouden. Zo weet R in deze sessie dat aan de variabele `leeftijd` de numerieke waarden `34` is toegewezen.

```
leeftijd <- 34

#Geeft:
leeftijd
```

```
## [1] 34
```

Hetzelfde geldt voor het toewijzen van een waarde met een ander datatype, bijvoorbeeld een woord. We maken de variabele `woonplaats` aan die de waarde "Almere" bevat.

```
#of
woonplaats <- "Almere"

#Geeft:
woonplaats
```

```
## [1] "Almere"
```

Alle waarden in R zijn vectoren Een vector is een object dat informatie, en in het geval van R, gegevens bevat. De output de vorige voorbeelden geven getallen vóór de waarden aan ([1]). Dit geeft aan dat het de eerste waarde is uit de vector. Zo staat "Sassenheim" op plaats 1 in de vector `woonplaats`. Je kunt dus stellen dat alle waarden in R vectoren zijn, zelfs als er slechts een waarde is toegewezen.

3.1.1 vectoren met meerdere waarden in R

Een vector kan ook meerdere waarden bevatten. Om meerdere waarden te wijzen aan een vector, gebruik je de `c()` functie. Dit kun je makkelijk onthouden door bijvoorbeeld aan de *concatenate* functie te denken van **Excel**.

We kunnen bijvoorbeeld een vector maken die drie waarden bevat:

```
# een vector 'woonplaatsen' aanmaken die 3 waarden bevat

woonplaatsen <- c("Almere", "Katwijk", "Utrecht")

# de waarden van de 'woonplaatsen' vector bekijken

woonplaatsen
```

```
## [1] "Almere" "Katwijk" "Utrecht"
```

Zoals je ziet, heeft de vector `woonplaatsen` 3 waarden opgeslagen. Raak niet in verwarring door de `[1]` die wordt aangegeven. Dit betekent niet dat de eerste waarde uit de `woonplaatsen` vector "Almere", "Katwijk", "Utrecht" is, maar dat de eerste waarde op plaats 1 staat. Per 10 waarden in een vector wordt namelijk een nummering (indexing) aangegeven. Dit kun je zien als we bijvoorbeeld 12 waarden toewijzen aan een vector.

```
woonplaatsen <- c("Sassenheim", "Katwijk", "Enschede", "Groningen",
                  "Maastricht", "Rotterdam", "Amsterdam", "Leeuwarden", "Assen",
                  "Almere", "Utrecht", "Den Haag")

woonplaatsen
```

```
## [1] "Sassenheim" "Katwijk"    "Enschede"   "Groningen"  "Maastricht"
## [6] "Rotterdam"   "Amsterdam"  "Leeuwarden" "Assen"      "Almere"
## [11] "Utrecht"    "Den Haag"
```

Op deze manier kun je bijvoorbeeld makkelijk aflezen dat "Den Haag" op plaats 12 staat in de `woonplaatsen` vector. Deze indexering kun je ook handig als je de waarde die op een bepaalde plek staat in een vector wilt ophalen. Als we bijvoorbeeld de waarde van de `woonplaats` willen weten die op plaats 7 staat, doen we dat als volgt:

```
# de waarde ophalen die op plaats 7 staat van de 'woonplaats' vector

woonplaatsen[7]
```

```
## [1] "Amsterdam"
```

```
# de woonplaatsen ophalen die op de plaatsen 3 en 8 staan in de 'woonplaats' vector

woonplaatsen[3:8]
```

```
## [1] "Enschede" "Groningen" "Maastricht" "Rotterdam" "Amsterdam"
## [6] "Leeuwarden"
```

Let op dat de indexering van de selectie die je gemaakt hebt nu weer anders is. In dit geval staat "Amsterdam" namelijk op plaats 5.

De index van R is 1. Dit betekent dat reeksen van vectoren of andere onderdelen in R bij 1 beginnen. Dit klinkt logisch, maar het is bij programmeren meer gebruikelijk dat reeksen bij 0 beginnen. Dit worden talen genoemd met een zero based indexing. Bij onze woonplaatsen vector in het voorbeeld zou in de programmeertaal Python het eerste element "Sassenheim" op plaats 0 staan, terwijl het bij R gewoon op plaats 1 staat.

3.1.2 Vectoren samenvoegen

Je kunt ook waarden uit andere vectoren samenvoegen. Hiervoor voer je in plaats van de absolute waarden, de namen van de vectoren in. Dit wordt in het volgende voorbeeld gedemonstreerd.

```
# de waarde "Nina" toewijzen aan de 'naam' vector
naam <- "Nina"

# de waarde 23 toewijzen aan de 'leeftijd' vector
leeftijd <- 23

# de waarde "Amsterdam" toewijzen aan de 'woonplaats' vector
woonplaats <- "Amsterdam"

# een FALSE waarde toewijzen aan de 'rookt' vector
rookt <- FALSE

# de waarden van alle vectoren bekijken
naam
```

```
## [1] "Nina"
```

```
leeftijd
```

```
## [1] 23
```

```
woonplaats
```

```
## [1] "Amsterdam"
```

```
# de vector 'persoon' aanmaken en de waarden van de andere vectoren hier aan toewijzen
persoon <- c(naam, leeftijd, woonplaats, rookt)

persoon
```

```
## [1] "Nina"      "23"        "Amsterdam" "FALSE"
```

Op deze manier wijs je 4 waarden toe aan de `persoon` vector.

3.2 Verschillende klassen in R

Waarden en variabelen hebben verschillende klassen en datatypen. Voorbeelden zijn getallen (**numeric/integer**), woorden (**character/string**), boolean (**TRUE** of **FALSE**). Ook zijn er types zoals matrixen (**matrix**), data frames (**data frame**), lijsten (**list**). Deze types zijn verzamelingen van data.

Ook onze zojuist aangemaakte variabelen hebben een klasse. R heeft deze automatisch toegewezen. Met de `class()` functie kunnen we er achter komen welk datatype een variabele is.

```
naam <- "Geertje"

class(naam)
```

```
## [1] "character"
```

```
class(leeftijd)
```

```
## [1] "numeric"
```

De klasse van de variabele `naam` is een **character** en `leeftijd` is **numeric**. Hierbij kun je het **character** datatype vergelijken met het **string** datatype die je in andere programmeertalen tegenkomt. Het **numeric** datatype in R kun je het best vergelijken met het **float** datatype die je in andere programmeertalen tegenkomt.

3.2.1 Automatische definitie van klassen in R

Zoals je misschien hebt gemerkt, hoef je in R niet aan te geven welke klasse een variabele moet krijgen. Dit doet R automatisch. Bij een numerieke waarde weet R namelijk dat het om een **numeric** klasse gaat, net als voor bijvoorbeeld een woord of een naam automatisch het **character** datatype wordt toegewezen.

Bij andere programmeertalen zoals Java en C# is het nodig om aan te geven welk datatype de nieuwe variabele moet hebben, bijvoorbeeld:

```
C#: static string naam = "Arie";
R: naam <- "Arie" of naam = "Arie".
```

3.2.2 datatypes van variabelen in R veranderen

Het is handig dat R automatisch de klasse definieert. Echter ben je soms niet tevreden met het datatype dat R automatisch toewijst. Daarom kun je in R ook zeer eenvoudig het datatype veranderen. Als wij bijvoorbeeld de variabele `leeftijd` willen veranderen van **numeric** naar **character**, gebruiken wij de `as.character`-functie.

```
# de variabele "leeftijd" definiëren

leeftijd <- 26
class(leeftijd)
```

```
## [1] "numeric"
```

```
#een variabele "leeftijd2" aanmaken die met "string" als datatype
leeftijd2 <- as.character(leeftijd)
leeftijd2
```

```
## [1] "26"
```

```
class(leeftijd2)
```

```
## [1] "character"
```

Echter kunnen niet alle variabelen naar de gewenste datatype geconverteerd worden. Zo kun je bijvoorbeeld een woord niet omzetten naar een numerieke waarde.

```
naam <- "David"

# een nieuwe variabele "naam2" aanmaken met een naam geconverteerd naar numeriek
naam2 <- as.numeric(naam)
```

```
## Warning: NAs introduced by coercion
```

```
class(naam2)
```

```
## [1] "numeric"
```

```
naam2
```

```
## [1] NA
```

Zoals je kunt zien, krijgen we een waarschuwing (**Warning message:**) die aangeeft dat de waarde wegvalt door het omzetten van het datatype.

NA's Wat de waarschuwing letterlijk zegt is: “Er zijn NA’s ontstaan, dat staat voor Not Available (lege en onbruikbare waarden), omdat we een character waarde ‘dwingen’ (coerce) naar een ander datatype”. De kans is groot dat je dit later tijdens het analyseren in R tegenkomt. Het is namelijk heel makkelijk om over het hoofd te zien dat we een datatype op een verkeerde manier converteren.

3.2.3 Character en Numeric vectoren

Als we een vector `leeftijd` willen aanmaken en daar het getal 26 willen toewijzen, gaat dit op dezelfde manier. Nadat de variabele is aangemaakt, kunnen we de waarde die eraan hangt terugvinden door simpelweg de variabele `leeftijd` weer in te voeren. Het is namelijk een van de principes van programmeertalen om een toegewezen waarde te onthouden. Zo weet R in deze sessie dat aan de variabele `leeftijd` de numerieke waarde 26 is toegewezen. Belangrijk om te weten is dat R vectoren kan overschrijven. Als we bijvoorbeeld eerst een vector genaamd `leeftijd` aanmaken met de waarde 26 en later een vector met dezelfde naam aanmaken met de waarde 44, wordt alleen 44 onthouden. Hierbij wordt de waarde 26 vergeten en kun je de toewijzing ook nooit meer terugvinden, je kunt het alleen maar opnieuw aanmaken.

```
leeftijd <- 26
```

```
leeftijd
```

```
## [1] 26
```

```
leeftijd <- 44
```

```
leeftijd
```

```
## [1] 44
```

```
#of
```

```
woonplaats <- "Lisse"
```

```
# geeft:
```

```
woonplaats
```

```
## [1] "Lisse"
```

```
# nieuw
```

```
woonplaats <- "Oldenzaal"
```

```
# geeft
```

```
woonplaats
```

```
## [1] "Oldenzaal"
```

```
# voor de duidelijkheid:
```

```
woonplaats <- "Amsterdam"
```

```
woonplaats <- "Den Haag"
```

```
woonplaats
```

```
## [1] "Den Haag"
```

Je kunt zien dat 26 en 44 niet tussen aanhalingstekens (") zijn aangegeven en "Lisse" wel. Dit komt omdat waarden verschillende **types** hebben. Het getal 26 is numeriek en heeft daarom de klasse **numeric**. Lisse is een woord omdat het letters bevat, dit type waarden wordt in programmeertalen een **string** genoemd. Deze verschillende types worden in R een **class** genoemd. De belangrijkste en meest gebruikte **classes** in R zijn:

Class/datatype:		
	Eenvoudige beschrijving:	Voorbeeld:
numeric	Numerieke waarden, ook met decimalen.	Huur = 654.67
character	Tekst, alfanumerieke waarden.	Land = Spanje
logical	Operatoren zoals TRUE of FALSE waarden.	2 > 3 = FALSE
factor	Character waarden met een volgorde/hierarchie.	hoog, gemiddeld, laag

3.3 Logical waarden en operators

Logical waarden moeten **niet** tussen aanhalingstekens (") worden geplaatst. Boolean waarden en logical operators zijn erg handig bij het maken van berekeningen en verwerken van data. Bijvoorbeeld bij het aanmaken van waarden, het maken van functies en het filteren van data. De volgende voorbeelden helpen je een idee te krijgen van logical operators.

```
# de boolean waarde FALSE toewijzen aan de 'roken' vector  
roken <- FALSE  
roken
```

```
## [1] FALSE
```

```
# het datatype van de 'roken' vector weergeven  
class(roken)
```

```
## [1] "logical"
```

```
# evalueren of 4 groter dan 5 is  
4 > 5
```

```
## [1] FALSE
```

```
# evalueren of 4 + 4 groter dan 7 is  
4 + 4 > 7
```

```
## [1] TRUE
```

```
# evalueren of 4 + 4 en 3 + 5 allebei gelijk aan 8 zijn  
4 + 4 & 3 + 5 == 8
```

```
## [1] TRUE
```

```
# evalueren of 4 + 4 of 2 + 5 gelijk zijn aan 8  
4 + 4 | 2 + 5 == 8
```

```
## [1] TRUE
```

```
# evalueren of FALSE, FALSE of TRUE is  
FALSE == FALSE | TRUE
```

```
## [1] TRUE
```

*Een artikel op R-bloggers geeft een paar handige rekenvoorbeelden met logical operators: [2]
Logical operators in R, (*R-bloggers*, 2010)

Niveaus in een factor *De niveaus van een factor vector kun je ook handmatig aanpassen. In de volgende hoofdstukken wordt er uitgebreid uitgelegd hoe je in R moet omgaan met de **factor** klasse. Met hoofdstuk 8 “Het factor datatype” in dit boek wordt er een heel hoofdstuk toegewijd aan dit datatype.*

3.4 Samenvatting

3.4.1 Programmeren in R

Programmeren in R bestaat voornamelijk uit twee handelingen: variabelen aanmaken en functies toepassen. Hierbij kunnen variabelen verschillende soorten objecten zijn zoals waarden en tabellen. Functies pas je toe om data te analyseren, te bewerken of verwerken. Deze functies zal je later in dit boek tegenkomen.

3.4.2 datatypes in R

Het is belangrijk dat je bij R rekening houdt met de verschillende datatypes die er zijn: **Numeric**, **integer**, **character**, **factor** en **logical** (boolean) zijn de datatypes die je het meest zal tegenkomen in R. Daarnaast zijn er ook datatypes zoals **Date** (datum) en **complex**, deze kun je voor nu nog met rust laten. De datatypes bepalen welke functies er op de variabelen toegepast kunnen worden. Zo kun je bijvoorbeeld geen rekenfuncties toepassen op **character** datatypes.

3.4.3 vectoren in R

Aan een variabele kun je meerdere waarden toekennen. Dit doe je met de **c()** functie. Hier maak je vectoren mee aan. Een vector heeft ook een datatype. Een vector met alleen getallen is bijvoorbeeld een **numeric** vector. Je kunt handig gebruik maken van vectoren door calculaties of andere functies toe te passen op meerdere waarden tegelijk. Als er echter ook maar een enkele **character** waarde in de vector met numerieke waarden staat, wordt de hele vector gelijk omgezet naar een **character** vector. Je kunt dan ook geen functies meer toepassen zoals je die bij een numeric vector kunt doen, zoals rekenfuncties.

3.5 Oefeningen

3.5.0.0.1 Maak verschillende variabelen/vectoren aan met jouw persoonlijke informatie zoals naam, leeftijd, woonplaats en opleiding

3.5.0.0.2 Voeg de vectoren samen in een enkele vector die deze informatie bevat

3.5.0.0.3 Selecteer de woonplaats uit de zojuist aangemaakte vector

4 Reken- en statistische functies in R

R is een statistische programmeertaal en bevat enorm veel statistische functies. Dit is een belangrijke reden waarom R een van de meest gebruikte programmeertalen is die gebruikt wordt om data uitgebreid en wetenschappelijk te analyseren. Echter bevat R ook simpele reken- en wiskunde functies zoals optellen, vermenigvuldigen, en worteltrekken. Een greep uit deze functies wordt in dit hoofdstuk behandeld. De meer uitgebreide statistische functies worden later in dit boek behandeld.

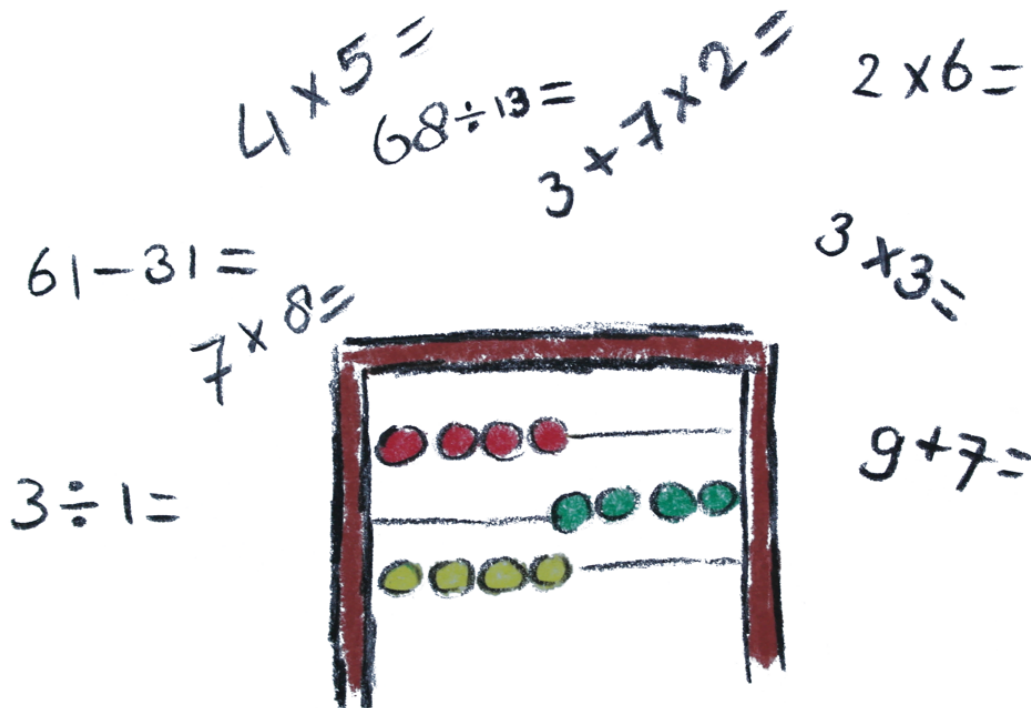


Figure 5:

4.1 Eenvoudige rekensommen

4.1.1 Gebruikelijke eenvoudige rekensommen

Optellen en aftrekken gaat heel eenvoudig met R door middel van de gebruikelijke operators:

Operator	Functie
+	Optellen
-	Aftrekken
/	Delen
*	Vermenigvuldigen
^	Machtsverheffen
sqrt()	Worteltrekken

```
# Optellen  
2 + 2
```

```
## [1] 4
```

```
# Aftrekken  
10 - 3
```

```
## [1] 7
```

```
# Vermenigvuldigen  
5 * 5
```

```
## [1] 25
```

```
# Delen  
36 / 6
```

```
## [1] 6
```

```
# Machtsverheffen  
2 ^ 3
```

```
## [1] 8
```

```
# Worteltrekken  
sqrt(81)
```

```
## [1] 9
```

Ook kun je vectoren gebruiken om een rekensom uit te voeren. De `sum()` functie telt bijvoorbeeld alle waarden in een vector op.

```
# Optellen  
sum(3,5,2,6,6,7,2,11)
```

```
## [1] 42
```

```
# of  
sum(1:8)
```

```
## [1] 36
```

4.1.2 Uitgebreide berekeningen (met rekenregels)

Naast gebruik te maken van eenvoudige rekensommen, kun je zelf ook formules schrijven. Bijvoorbeeld de berekening van de oppervlakte van een cirkel: `straal^2 * pi`.

In R kun je met `pi` zeer eenvoudig de waarde van `pi` oproepen.

```
# De waarde van pi laten zien  
pi
```

```
## [1] 3.141593
```

```
# De waarde van de straal aangeven  
straal <- 4  
  
# Het resultaat van de berekening opslaan in de variabele "oppervlakte_circel"  
oppervlakteCircel <- (straal^2)* pi  
  
# Het resultaat van de berekening laten zien  
oppervlakteCircel
```

```
## [1] 50.26548
```

Hier kun je zien dat het zeer eenvoudig is om speciale berekeningen uit te voeren in R. Het wordt nog interessanter als je gebruik gaat maken van functies. Dit wordt in hoofdstuk 11 “Correlaties, functies en loops in R” van dit boek behandeld.

4.2 Simpele statistische functies gebruiken in R

Een statistische programmeertaal zou natuurlijk geen statistische programmeertaal zijn als het geen statistische functies zou bevatten. Vanwege het eenvoudige gebruik van deze functies in R, is het een van de favoriete hulpmiddelen van o.a. statistici, data analisten en data scientists. Waar je bij andere programma's meer moeite moet doen om bepaalde statistische functies uit te voeren, heeft R deze eenvoudige statistische functies al ingebouwd. De simpele statistische functies worden in de komende voorbeelden gedemonstreerd.

4.2.1 Willekeurige data (normaal verdeeld) genereren met R

Met de `rnorm()` functie kun je getallen genereren die normaal zijn verdeeld. Om dit op een simpele manier te doen, kun je twee argumenten aan de functie geven:

- **n**: Het aantal getallen dat je wilt genereren;
- **m**: De **mean**, het gemiddelde, van de normaal verdeelde getallen vaststellen. Deze staat op 0 als dit argument niet wordt ingevuld.

```
getallen <- rnorm( n = 100, m =30)

# of simpelweg
getallen <- rnorm(1000, 300)
```

In dit geval hebben we een reeks van 1000 getallen gegenereerd met een gemiddelde van 300.

4.2.2 Het gemiddelde, de standaardafwijking en variantie berekenen in R

Met de `mean()`, `sd()`, `var()` functie worden respectievelijk het gemiddelde, de standaard-afwijking en de variantie van de data berekend:

```
# Het gemiddelde uitrekenen
mean(getallen)
```

```
## [1] 299.9972
```

```
# De standaardafwijking uitrekenen
sd(getallen)
```

```
## [1] 1.034659
```

```
# De variantie uitrekenen
var(getallen)
```

```
## [1] 1.07052
```

In hoofdstuk 4 wordt verder ingegaan op de statistische functies in R. Daarbij wordt uitgelegd hoe je de statistische functies handig kunt gebruiken om een dataset te analyseren.

4.2.3 Krachtige berekeningen uitvoeren in R

Door je creativiteit te gebruiken en er handiger in te worden, kan R als een zeer krachtige rekenmachine worden gebruikt waarmee je snel uitgebreide berekeningen kunt uitvoeren. Hier worden een aantal voorbeelden gegeven. Je zult zien dat je heel handig kunt zijn door gebruik te maken van gedefinieerde variabelen en de datastructuren in R.

```
# getallen in een vector

waarden1 <- c(11,12,35,36)
waarden1
```

```
## [1] 11 12 35 36
```

```
# alle leeftijden met 1 verhogen  
waarden1 +1
```

```
## [1] 12 13 36 37
```

```
# vector met nieuwe getallen  
waarden2 <- c(2,3,1,5)  
waarden2
```

```
## [1] 2 3 1 5
```

```
# getallen uit waarden1 en waarden2 met elkaar vermenigvuldigen  
waarden1 * waarden2
```

```
## [1] 22 36 35 180
```

```
# stralen cirkels  
stralen_cirkels <- c(3,5,7,12)  
oppervlakten_cirkels <- stralen_cirkels^2 * pi  
oppervlakten_cirkels
```

```
## [1] 28.27433 78.53982 153.93804 452.38934
```

4.2.4 Handige berekeningen in R: Een voorbeeldsituatie

Na een beetje handigheid, kun je ook voor berekeningen in kleine situaties handig gebruik maken van R. Neem bijvoorbeeld de volgende situatie:

Er zijn 30 kinderen in een klas. Voor deze kinderen moeten er stoelen worden besteld die zo goed mogelijk bij de lengtes van deze kinderen passen. Er zijn 2 maten mogelijk en worden als volgt verdeeld:

- Tussen de 1 en 1.20 meter krijgt stoel A, die 35 euro per stuk kosten;
- Groter dan 1.20 meter krijgt stoel B, die 45 euro per stuk kosten.

Vraag: Op basis van de lengtes van de kinderen. Wat zijn de totale kosten van de stoelen die voor de kinderen besteld moeten worden?

Hier wordt de `runif()` functie gebruikt om 30 willekeurige getallen te genereren die variëren van 1 tot 1.40. In een echte situatie hoef je natuurlijk geen willekeurige getallen te genereren

```
# we gebruiken de runif functie om de lengtes van de kinderen te genereren.  
  
lengtes_kinderen <- runif(30, min = 1, max = 1.4)  
lengtes_kinderen
```

```
## [1] 1.355546 1.116390 1.208466 1.088046 1.206857 1.116748 1.168355
## [8] 1.249527 1.243829 1.298961 1.087983 1.058224 1.297046 1.166659
## [15] 1.366784 1.029067 1.227664 1.220640 1.317800 1.382838 1.306439
## [22] 1.351855 1.019604 1.281678 1.244090 1.397893 1.230010 1.055899
## [29] 1.310037 1.207196
```

```
# De rekensom

# tel alle waarden op uit de array "lengtes_kinderen"
# die kleiner of gelijk zijn dan 1.20. Vermenigvuldig dit aantal met 35

# kosten voor de groep kinderen die kleiner dan of 1.20 meter zijn
kosten_groepA <- sum(lengtes_kinderen <= 1.20) * 35

# kosten voor de groep kinderen die groter dan 1.20 meter zijn
kosten_groepB <- sum(lengtes_kinderen > 1.20) * 45

#De totale kosten van de klas optellen
totale_kosten <- kosten_groepA + kosten_groepB

totale_kosten
```

```
## [1] 1250
```

```
# Overzicht
aantal_groepA <- sum(lengtes_kinderen <= 1.20)
aantal_groepB <- sum(lengtes_kinderen > 1.20)

aantal_groepA
```

```
## [1] 10
```

```
aantal_groepB
```

```
## [1] 20
```

```
# Checken of de berekening klopt:
(15 * 35) + (15 * 45) == totale_kosten
```

```
## [1] FALSE
```

```
# De berekening klopt.
```

Door gebruik te maken van verschillende functies, kun je voor verschillende situaties vrij handig en snel de uitkomst uitrekenen. Maak je geen zorgen als dit nog moeilijk overkomt. Je raakt vrij snel handig genoeg met R om dit soort situaties zelf op soortgelijke manieren op te lossen.

4.3 Samenvatting

4.3.1 Hulpmiddel voor eenvoudige rekensommen

Eenvoudige rekensommen zijn een fluitje van een cent voor R. Omdat de algemene rekenregels worden toegepast, is het mogelijk om een rekensom volledig in R uit te schrijven om achter het resultaat te komen.

```
# een volledig uitgeschreven rekensom uitrekenen
4 * (9 ^ 2 + (3 + 12 / 6) ) + 11 - 3
```

```
## [1] 352
```

Naast de waarde direct in te voeren kunnen er natuurlijk ook berekeningen worden uitgevoerd die in vectoren zijn opgeslagen. In principe doe je met programmeren niet anders.

4.3.2 Data genereren

In sommige gevallen heb je proef data (sample data) nodig om te experimenteren. Hier zijn verschillende functies voor in R. In het voorbeeld van dit hoofdstuk is de `rnorm()` functie gebruikt. Dit zou vooral gebruikt worden door statistici omdat dit normaal verdeelde data genereert. De `runif` functie die we gebruikt hebben heeft niets te maken met een normale verdeling en genereert gewoon data. Naast deze twee populaire manieren zijn er nog veel meer mogelijkheden om data te genereren in R.

Als je meer wilt weten over de `runif()` functie in R om data te genereren, kun je het `?runif()` command invoeren om de instructie van de functie te bekijken. Dit kun je voor iedere functie doen.

4.3.3 Berekeningen uitvoeren

Door data op te slaan in vectoren, kun je zeer krachtige berekeningen maken. Als je daarbij ook nog eens gebruikt maakt van zelfgeschreven formules die je opslaat in functies, heb je een heel krachtig rekenmiddel tot je beschikking. Vector en matrix-calculaties zijn in sommige bedrijven de vaste kost, R leent zich hier uitstekend voor.

```
aantallen <- c(23, 45, 63, 78, 12, 100)
prijzen <- c(120, 356, 183, 59, 12, 31)
aantallen * prijzen
```

```
## [1] 2760 16020 11529 4602 144 3100
```

```
sum(aantallen * prijzen)
```

```
## [1] 38155
```

4.4 Oefeningen

4.4.0.1 Bereken de oppervlakte van een cirkel die een doorsnede heeft van 11 meter

4.4.0.0.2 Gebruik de stelling van Pythagoras om de C van een driehoek te berekenen waarvan A 11cm is en B 4cm

4.4.0.0.3 Reken uit hoeveel geld je op je spaarrekening hebt na 25 jaar als je 1000 vastzet voor 2%, EN de spaarrekening 2,70 euro per jaar kost

5 Data analyseren en bewerken in R

Data analyseren en data bewerken zijn handelingen die hand in hand samengaan. Je analyseert een dataset en komt erachter dat deze nog niet helemaal geschikt is voor de analyses die je wilt toepassen, dus doe je een bewerking op de data. Deze bewerkingen verschillen van het maken van extra kolommen tot het veranderen van waarden in een vector. Deze werkzaamheden is wat je het grootste deel van de tijd doet in een project met R, hier komt een groot deel van de waardevolle inzichten uit. Succes!

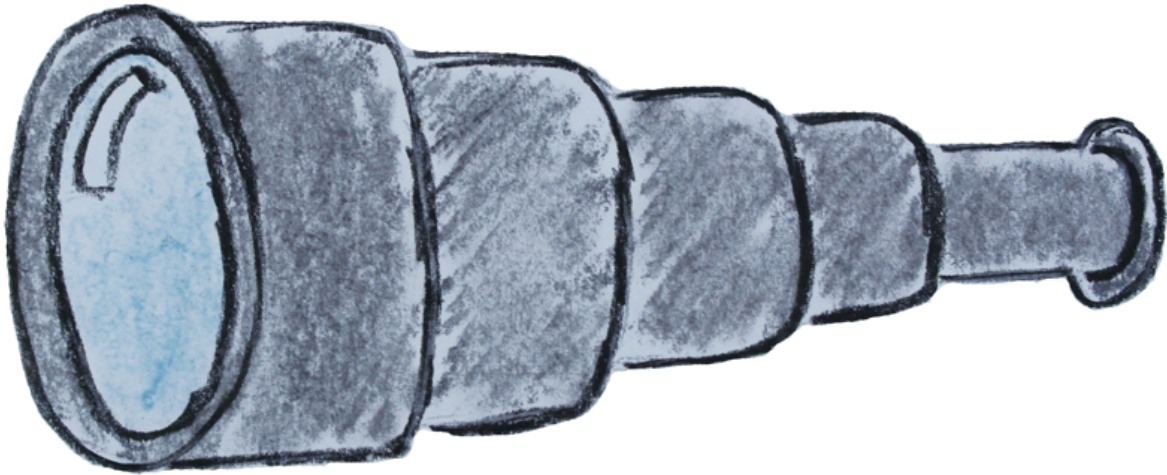


Figure 6:

5.1 Ingebouwde dataset laden in R

R bevat ingebouwde datasets die je kunt gebruiken om mee te experimenteren. De beschikbare datasets kun je weergeven met de `data()` functie.

```
# voor een overzicht van de datasets, gebruik je het "data()" command

data()
```

*Sommige packages bevatten ook datasets, zoals het **kernelab** package of het **ggplot2** package. Deze roep je ook gewoon op met de `data()` functie.*

Voor de volgende voorbeelden wordt de ingebouwde dataset `mtcars` gebruikt. Dit is een dataset die voor een aantal auto's een verschillende eigenschappen weergeeft. De volgende commands laten zien hoe je de dataset kunt inladen en een aantal basis-eigenschappen van de data kunt weergeven. Om de dataset in de R-sessie te laden, gebruik je `data(<naam van de dataset>)` zoals in het volgende voorbeeld:

```
# de mtcars dataset inladen

data(mtcars)
```

5.2 Eenvoudige manieren om de dataset te verkennen

Voordat je aan de slag gaat met een dataset, is het handig om een goed beeld te krijgen van de dataset. Zo voorkom je om dingen te willen doen met de dataset die helemaal niet kunnen. Er is geen standaard aanpak om een dataset te verkennen, iedereen heeft zijn eigen manier om dit te doen. Echter biedt R wel handige functies en methoden om de dataset te verkennen. We zullen er een paar bespreken.

5.2.1 De structuur van de dataset weergeven: de `str()` functie

De `str()` (structure) functie geeft de structuur van de dataset weer. Het geeft aan uit hoeveel rijen en kolommen de dataset bestaat en welk datatype elke kolom is. Deze kolommen worden variabelen (**variables**) genoemd.

```
# de structuur van de dataset laten zien met de str() functie

str(mtcars)

## 'data.frame':   32 obs. of  11 variables:
## $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num  160 160 108 258 360 ...
## $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num  16.5 17 18.6 19.4 17 ...
## $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
## $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Zoals je in het voorbeeld kunt zien, wordt voor iedere variabele een voorbeeld van de data gegeven. Daarbij staan bovenaan in de output van de `str()` functie de dimensies van de dataset: **32 obs:** 32 observaties/rijen en **11 variables:** 11 variabelen/kolommen.

5.2.2 De dimensies, het aantal rijen en kolommen, laten zien van de dataset

```
dim(mtcars)
```

```
## [1] 32 11
```

5.2.3 Individueel het aantal rijen en aantal kolommen van de dataset laten zien

Om het aantal rijen in de dataset te laten zien, gebruik je de `nrow()` functie:

```
nrow(mtcars)
```

```
## [1] 32
```

Om het aantal kolommen in de dataset te laten zien, gebruik je de `ncol()` functie:

```
ncol(mtcars)
```

```
## [1] 11
```

5.2.4 De kolomnamen van de dataset weergeven

Om de namen van de variabelen van de dataset weer te geven, gebruik je de `names()` functie:

```
names(mtcars)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

Ook kun je de `colnames()` functie gebruiken:

```
colnames(mtcars)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

5.2.5 Eenvoudig een statistische samenvatting geven van de dataset met de `summary()` functie

De `summary()` functie laat heel eenvoudig basale statistische informatie zien over iedere kolom in de dataset.

```
summary(mtcars)
```

```
##      mpg          cyl          disp          hp
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0
## 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##   Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
```



```
## 3rd Qu.:22.80 3rd Qu.:8.000 3rd Qu.:326.0 3rd Qu.:180.0
## Max. :33.90 Max. :8.000 Max. :472.0 Max. :335.0
##      drat      wt      qsec      vs
## Min. :2.760 Min. :1.513 Min. :14.50 Min. :0.0000
## 1st Qu.:3.080 1st Qu.:2.581 1st Qu.:16.89 1st Qu.:0.0000
## Median :3.695 Median :3.325 Median :17.71 Median :0.0000
## Mean :3.597 Mean :3.217 Mean :17.85 Mean :0.4375
## 3rd Qu.:3.920 3rd Qu.:3.610 3rd Qu.:18.90 3rd Qu.:1.0000
## Max. :4.930 Max. :5.424 Max. :22.90 Max. :1.0000
##      am      gear      carb
## Min. :0.0000 Min. :3.000 Min. :1.000
## 1st Qu.:0.0000 1st Qu.:3.000 1st Qu.:2.000
## Median :0.0000 Median :4.000 Median :2.000
## Mean :0.4062 Mean :3.688 Mean :2.812
## 3rd Qu.:1.0000 3rd Qu.:4.000 3rd Qu.:4.000
## Max. :1.0000 Max. :5.000 Max. :8.000
```

Ook kun je de `summary()` functie gebruiken voor een enkele variabele in de dataset:

```
summary(mtcars$hp)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      52.0   96.5   123.0   146.7   180.0   335.0
```

Je kunt zien dat de `$` operator wordt gebruikt om een variabele uit de dataset te selecteren. Later in dit boek wordt deze en andere handelingen met data frames behandeld.

5.3 Eenvoudige statistische calculaties uitvoeren in R

R is een programmeertaal die uit de statistiek-wereld komt. Hierdoor zitten er standaard al zeer veel statistische functies in R. Deze functies zijn zeer eenvoudig te gebruiken met R, terwijl je daar in andere programmeertalen veel meer moeite voor zou moeten doen. In de komende voorbeelden worden er een aantal statistische functies gebruikt op de `mtcars` dataset.

5.3.1 Het gemiddelde berekenen in R met de `mean()` functie:

Met de `mean()` functie kun je het gemiddelde berekenen van een variabele in de dataset. Dit moet natuurlijk wel een numerieke vector zijn zijn:

```
mean(mtcars$hp)
```

```
## [1] 146.6875
```

Soms komt het voor dat een variabele lege waarden bevat. Dit worden in de statistiek “missing values” genoemd en worden in R met `NA` aangegeven (“Not Available”). Dit zorgt er voor dat je geen berekening kunt doen over die variabele. Echter kun je als nog de berekening uitvoeren door de `NA`s in de variabele over te slaan. Dit doe je door het `na.rm` argument toe te voegen aan de `mean()` functie. De functie ziet er dan bijvoorbeeld als volgt uit:

```
# Een variabele/vector aanmaken die een "NA" bevat
cijfers <- c(1,4,1,5,NA,4)

mean(cijfers)
```

```
## [1] NA
```

```
# De mean functie met het "na.rm" argument
mean(cijfers,na.rm=TRUE)
```

```
## [1] 3
```

Zoals je kunt zien, kan de berekening door het `na.rm` argument **wel** uitgevoerd worden. De `NA` wordt hiermee namelijk overgeslagen in de berekening. Hierdoor wordt de rekensom: $(1 + 4 + 1 + 5 + 4) / 5 = 15 / 5 = 3$.

5.3.2 De standaard-afwijking en variantie berekenen met de `sd()` en `var()` functies in R:

Bij statistisch onderzoek zal je te maken krijgen met de *standaard-afwijking* en de *variantie*. Deze kun je respectievelijk berekenen met de `sd()` en de `var()` functie.

```
# de standaard deviatie van de pk's van de auto's berekenen
sd(mtcars$hp)
```

```
## [1] 68.56287
```

```
# de variantie van de pk's van de auto's berekenen
var(mtcars$hp)
```

```
## [1] 4700.867
```

5.3.3 De minimum en maximum bekijken in R met de `min()` en `max()` functies:

Soms wil je de minimum of maximum van een bepaalde variabele of vector weten. Hier kun je eenvoudig respectievelijk de `min()` en `max()` functies voor gebruiken.

```
min(mtcars$hp)
```

```
## [1] 52
```

```
max(mtcars$hp)
```

```
## [1] 335
```

5.3.4 De correlatie berekenen in R

De correlatie geeft de samenhang tussen twee variabelen weer en kan met de `cor()` functie worden berekend. Deze `cor` functie neemt twee vectoren aan als argumenten waar de correlatie over berekend moet worden.

```
cor(mtcars$hp,mtcars$mpg)
```

```
## [1] -0.7761684
```

Een hoge negatieve correlatie, hoe meer paardenkracht (**hp**) des te minder zuinig de auto is (**mpg**, miles per gallon). Dat klinkt voor nu nog wel logisch.

In hoofdstuk 8 wordt het onderwerp correlatie uitgebreid behandeld.

5.3.5 Een lineair voorspellingsmodel maken in R:

Ook een lineair (regressie) model kan in R eenvoudig met de `lm()` functie gemaakt worden. In het volgende voorbeeld wordt een lineair voorspellingsmodel gemaakt waarmee we de zuinigheid (**mpg**) van een auto voorspellen op basis van de paardenkracht (**hp**) van de auto. Maak je geen zorgen als de uitkomst van deze functie totaal onduidelijk is. In **Hoofdstuk 12** van dit boek wordt het onderwerp *lineaire regressie* uitgebreid behandeld en zal je in staat zijn de uitkomst van dit model uitstekend te kunnen lezen.

```
carsModel <- lm(mpg~hp,mtcars)
```

```
# met de summary() functie kunnen we de output van het regressiemodel uitlezen
```

```
summary(carsModel)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

In hoofdstuk 9 wordt het onderwerp lineair regressiemodel uitgebreid behandeld.

5.4 Samenvatting

5.4.1 Data verkennen in R

Er zijn verschillende manieren om data te verkennen in R. Je hebt hierbij de keuze uit veel mogelijkheden en een flink pakket aan functies. Er is geen standaard manier om data te verkennen. Wel zijn er functies en

manieren die over het algemeen veel gebruikt worden. Het verschilt ook per situatie, je zult bijvoorbeeld vooral de `str()` functie gebruiken als de datatypes een cruciale rol spelen in de analyse. Aan de andere kant gebruik je de `summary()` functie als je meer in de hoek van statische analyse aan de slag bent. Natuurlijk kun je ook gewoon beide gebruiken, niemand houdt je daarbij tegen.

5.4.2 Basis statistieken gebruiken om data te verkennen

Door middel van functies om de data te verkennen, zoals het gemiddelde (`mean()`) en het minimum en maximum (`min()` en `max()`). Door deze basisfunctie te gebruiken tijdens het data verkennen, krijg je een goed idee van de dataset. Daarbij zou het ook heel goed kunnen dat je opvallende dingen in de data tegenkomt. Gevoel van de dataset krijgen door even kort de tijd te nemen om deze functie toe te passen, zal je helpen om even warm te draaien voor het analyseren van de dataset. Een manier is om van simpele data verkenning geleidelijk over te gaan naar een uitgebreide analyse.

5.4.3 Uitgebreide statistische functies in R

Een van de sterkste punten van R is dat je heel eenvoudig statistische functies kunt gebruiken. Achter iedere statistische functie zit natuurlijk een formule, bijvoorbeeld alleen al achter het gemiddelde. In andere programmeertalen zou het gemiddelde nog wel meevallen om op te zoeken, maar meer uitgebreide statistische functies zoals een lineair model of standaard deviatie is veel moeilijker. Bij R zitten deze functies ingebakken met o.a. de `mean()`, `sd()`, `var()` en `lm()`.

5.5 Oefeningen

5.5.0.0.1 Geef een statistische samenvatting van de `mpg` variabele uit de `mtcars` dataset

5.5.0.0.2 Maak een lineair model die de zuinigheid (`mpg`) voorspelt op basis van het gewicht (`wt`) van de auto

5.5.0.0.3 Welke variabelen uit de `mtcars` dataset hebben de hoogste correlatie?

6 Data importeren en exporteren in R

Data moet natuurlijk naar de R-sessie worden gehaald om geanalyseerd te worden. Hiervoor biedt R verschillende mogelijkheden. R is erg flexibel als het gaat om data importeren naar de R-sessie, dit geldt ook voor exporteren. Data is in allerlei vormen en maten op verschillende plekken beschikbaar. Dit kan data zijn die op je eigen machine staat, op een website staat opgeslagen of via een API ontsloten moet worden. Dit is uiteindelijk allemaal mogelijk in de R-sessie te importeren. In dit hoofdstuk leer je de basics van datasets importeren in de R-sessie. Veel plezier!



Figure 7:

6.1 Een ingebouwde dataset openen in R

In het vorige hoofdstuk heb je kunnen zien dat R een aantal ingebouwde datasets heeft. De datasets hebben verschillende eigenschappen en verschillen daarom per situatie in hoe toepasbaar deze zijn.

6.1.1 Alle ingebouwde datasets van R weergeven met het `data()` command

Met het `data()` command krijg je in RStudio een overzicht van de data en daarbij een korte beschrijving, zoals je in *Figure 8* kunt zien.

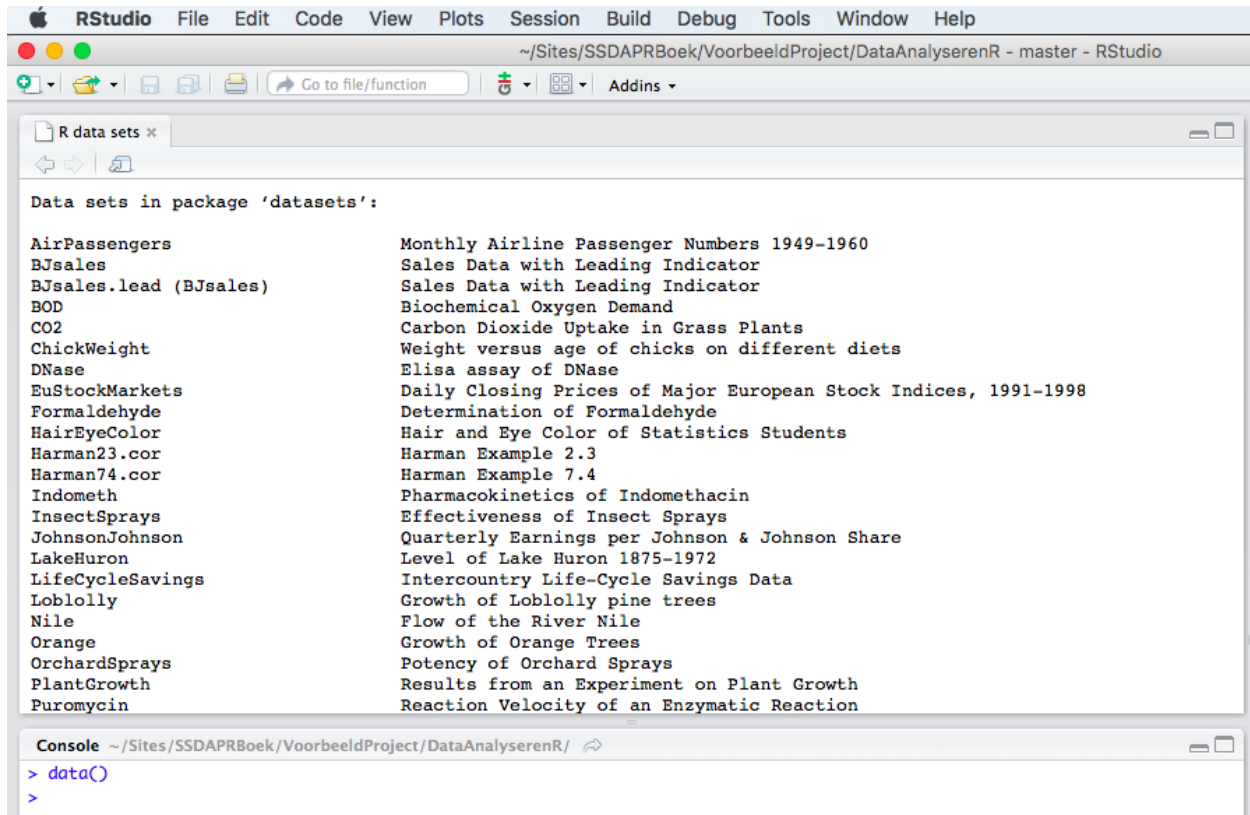


Figure 8:

6.1.2 Een ingebouwde dataset gebruiken

Met hetzelfde `data()` command, kun je een ingebouwde dataset selecteren.

```
# de ingebouwde dataset "mtcars" selecteren
data(mtcars)

# een voorproefje zien van de "mtcars" dataset met de "head()" functie
head(mtcars)
```

```
##           mpg  cyl  disp  hp  drat    wt  qsec vs  am  gear  carb
## Mazda RX4   21.0    6  160 110 3.90 2.620 16.46 0   1    4    4
## Mazda RX4 Wag 21.0    6  160 110 3.90 2.875 17.02 0   1    4    4
```

```
## Datsun 710      22.8  4 108  93 3.85 2.320 18.61  1  1   4   1
## Hornet 4 Drive  21.4  6 258 110 3.08 3.215 19.44  1  0   3   1
## Hornet Sportabout 18.7  8 360 175 3.15 3.440 17.02  0  0   3   2
## Valiant        18.1  6 225 105 2.76 3.460 20.22  1  0   3   1
```

Zoals je kunt zien, is de dataset nu ingeladen.

Het is handig om de dataset te koppelen aan een nieuw object. Zo heb je een eigen versie van de ingebouwde dataset en voorkom je verwarring.

```
# een object aanmaken die de waarden van de ingebouwde dataset bevat
autos_data <- mtcars

head(autos_data)
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160 110  3.90 2.620 16.46  0  1   4    4
## Mazda RX4 Wag  21.0   6  160 110  3.90 2.875 17.02  0  1   4    4
## Datsun 710     22.8   4  108  93  3.85 2.320 18.61  1  1   4    1
## Hornet 4 Drive  21.4   6  258 110  3.08 3.215 19.44  1  0   3    1
## Hornet Sportabout 18.7   8  360 175  3.15 3.440 17.02  0  0   3    2
## Valiant        18.1   6  225 105  2.76 3.460 20.22  1  0   3    1
```

Zoals je kunt zien, bevat het object `autos_data` nu dezelfde data als `mtcars`. Het is nu beter om verder te gaan met `autos_data`, maar dit is slechts een advies.

6.2 Een lokale dataset importeren in R

Een dataset importeren die op jouw eigen machine staat opgeslagen, is ook zeer eenvoudig in R. Dit kan met de commands `read.table()` en `read.csv()`. Aan de hand van argumenten kunnen we specificeren welk bestand en op welke manier we deze willen importeren. Aangezien we werken in RStudio, wordt het importeren van datasets vanuit de **Import Dataset** functie van RStudio uitgelegd. Het command wordt op basis van deze functionaliteit gegenereerd en kunnen we vanaf dat moment beter bekijken.

6.2.1 Gebruik maken van de **Import Dataset** functionaliteit in RStudio voor lokale databestanden

In de RStudio IDE kun je onder de *Environment*-tab de knop **Import Dataset** vinden. We willen een dataset importeren die zich op onze eigen machine (laptop of PC) staat opgeslagen, daarom kiezen we voor *From Local File*.

*Een handige pagina waarin een handig overzicht wordt gegeven over de mogelijkheden om databestanden te importeren in R is de **R Tutorial** [3] pagina “Data Import van Chi Yau, (2016)”. Hierin wordt o.a. ook aangegeven dat je voor sommige formaten, bijvoorbeeld voor Microsoft Excel spreadsheets, een extra package moet installeren.*

6.2.2 Specificeren hoe je de lokale dataset wilt importeren in de **Import Dataset** functie

Zodra je op **Import Dataset** heb geklikt, komt er venster tevoorschijn waarmee je moet navigeren naar de dataset die je wilt importeren. Zodra je de dataset heb geselecteerd, krijg je het venster “Import Dataset” te zien. Hiermee kun je precies specificeren op welke manier je de dataset wilt importeren. Elk onderdeel wordt hieronder kort beschreven.

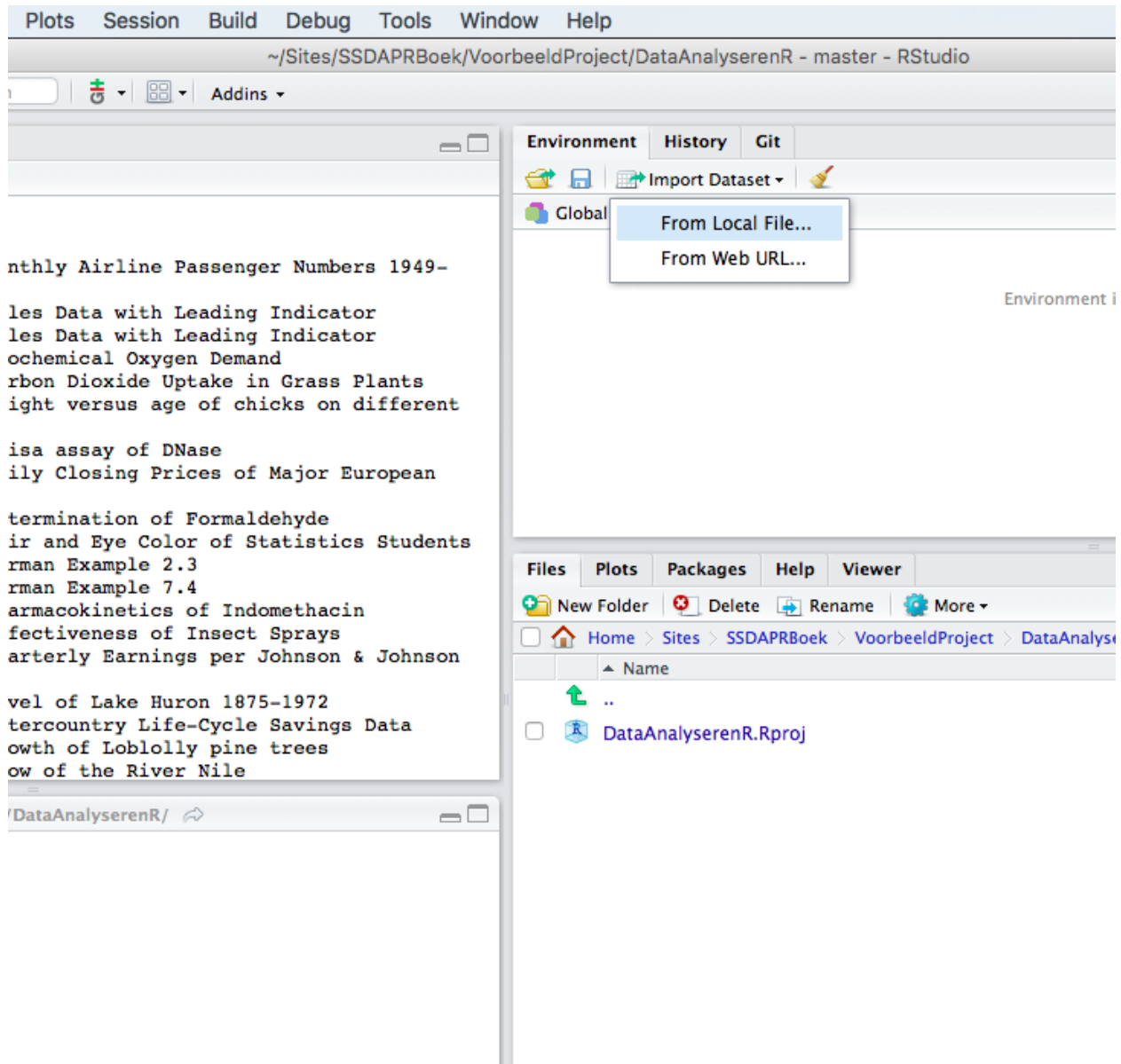


Figure 9: From Local File

RStudio heeft de goede eigenschap door van te voren al op de juiste manier te specificeren hoe de dataset geïmporteerd moet worden. Is dit niet ingevuld zoals je dat wilt, kun je het in dit venster aanpassen.

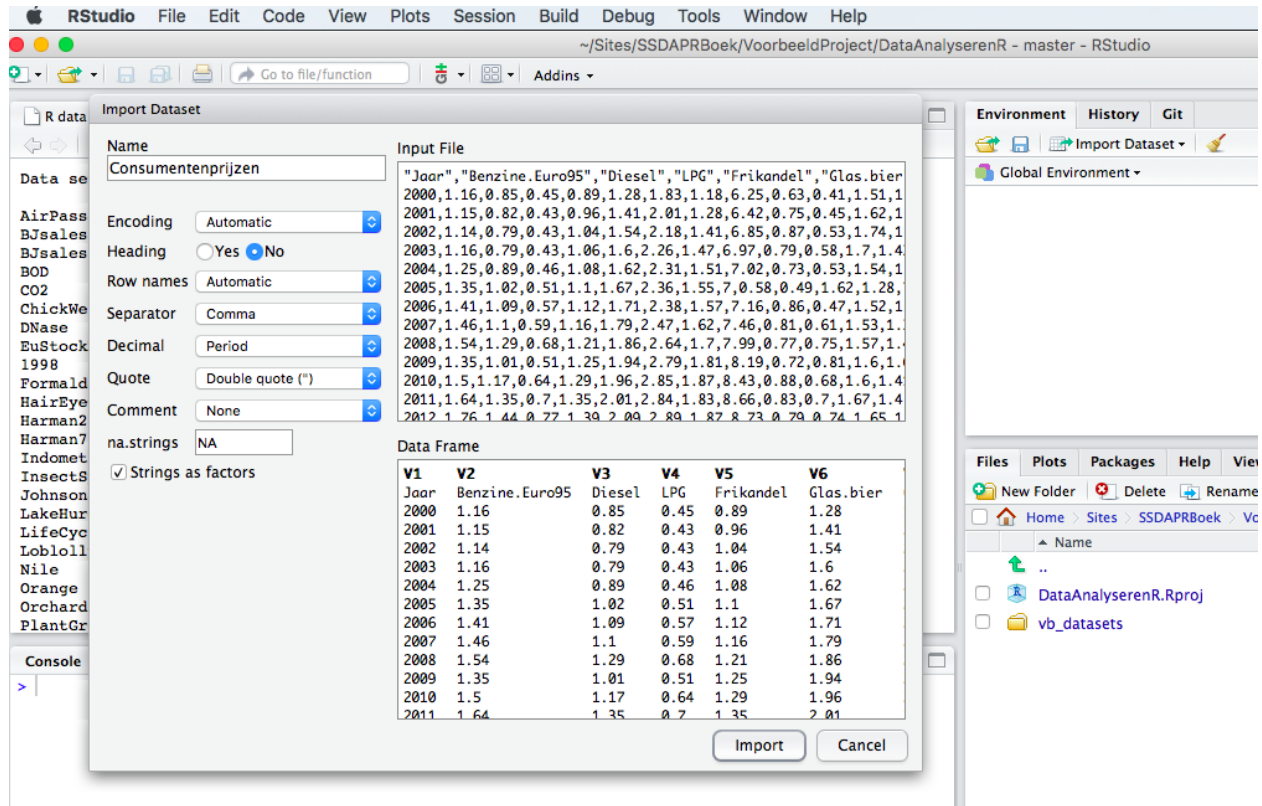


Figure 10: Import Dataset specificatie

Selectie	Omschrijving
* Encoding	De codering van het tekstbestand. Laat deze lekker op “automatic” staan.
* Heading	Heading is een belangrijk onderdeel in deze pagina. Je geeft hier namelijk aan of de eerste rij van de dataset de namen van de kolommen bevat.
* Row names	Hiermee kun je aangeven of rijen een naam hebben in plaats van nummers. (De <i>mtcars</i> dataset bevat row names, namelijk de namen van de auto's).
* Seperator	Geeft aan waardoor de waarden in het originele bestand gescheiden worden. Dit is meestal een komma (,) of een lijstscheidingstekens (;). Is dit anders, dan kun je dit ook selecteren.
* Decimal	Hier kun je de keuze maken hoe decimalen staan genoteerd in de dataset. In het Engels (Amerikaans) is dit een punt (.), waarbij het bij ons een komma (,) is.
* Quote	Hiermee kun je aangeven hoe aanhalingstekens in de dataset genoteerd zijn.

Selectie	Omschrijving
* <code>Comment</code>	Sommige databestanden bevatten commentaar. Net zoals we daar in R <code>#</code> voor gebruiken. Hierin kun je specificeren welk teken wordt gebruikt voor commentaar in de dataset die wordt geïmporteerd. Op deze manier worden deze regels overgeslagen in R;
* <code>na.strings</code>	Geeft aan hoe NA's (lege waarden) worden weergegeven in R.
* <code>Strings as factors</code>	Selecteer of je <code>character</code> waarden (strings) als <code>factor</code> datatype wilt aangeven inladen in R.

Op basis van wat er ingevoerd is in het **Import Dataset** venster, wordt er door RStudio automatisch de code gegenereerd die de data op de gewenste manier importeert. Het onderstaande voorbeeld laat zien dat de code automatisch gegenereerd wordt en daarna direct wordt opgeslagen als het object `Consumentenprijzen`.

```
# Het csv bestand wordt geïmporteerd
Consumentenprijzen <- read.csv("<jouw locatie>/Consumentenprijzen.csv")
```

6.2.3 De dataset bekijken met het `View()` command

RStudio geeft automatisch een `View()` command zodra er een dataset wordt ingeladen. Hierdoor zie je direct de juist geïmporteerde dataset in het venster in RStudio.

6.2.4 Het geïmporteerde object hernoemen

Aangezien `Consumentenprijzen` een vrij lange naam is en het misschien vervelend is om deze naam iedere keer opnieuw volledig in te tikken, kunnen we deze naam veranderen. Hiervoor maken we gewoon een object aan dat precies dezelfde data bevat als `Consumentenprijzen`. We noemen `Consumentenprijzen` voortaan `CP`. Dit doen we met het volgende command:

```
CP <- Consumentenprijzen
```

6.3 Een online dataset in R importeren

Naast ingebouwde datasets en datasets die opgeslagen staan op jouw eigen machine, kun je ook data direct vanaf een locatie op het web importeren in R. Door de `URL` aan te geven die de locatie van de data bevat, kun je zonder dat je de data eerst op jouw machine opslaat direct naar de R-sessie halen.

Hiervoor kun je eenvoudig gebruik maken van dezelfde **Import Dataset** knop. Echter kies je deze keer voor **From Web URL**...

6.3.1 De URL die de dataset bevat opzoeken en invoeren

In dit geval willen we een dataset van de open data website van de Nederlandse overheid (<https://data.overheid.nl>) importeren. Hiervoor gaan we naar de website en selecteren we een dataset. We kiezen voor een dataset over lichtmasten. We willen de data in csv-formaat importeren. Daarom kiezen we voor de link: <http://schagen.gemeentedocumenten.nl/OpenData/OpenbareVerlichting/Lichtmasten.csv>.

In dit boek kun je bij Case 2 zien hoe je een JSON-bestand importeert en omzet in een data frame in R.

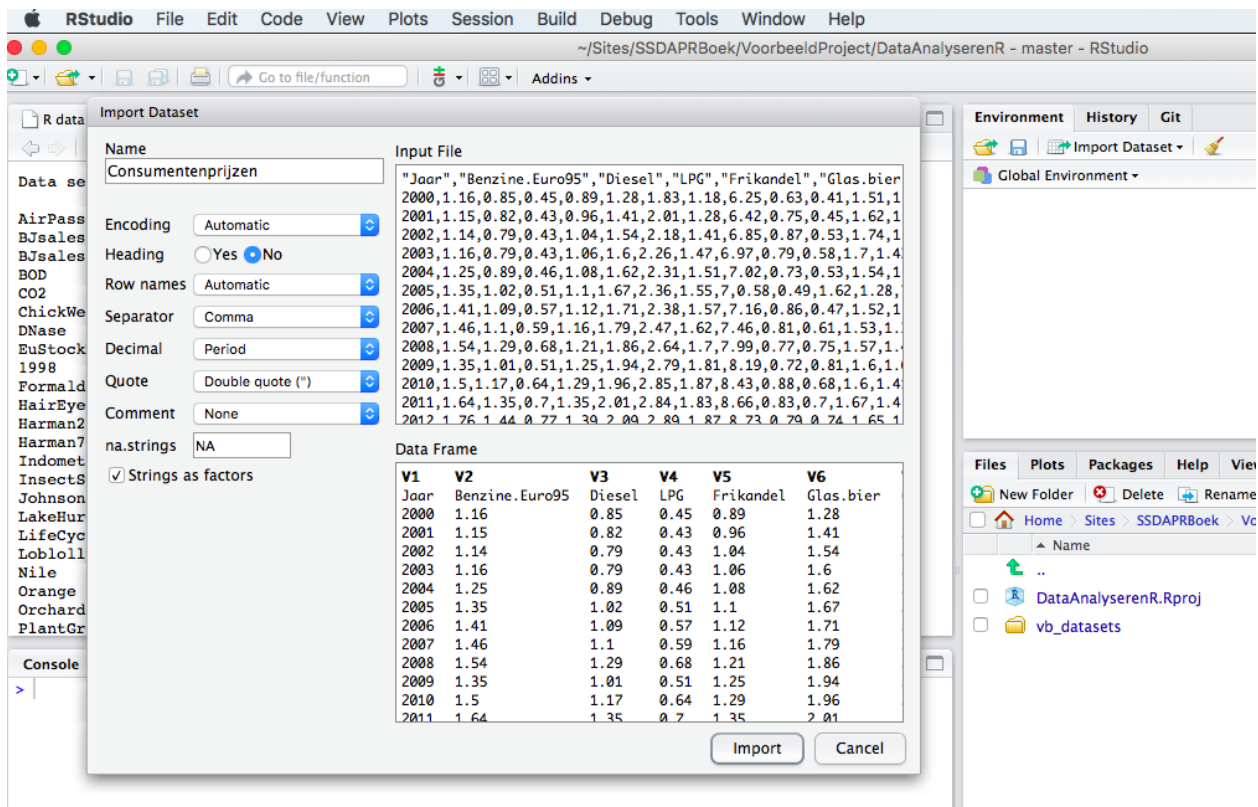


Figure 11: Import Dataset specificatie

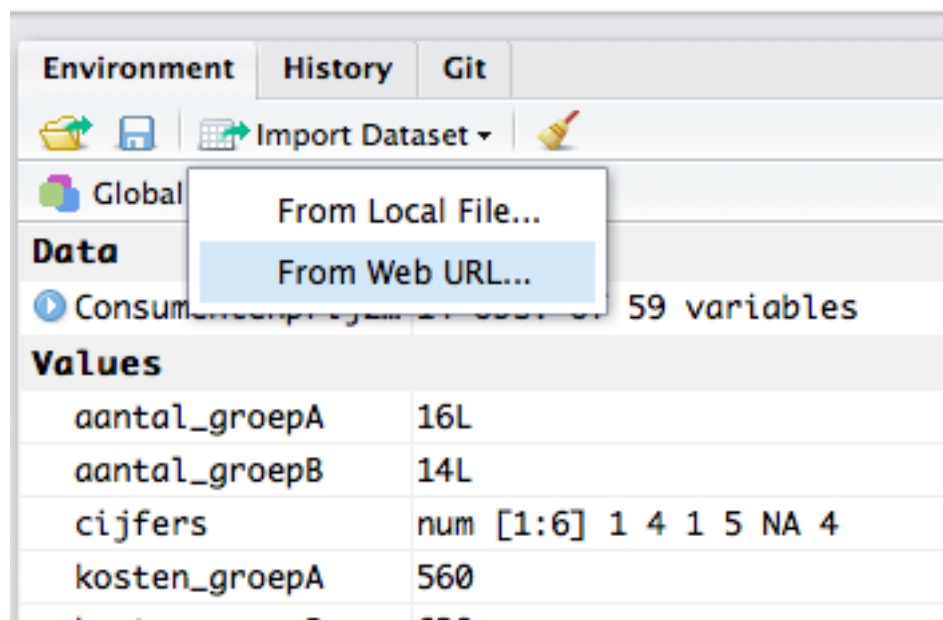


Figure 12: Import from Web URL

Openbare Verlichting

Alle lichtmasten die in beheer zijn bij de gemeente Schagen. Inventarisatie loopt nog.
Gegevens zijn voor ca. 80 % gevuld.

[Meer informatie over deze dataset](#)

lichtmasten

openbare verlichting

Bronnen

Lichtmasten

[download bestand](#)

Toegangslink: <http://www.schagen.nl/opendata>

Downloadlink:

<http://schagen.gemeentedocumenten.nl/OpenData/OpenbareVerlichting/Lichtmasten.csv>

Formaat:

CSV

Wijzigingsdatum:

15 februari, 2016

Grootte bestand:

3,4 MB

Figure 13: Lichtmasten dataset overheid

6.3.2 De dataset achter de URL specificeren

Net als bij het importeren van een lokale dataset, kun je in R studio ook een dataset specificeren die door middel van een URL wordt geïmporteerd. Dit gaat exact op dezelfde manier.

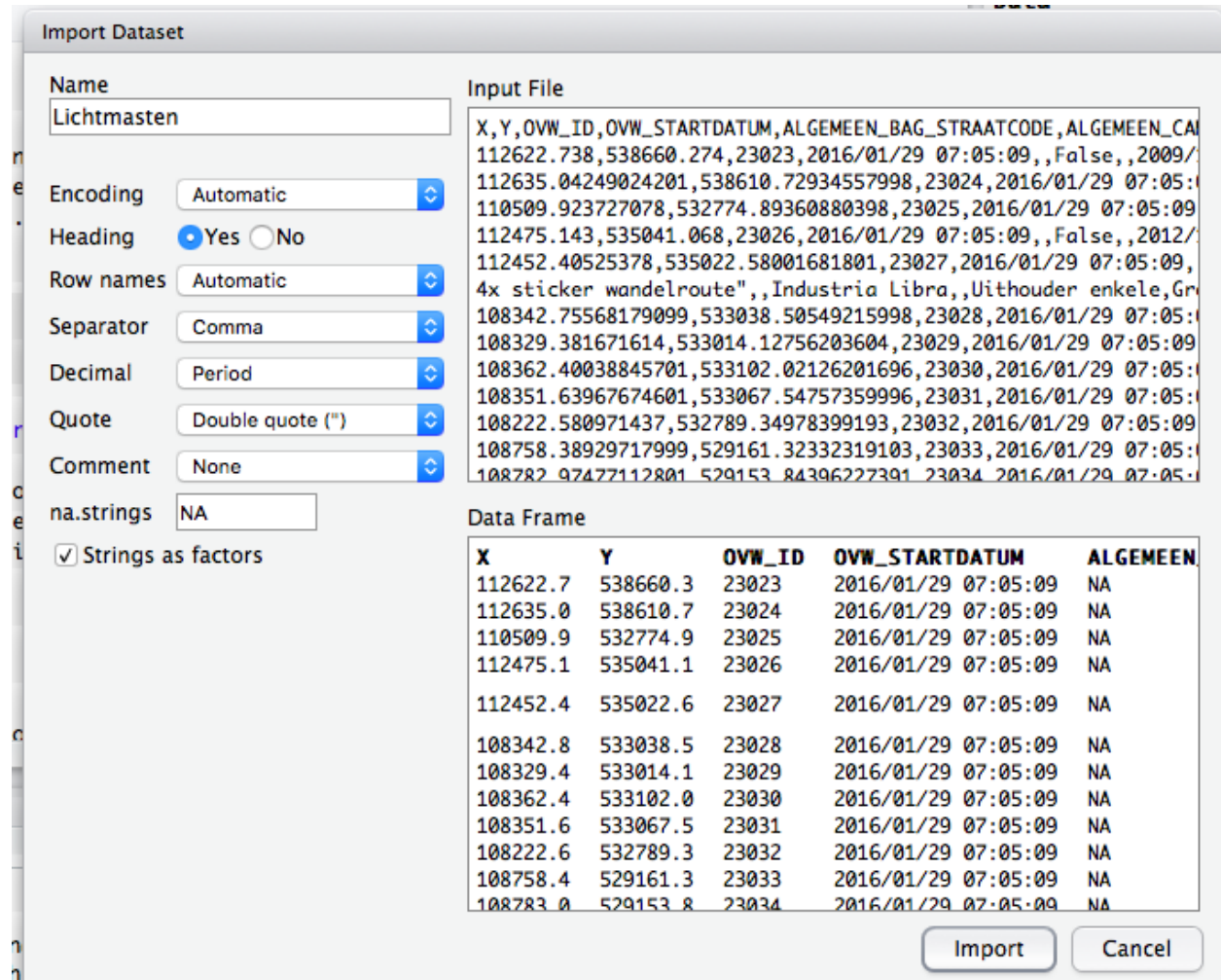


Figure 14: Specificeren URL

6.3.3 Direct vanuit de URL downloaden of door middel van de tussenstap in RStudio

Voor het importeren van de dataset wordt de volgende code gegenereerd. Zoals je kunt zien, staat niet de URL in deze code aangegeven, maar een lokaal pad op onze eigen machine. De reden hiervoor is dat R een tijdelijk bestand (Rtmp) aanmaakt van het bestand, deze opslaat en vanaf deze locatie het bestand importeert naar de R-sessie. Dit is prima en geeft geen problemen.

```
Lichtmasten <-  
read.csv("/var/folders/f9/zhcclk8j25qfft2yw116z_100000gn/T//Rtmp4YZN8X/data3245f921aa5")
```

Mocht je echter per se de data direct vanaf de externe locatie naar de R-sessie importeren, kun je gebruik maken van de `read.csv()` functie in combinatie van de `url()` functie. Dit gaat op de volgende manier.

```
Lichtmasten2 <-
  read.csv(
    url("http://schagen.gemeentedocumenten.nl/OpenData/OpenbareVerlichting/Lichtmasten.csv")
  )
```

Simpelweg geven we in dit command aan dat we de normale `read.csv()` functie willen gebruiken, echter is de locatie van de data extern. Omdat het in dit geval gaat op een URL, moeten we de `url()` functie gebruiken zodat R deze kan uitlezen.

Het nadeel van direct importeren op deze manier is dat je niet door middel van het venster Import Dataset eenvoudig kan specificeren hoe de data geïmporteerd moet worden. Dit zou je dan zelf in de code moeten aangeven als extra argumenten. Dit is makkelijker dan het lijkt, je kunt namelijk gewoon eerst via de R Studio manier de data importeren via het venster op de manier die je wilt. De gegenereerde code en extra argumenten kopieer en plak je dan in de `read.csv(url())` methodiek.

Het resultaat wordt net als bij het importeren van een lokale dataset met de `View()` functie aangegeven.

	X	Y	OVW_ID	OVW_STARTDATUM	ALGEMEEN_BAG_STRAATCODE	ALGEMEEN_CAMERA_MAST	ALGEMEEN_CONSERVERING
1	112622.7	538660.3	23023	2016/01/29 07:05:09	NA	False	NA
2	112635.0	538610.7	23024	2016/01/29 07:05:09	NA	False	NA
3	110509.9	532774.9	23025	2016/01/29 07:05:09	NA	False	NA
4	112475.1	535041.1	23026	2016/01/29 07:05:09	NA	False	NA
5	112452.4	535022.6	23027	2016/01/29 07:05:09	NA	False	NA
6	108342.8	533038.5	23028	2016/01/29 07:05:09	NA	False	NA
7	108329.4	533014.1	23029	2016/01/29 07:05:09	NA	False	NA
8	108362.4	533102.0	23030	2016/01/29 07:05:09	NA	False	NA
9	108351.6	533067.5	23031	2016/01/29 07:05:09	NA	False	NA
10	108222.6	532789.3	23032	2016/01/29 07:05:09	NA	False	NA
11	108758.4	529161.3	23033	2016/01/29 07:05:09	NA	False	NA
12	108783.0	529153.8	23034	2016/01/29 07:05:09	NA	False	NA
13	108833.7	529122.2	23035	2016/01/29 07:05:09	NA	False	NA
14	110518.8	532750.7	23036	2016/01/29 07:05:09	NA	False	NA
15	110517.3	532731.6	23037	2016/01/29 07:05:09	NA	False	NA
16	110504.1	532719.1	23038	2016/01/29 07:05:09	NA	False	NA
17	110492.1	532695.8	23039	2016/01/29 07:05:09	NA	False	NA
18	110463.9	532638.5	23040	2016/01/29 07:05:09	NA	False	NA

Showing 1 to 18 of 11,479 entries

Figure 15: View Dataset URL

6.4 Variabelen van de dataset hernoemen

In sommige gevallen willen we de namen van de variabelen in de dataset veranderen. Hiervoor kunnen we op verschillende manieren de `names()` functie gebruiken.

6.4.1 De namen van de dataset weergeven met de `names()` functie

Door de naam van de dataset in de `names()` functie aan te geven, krijgen we een genummerde lijst met de namen van de data frame.


```
# de namen van CP weergeven
names(CP)
```

```
## [1] "Jaar"                "Benzine.Euro95"
## [3] "Diesel"              "LPG"
## [5] "Frikandel"           "Glas.bier"
## [7] "Glas.rode.huiswijn"  "Kopje.zwarte.koffie"
## [9] "Pizza"               "Aardappelen"
## [11] "Appelmoes"           "Bananen"
## [13] "Bier"                "Boterhamworst"
## [15] "Brood"               "Champignons"
## [17] "Cornflakes"          "Diepvries.patates.frites"
## [19] "Eieren"              "Hagelslag"
## [21] "Hamlappen"           "Kaas"
## [23] "Kipfilet"            "Koffie"
## [25] "Komkommer"           "Koolzuurhoudend.mineraalwater"
## [27] "Krentenbol"          "Macaroni"
## [29] "Margarine"           "Mayonaise"
## [31] "Melk"                "Mineraalwater.zonder.koolzuur"
## [33] "Olijfolie"           "Pasta"
## [35] "Plantaardige.olie"   "Riblappen"
## [37] "Rijst"               "Roomboter"
## [39] "Rundergehakt"        "Shampoo"
## [41] "Sinaasappelsap"      "Sperziebonen.in.blik"
## [43] "Suiker"              "Tandpasta"
## [45] "Thee"                "Toiletpapier"
## [47] "Tomatenketchup"      "Tonijn.in.blik"
## [49] "Vanillevla"          "Zelfrijzend.bakmeel"
## [51] "APK.keuring"         "Autorijles"
## [53] "Bioscoopkaartje"     "Car.wash"
## [55] "Dameskapper"         "Herenkapper"
## [57] "Krant"               "Testament"
## [59] "Zwemles"
```

6.4.2 Namen van de dataset veranderen

Met de `names()` functie kunnen we ook namen uit de data frame veranderen. Hiervoor is de genummerde lijst met namen van de data frame heel handig. We kunnen namelijk met de nummers aangeven van welke variabele we de naam willen veranderen. In dit voorbeeld gaan we de naam van `Benzine.Euro95` veranderen in `Benzine`.

```
# "Benzine.Euro95" staat op plek 2

names(CP)[2] <- "Benzine"
names(CP)
```

```
## [1] "Jaar"                "Benzine"
## [3] "Diesel"              "LPG"
## [5] "Frikandel"           "Glas.bier"
## [7] "Glas.rode.huiswijn"  "Kopje.zwarte.koffie"
## [9] "Pizza"               "Aardappelen"
## [11] "Appelmoes"           "Bananen"
```

```
## [13] "Bier"                "Boterhamworst"
## [15] "Brood"               "Champignons"
## [17] "Cornflakes"         "Diepvries.patates.frites"
## [19] "Eieren"             "Hagelslag"
## [21] "Hamlappen"          "Kaas"
## [23] "Kipfilet"           "Koffie"
## [25] "Komkommer"          "Koolzuurhoudend.mineraalwater"
## [27] "Krentenbol"         "Macaroni"
## [29] "Margarine"          "Mayonaise"
## [31] "Melk"               "Mineraalwater.zonder.koolzuur"
## [33] "Olijfolie"          "Pasta"
## [35] "Plantaardige.olie"  "Riblappen"
## [37] "Rijst"              "Roomboter"
## [39] "Rundergehakt"       "Shampoo"
## [41] "Sinaasappelsap"     "Sperziebonen.in.blik"
## [43] "Suiker"             "Tandpasta"
## [45] "Thee"               "Toiletpapier"
## [47] "Tomatenketchup"     "Tonijn.in.blik"
## [49] "Vanillevla"         "Zelfrijzend.bakmeel"
## [51] "APK.keuring"        "Autorijles"
## [53] "Bioscoopkaartje"    "Car.wash"
## [55] "Dameskapper"        "Herenkapper"
## [57] "Krant"              "Testament"
## [59] "Zwemles"
```

6.5 Omgaan met lege waarden (NA's)

Lege of onbeschikbare waarden worden in R aangegeven met `NA`. In andere systemen worden dit ook wel eens “Missing Values” of `NULL` waarden genoemd. Deze waarden kunnen vervelend zijn omdat ze er voor zorgen dat je bepaalde berekeningen niet kunt uitvoeren. Ook mis je gewoon een stukje belangrijke data. Er zijn verschillende manieren om met deze `NA` waarden om te gaan.

- De hele rijen of kolommen die een `NA` waarde bevat, verwijderen;
- De `NA`'s vervangen door 0;
- De `NA`'s vervangen door een berekende waarde.

In dit hoofdstuk wordt gedemonstreerd hoe je `NA`'s in de hele data frame kunt tellen en hoe je deze kolommen of rijen kunt verwijderen. Maak je geen zorgen, de andere twee methoden worden uitgebreid in **hoofdstuk 11** van dit boek besproken.

6.5.1 NA's in een data frame tellen

De CP dataframe heeft “helaas” geen `NA`'s. Daarom voegen we zelf een kolom toe met waarden die `NA`'s bevat voor de demonstratie. Op deze manier kunnen we handelingen met `NA`'s in een data frame demonstreren.

```
# een vector aanmaken met waarden en NA's
Sushi <- c(5.45,5.75,5.60,NA,6.15,6.30,7.15,7.50,NA,8.10,7.65,NA,8,8.45)

# de vector onder de naam 'sushi' aan CP toevoegen
CP$Sushi <- Sushi

# controleren of de variabele 'sushi' daadwerkelijk aan CP is toegevoegd
names(CP)
```



```
## [1] "Jaar" "Benzine"
## [3] "Diesel" "LPG"
## [5] "Frikandel" "Glas.bier"
## [7] "Glas.rode.huiswijn" "Kopje.zwarte.koffie"
## [9] "Pizza" "Aardappelen"
## [11] "Appelmoes" "Bananen"
## [13] "Bier" "Boterhamworst"
## [15] "Brood" "Champignons"
## [17] "Cornflakes" "Diepvries.patates.frites"
## [19] "Eieren" "Hagelslag"
## [21] "Hamlappen" "Kaas"
## [23] "Kipfilet" "Koffie"
## [25] "Komkommer" "Koolzuurhoudend.mineraalwater"
## [27] "Krentenbol" "Macaroni"
## [29] "Margarine" "Mayonaise"
## [31] "Melk" "Mineraalwater.zonder.koolzuur"
## [33] "Olijfolie" "Pasta"
## [35] "Plantaardige.olie" "Riblappen"
## [37] "Rijst" "Roomboter"
## [39] "Rundergehakt" "Shampoo"
## [41] "Sinaasappelsap" "Sperziebonen.in.blik"
## [43] "Suiker" "Tandpasta"
## [45] "Thee" "Toiletpapier"
## [47] "Tomatenketchup" "Tonijn.in.blik"
## [49] "Vanillevla" "Zelfrijzend.bakmeel"
## [51] "APK.keuring" "Autorijles"
## [53] "Bioscoopkaartje" "Car.wash"
## [55] "Dameskapper" "Herenkapper"
## [57] "Krant" "Testament"
## [59] "Zwemles" "Sushi"
```

We hebben nu een data frame CP die NA's bevat. Met de `is.na` functie kunnen we de NA's in een data frame opzoeken. We kunnen functies combineren als we willen weten welke kolommen missing values bevatten. We combineren de `names()` functie (om de namen van de kolommen/producten te kunnen zien), de `colSums` functie (om het aantal NA's op te tellen) en de `is.na()` functie die aangeeft of de waarde een NA is of niet. Hiermee wordt er voor iedere kolom een optelsom gemaakt van de NA's. Als de uitkomst groter is dan 0, wordt de naam van de kolom aangegeven. Om het aantal NA's te tellen, gebruiken we de `sum()` functie.

```
# evalueren welke waarde een NA is - de tail() functie wordt hier alleen gebruikt om
# de output te beperken
names(CP)[colSums(is.na(CP)) > 0]
```

```
## [1] "Sushi"
```

```
# de NA-waarden optellen
sum(is.na(CP))
```

```
## [1] 3
```

Sushi is dus de enige kolom in de dataframe die NA waarden bevat. De optelsom over de hele dataframe laat zien om hoeveel NA waarden dit gaat.

6.5.2 De rijen met NA's verwijderen

De hele rij of kolom met een NA verwijderen, is een vrij grove aanpak. Echter is dit in sommige gevallen wenselijk. Dit doe je aan de hand van de volgende commands. Om een versie van CP te behouden die wel gewoon alle rijen en kolommen bevat, overschrijven we deze niet, maar maken we een nieuw object aan voor de data frame waar we de kolommen en rijen met NA's hebben weggehaald.

```
# een object CP2 aanmaken die alle data van CP bevat zonder de NA's
CP2 <- na.omit(CP)

# ter controle tellen hoeveel NA's CP2 bevat
sum(is.na(CP2))
```

```
## [1] 0
```

Ter controle vergelijken we ook de dimensies tussen CP en CP2.

```
dim(CP)
```

```
## [1] 14 60
```

```
dim(CP2)
```

```
## [1] 11 60
```

We kunnen zien dat CP2 3 rijen minder heeft dan CP. Door onze `na.omit()` functie zijn er dus 3 rijen verwijderd.

6.6 De `attach()` functie gebruiken

Met de `attach()` functie zorg je ervoor dat je de kolomnamen kunt gebruiken zonder steeds de naam van de data frame aan te geven.

Dus:

Het gemiddelde berekenen van de variabele `Diesel` uit de CP data frame:

```
mean(CP$Diesel)
```

```
## [1] 1.073571
```

Nu wordt de CP data frame als de `default` (standaard) dataset gemaakt. Hierdoor kunnen kolomnaam direct genoemd worden. Het is daarom niet meer nodig om `Dataset + $ + Kolomnaam` syntax te gebruiken.

```
attach(CP)
mean(Diesel)
```

```
## [1] 1.073571
```

Probeer te vermijden Probeer de `attach()` functie zoveel mogelijk te vermijden. De `attach()` functie wijst namelijk data aan namen toe. In dit geval is er data toegewezen aan de namen uit de CP dataset. Probeer daarom zoveel mogelijk gewoon de `$` operator te gebruiken om variabelen te selecteren.

6.7 Een data frame of matrix exporteren

Het is erg eenvoudig om een data frame of matrix die zich in R bevindt te exporteren naar een locatie op jouw machine. De eenvoudigste methode is de `write.table()` functie. Binnen de `write.table()` functie kun je door middel van de parameters aangeven op welke manier de data frame geëxporteerd moet worden. Voorbeelden hiervan zijn de naam van het bestand of het scheidingsteken. In het onderstaande voorbeeld wordt de CP dataset geëxporteerd als Excel-bestand en als csv bestand.

```
# de data frame CP exporteren als csv bestand met een lijstscheidingsteken
# en zonder rijnummers
write.table(CP, "Consumentenprijzen_bewerkt.csv", row.names = FALSE, sep = ";")

# de data frame CP exporteren als Excel bestand met een lijstscheidingsteken
# en zonder rijnummers
write.table(CP, "Consumentenprijzen_bewerkt.xls", row.names = FALSE, sep = ";")
```

Het bestand komt terecht in de map die als **Working Directory** is ingesteld. Om er achter te komen wat de **Working Directory** is, kun je de `getwd()` functie gebruiken.

Als je het bestand naar een andere locatie wilt exporteren, geef je dit pad aan in de bestandsnaam. Als wij bijvoorbeeld het bestand willen exporteren naar de **datasets** map in de **Working Directory**, doen we dat als volgt.

```
write.table(CP, "datasets/Consumentenprijzen_bewerkt.csv", row.names = FALSE, sep = ";")
```

Het bestand is nu opgeslagen in de map **datasets**.

6.7.1 Een Excel-bestand (xlsx) exporteren

Het is ook mogelijk een recenter type Excel bestand te exporteren, een **xlsx** bestand. Hiervoor is het nodig om een package te installeren, het **xlsx** package.

Om gebruik te maken van dit package, is het nodig dat Java is geïnstalleerd op jouw machine.

Een bestand als **xlsx** bestand exporteren gaat als volgt, het is zelfs mogelijk om de naam van de sheet aan te geven.

```
require(xlsx)
write.xlsx(CP,
           "datasets/Consumentenprijzen_bewerkt.xlsx",
           sheetName = "Overzicht Prijzen",
           row.names = FALSE)
```

6.8 Samenvatting

6.8.1 Data uit interne en externe bronnen importeren

RStudio maakt het eenvoudig om interne en externe data in te laden naar de R-sessie. Tegenwoordig zijn er steeds meer mogelijkheden om externe databronnen te ontsluiten. Zoals je hebt kunnen zien, biedt RStudio genoeg handvatten om deze bronnen te ontsluiten. Daarbij zijn er ook verschillende functies en packages om data in te laden. In **Case 2** van dit boek, wordt bijvoorbeeld gebruik gemaakt van het *jsonlite* package om externe JSON-data te ontsluiten.

Externe data en JSON Websites bieden door middel van API's (*Application Program Interfaces*) de mogelijkheid om hun data te gebruiken. Deze data wordt meestal als **JSON** formaat geïnstalleerd. JSON staat voor *Javascript Object Notation* en is opgebouwd uit Javascript objecten. Dit zijn *key - value* paren. In R wordt JSON gezien als een lijst (*list*) object. Het daarna eenvoudig om het om te zetten en te gebruiken in de R-sessie, net als in **Case 2** te zien is.

6.8.2 Specificeren hoe de data geïmporteerd moet worden

Een belangrijk onderdeel van data importeren in een R-sessie, is specificeren op welke manier de data ingeladen moet worden. Belangrijke punten hierbij zijn bijvoorbeeld aangeven wat de decimalen zijn, of de eerste rij van de dataset de kolomnamen zijn en wat de scheidingstekens zijn. De makkelijkste manier om data te importeren met RStudio is de *Import Dataset* functie gebruiken. RStudio heeft de goede eigenschap dat het voor het grootste deel al de juiste instellingen voor jou selecteert. Echter moet je er altijd zelf goed naar kijken of het echt de manier is hoe jij data wilt importeren.

6.8.3 Data exporteren

In sommige situaties wil je de data exporteren buiten de R-sessie. Dit kan zijn omdat R een tussenstap is voor databewerking of verrijking of omdat je met dezelfde dataset in een ander systeem verder wilt gaan. R biedt genoeg mogelijkheden om dit te doen. Een data frame of matrix exporteren is daarbij zeer eenvoudig. Ook hierbij kun je specificeren hoe de data geëxporteerd wordt. Je hierbij bijvoorbeeld ook aangeven of de tabel rij- of kolomnamen heeft, wat het scheidingsteken moet zijn en welke extensie de dataset moet hebben (.xls, .csv, .txt etc.).

6.9 Oefeningen

6.9.0.0.1 Exporteer de ingebouwde *mtcars* dataset zodat deze perfect kan worden geopend in Excel (mag .csv of .xls zijn)

6.9.0.0.2 Importeer een externe dataset. Dit kan bijvoorbeeld een tabel zijn op de website van de Centraal Bureau voor de statistiek

6.9.0.0.3 Exporteer deze externe dataset als .csv bestand

7 Data frame bewerkingen

Zoals eerder in deze handleiding is aangegeven, is de data frame de belangrijkste entiteit in R voor een dataset. Een dataframe is meestal de input voor een functie waarop bepaalde bewerkingen worden uitgevoerd. Echter kun je op een data frame zelf verschillende bewerkingen doen.

Enkele bewerkingen op een data frame kunnen al waardevolle inzichten opleveren uit de dataset. Hiervoor pas je verschillende selecties en filters toe op de data frame. Dit wordt ook wel het slicing and dicing genoemd in de Business Intelligence wereld. Deze metafoor komt van de bewerkingen die je op een rubik's kubus doet: slicen: horizontaal draaien en dicen: verticaal draaien. In dit hoofdstuk wordt dit slicen en dicen op een data frame gedemonstreerd aan de hand van een aantal voorbeelden.

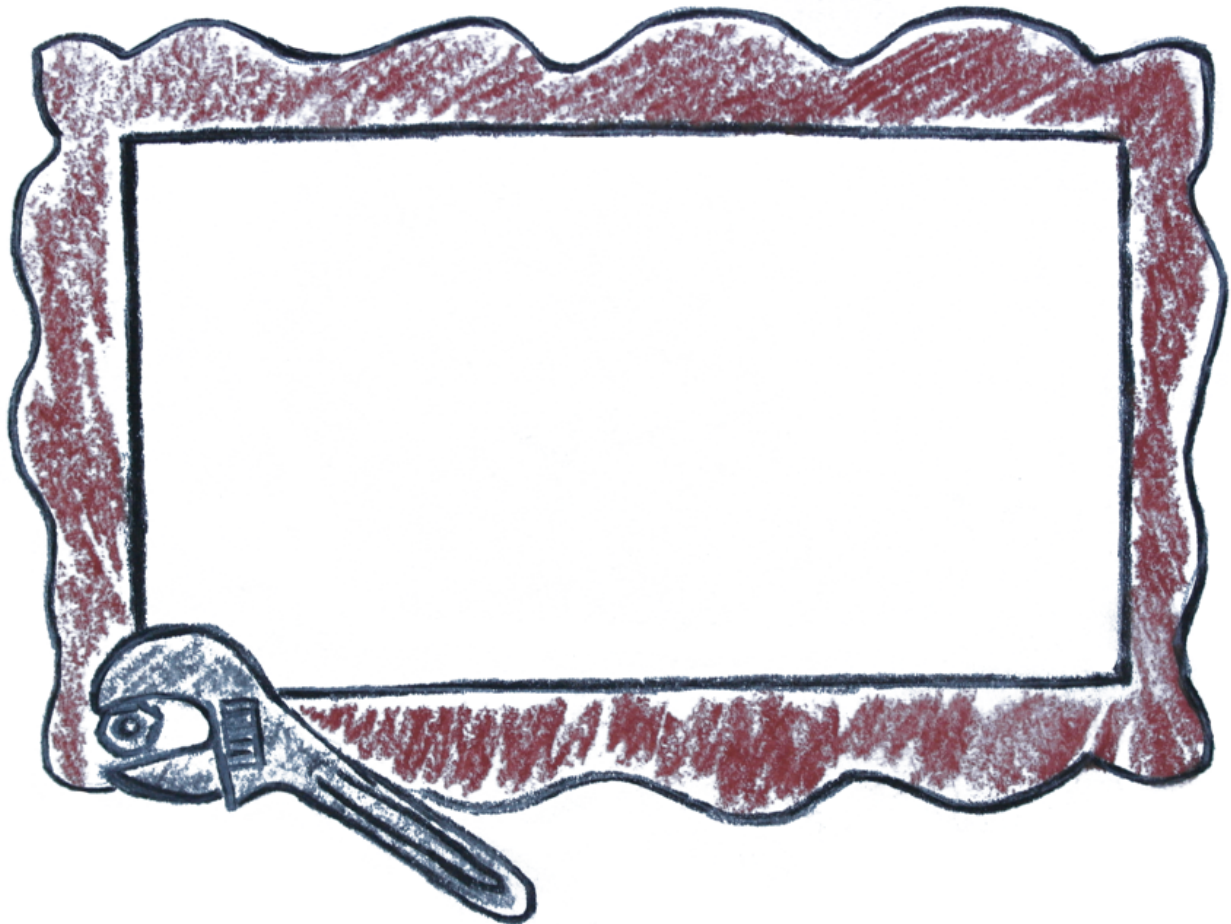


Figure 16:

7.1 De “mtcars” dataset

Ook voor deze voorbeelden wordt de `mtcars` dataset gebruikt. Voor deze dataset wordt het object `autos` aangemaakt, die dezelfde data bevat als `mtcars`. Op deze manier kunnen we altijd de beginstand van `mtcars` terughalen voor het geval we de dataset op z’n zachtst gezegd verpesten.

```
# de interne dataset 'mtcars' openen
data(mtcars)

# het object 'autos' aanmaken die de data van 'mtcars' bevat
autos <- mtcars

head(autos)
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46 0  1   4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02 0  1   4    4
## Datsun 710     22.8   4  108  93 3.85 2.320 18.61 1  1   4    1
## Hornet 4 Drive  21.4   6  258 110 3.08 3.215 19.44 1  0   3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02 0  0   3    2
## Valiant        18.1   6  225 105 2.76 3.460 20.22 1  0   3    1
```

7.2 Rijen en kolommen van een data frame selecteren

Rijen en kolommen selecteren doe je aan de hand van *square brackets* `[ , ]`. Hierbij selecteer je met de waarde voor de komma de rij en na de komma de kolom. Respectievelijk kun je hiermee rijen en kolommen in een data frame selecteren.

7.2.1 Een rij uit een data frame selecteren

```
# rijnummer 1 selecteren uit de 'autos' dataset
autos[1,]
```

```
##           mpg cyl  disp  hp  drat    wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46 0  1   4    4
```

7.2.2 Een kolom uit een dataframe selecteren

```
# kolomnummer 3 selecteren uit de 'autos' dataset
autos[,3]
```

```
## [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6
## [12] 275.8 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0
## [23] 304.0 350.0 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

7.2.3 Meerdere rijen en kolommen uit een data frame selecteren

```
# rij 1 t/m 3 selecteren
autos[c(1:3),]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
# rij 1,3,7 en 8 selecteren
autos[c(1,3,7,8),]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21.0   6 160.0 110 3.90 2.62 16.46  0  1    4    4
## Datsun 710  22.8   4 108.0  93 3.85 2.32 18.61  1  1    4    1
## Duster 360  14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
## Merc 240D   24.4   4 146.7  62 3.69 3.19 20.00  1  0    4    2
```

```
# kolom 4 en 5 selecteren
autos[,c(4,5)]
```

```
##           hp drat
## Mazda RX4      110 3.90
## Mazda RX4 Wag  110 3.90
## Datsun 710       93 3.85
## Hornet 4 Drive  110 3.08
## Hornet Sportabout 175 3.15
## Valiant         105 2.76
## Duster 360      245 3.21
## Merc 240D        62 3.69
## Merc 230         95 3.92
## Merc 280        123 3.92
## Merc 280C        123 3.92
## Merc 450SE       180 3.07
## Merc 450SL       180 3.07
## Merc 450SLC      180 3.07
## Cadillac Fleetwood 205 2.93
## Lincoln Continental 215 3.00
## Chrysler Imperial 230 3.23
## Fiat 128         66 4.08
## Honda Civic      52 4.93
## Toyota Corolla   65 4.22
## Toyota Corona    97 3.70
## Dodge Challenger 150 2.76
## AMC Javelin      150 3.15
## Camaro Z28       245 3.73
## Pontiac Firebird 175 3.08
## Fiat X1-9        66 4.08
## Porsche 914-2     91 4.43
## Lotus Europa     113 3.77
## Ford Pantera L   264 4.22
## Ferrari Dino     175 3.62
## Maserati Bora    335 3.54
## Volvo 142E       109 4.11
```

```
# rij 1,4,5 en kolom 2,5,7 selecteren
autos[c(1,4,5),c(2,5,7)]
```

```
##              cyl drat  qsec
## Mazda RX4      6 3.90 16.46
## Hornet 4 Drive  6 3.08 19.44
## Hornet Sportabout 8 3.15 17.02
```

7.2.4 Een specifiek datapunt uit een data frame selecteren

```
# de data selecteren die op rij 4 en kolom 3 staat
autos[4,3]
```

```
## [1] 258
```

7.3 Kolommen uit een data frame selecteren op basis van de naam

Soms wil je direct de naam van de variabele selecteren uit de data frame. Dit kan over het algemeen op twee makkelijke manieren, met square brackets en met de \$ operator:

```
# variabele selecteren aan de hand van square brackets
autos[["qsec"]]
```

```
## [1] 16.46 17.02 18.61 19.44 17.02 20.22 15.84 20.00 22.90 18.30 18.90
## [12] 17.40 17.60 18.00 17.98 17.82 17.42 19.47 18.52 19.90 20.01 16.87
## [23] 17.30 15.41 17.05 18.90 16.70 16.90 14.50 15.50 14.60 18.60
```

```
# variabele selecteren met de $ operator
autos$qsec
```

```
## [1] 16.46 17.02 18.61 19.44 17.02 20.22 15.84 20.00 22.90 18.30 18.90
## [12] 17.40 17.60 18.00 17.98 17.82 17.42 19.47 18.52 19.90 20.01 16.87
## [23] 17.30 15.41 17.05 18.90 16.70 16.90 14.50 15.50 14.60 18.60
```

7.4 Filters toepassen op data frames

Door gebruik te maken van filters, wordt alleen data uit de data frame weergegeven die het criterium of de criteria van de filters matchen.

```
# een filter toepassen: alleen auto's die 6 of meer cylinders hebben (cyl > 6)
autos[autos$cyl >= 6,]
```

```
##              mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive  21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
```


## Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
## Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
## Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
## Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
## Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
## AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
## Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
## Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
## Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
## Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
## Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8

7.5 Berekende kolommen toevoegen

In dit voorbeeld voegen we een kolom toe aan de `autos` dataset die aangeeft of er sprake is van een sportauto of niet. Dit doen we door te kijken naar het aantal pk's (`hp`) dat een auto heeft. Voor het gemak zeggen we dat als een auto meer dan 150 `hp` heeft, er sprake is van een sportauto. Deze kolom noemen we `sportsCar`.

```
# een kolom 'sportsCar' aanmaken in de 'autos' data set
# op basis van evaluatie van de kolom 'hp'

autos$sportsCar <- ifelse(autos$hp >= 150, TRUE, FALSE)
head(autos$sportsCar)
```

```
## [1] FALSE FALSE FALSE FALSE TRUE FALSE
```

We hebben met boolean (logical datatype) waarden, `TRUE`, of `FALSE` aangegeven of er sprake is van een sportauto. Echter hadden we ook iets anders kunnen gebruiken, zoals een bepaalde waarde of een `string`. We bekijken op basis van `mpg` (miles per gallon) of een auto zuinig is of niet. Voor het gemak zeggen we dat als een auto 20 `mpg` of meer heeft, we spreken over een zuinige auto.

```
autos$efficiënt <- ifelse(autos$mpg >= 20,"efficiënt","non-efficiënt")
head(autos$efficiënt)
```

```
## [1] "efficiënt" "efficiënt" "efficiënt" "efficiënt"
## [5] "non-efficiënt" "non-efficiënt"
```

7.6 Kolommen met een bepaalde waarde selecteren

Als we nu alle efficiënte auto's willen halen uit de `autos` dataset, willen we alle rijen hebben die in de kolom `efficiënt` de waarde "efficiënt" bevat. Deze selectie maken we als volgt:

```
autos[autos$efficiënt %in% "efficiënt",]
```

```
##           mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46 0  1   4   4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02 0  1   4   4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61 1  1   4   1
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44 1  0   3   1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00 1  0   4   2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90 1  0   4   2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47 1  1   4   1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52 1  1   4   2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90 1  1   4   1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01 1  0   3   1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90 1  1   4   1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70 0  1   5   2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90 1  1   5   2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60 1  1   4   2
##           sportsCar efficiënt
## Mazda RX4             FALSE efficiënt
## Mazda RX4 Wag         FALSE efficiënt
## Datsun 710             FALSE efficiënt
## Hornet 4 Drive        FALSE efficiënt
## Merc 240D              FALSE efficiënt
## Merc 230               FALSE efficiënt
## Fiat 128               FALSE efficiënt
## Honda Civic            FALSE efficiënt
## Toyota Corolla         FALSE efficiënt
## Toyota Corona          FALSE efficiënt
## Fiat X1-9              FALSE efficiënt
## Porsche 914-2          FALSE efficiënt
## Lotus Europa           FALSE efficiënt
## Volvo 142E             FALSE efficiënt
```

We kunnen ook meer filters tegelijk toepassen. Laten we zoeken naar een efficiënte sportauto. Hierbij gebruiken we onze zojuist zelf aangemaakte kolommen.

```
autos[autos$sportsCar %in% TRUE & autos$efficiënt %in% "efficiënt",]
```

```
## [1] mpg      cyl      disp      hp      drat      wt      qsec
## [8] vs      am      gear      carb      sportsCar efficiënt
## <0 rows> (or 0-length row.names)
```

Zoals je kunt zien, worden er 0 rows aangegeven. Er bestaan in deze dataset dus geen efficiënte sportauto's. Althans, op de manier waarop wij het gedefinieerd hebben.

7.7 Kolomnamen van een data frame veranderen

Met de `names()` functie kunnen kolomnamen van kun je kolomnamen in de `data frame` veranderen.

```
# Huidige naam van kolom nummer 12
names(autos)[12]
```

```
## [1] "sportsCar"
```

```
# met het volgende command veranderen we de naam van kolom nummer 12 in de 'autos' dataset

names(autos)[12] <- "Sportwagen"

# alle namen van de 'autos' dataset bekijken
# zoals je kunt zien, is de naam "sportsCar" verdwenen
# en is "Sportwagen" # daarvoor in de plaats gekomen

names(autos)
```

```
## [1] "mpg"      "cyl"      "disp"     "hp"      "drat"
## [6] "wt"       "qsec"     "vs"       "am"      "gear"
## [11] "carb"     "Sportwagen" "efficiënt"
```

7.8 Samenvatting

7.8.0.0.1 Slicen en Dicen

Door te slicen en dicen met data frames, gebruik je R als een echte Business Intelligence tool. Er zijn extreem veel mogelijkheden om selecties te maken met data frames in R. Hierdoor kun je eigenlijk alle data uit een dataset op iedere manier selecteren. Daarbij kun je het maken van selecties in data frames combineren met handige functies.

7.8.0.0.2 Gecalculeerde kolommen aanmaken

Gecalculeerde kolommen zijn eenvoudig te maken, zoals de `autos$efficiënt` en `autos$sportsCar` kolommen die in dit hoofdstuk aangemaakt zijn. Op deze manier kun je een extra kolom maken die het resultaat is van een evaluatie op andere kolommen.

7.9 Oefeningen

7.9.0.0.1 Maak een selectie van auto's in de `mtcars` data frame die meer dan 110 pk (hp) hebben

7.9.0.0.2 Voeg een nieuwe kolom toe die aangeeft met “Ja” of “Nee” of de auto wel of niet meer dan 110 hp heeft

7.9.0.0.3 Voeg zelf een auto toe aan de data frame met dezelfde eigenschappen als in de `mtcars` dataset (Tip, maak gebruik van de `rbind()` functie)

8 Werken met lijsten en matrixen in R

Een matrix is een ander object in R dat een verzameling aan data onderverdeeld in rijen en kolommen kan bevatten. Een matrix lijkt veel op een data frame, maar heeft op sommige gebieden andere eigenschappen. Aan de hand van verschillende demonstraties en databewerkingen, krijg je een goed van matrixen in R en hoe deze zich verhouden ten opzichte van een data frames in R.

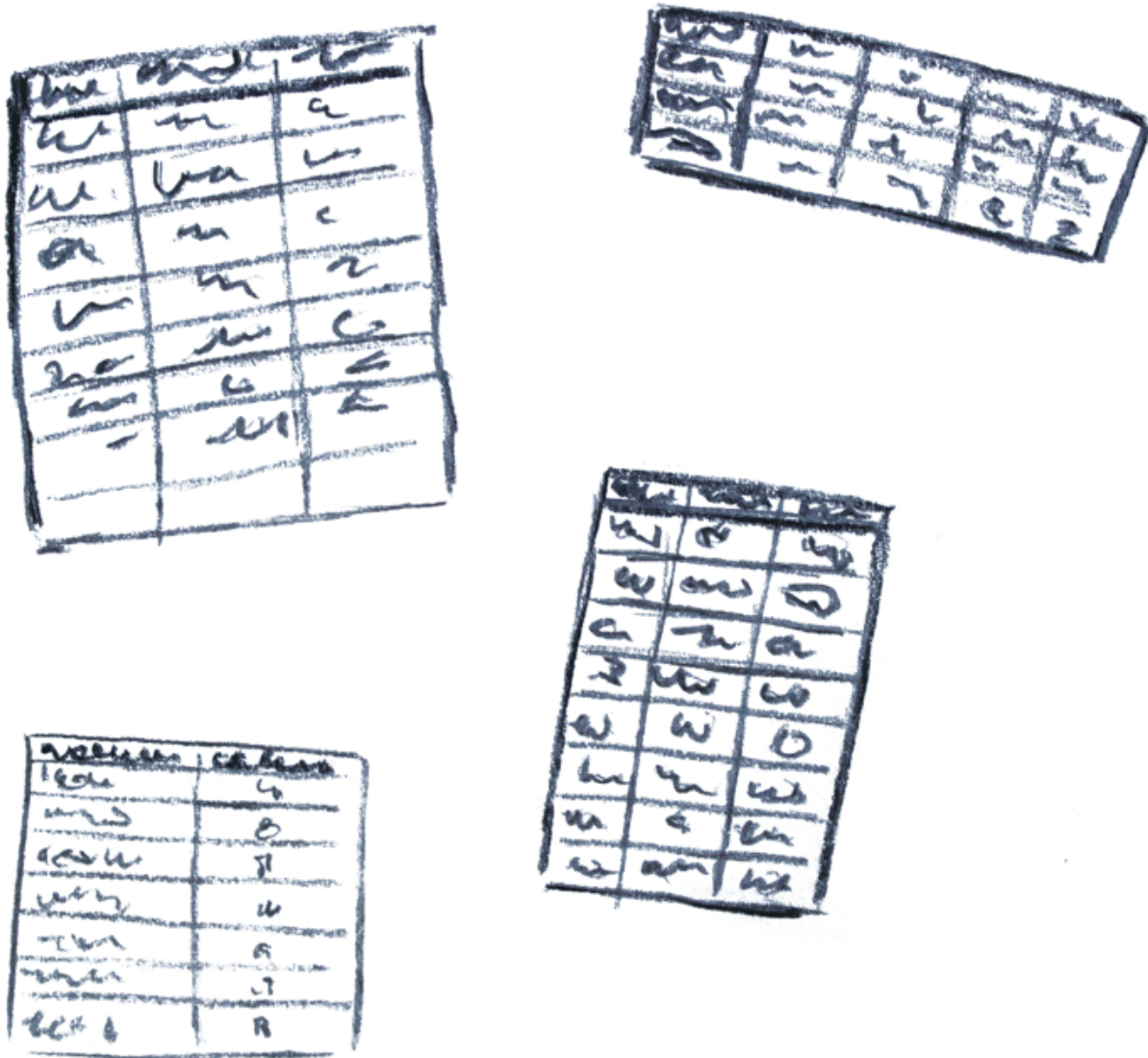


Figure 17:

8.1 Handmatig een matrix maken

De handigste manier om een matrix aan te maken in R is met de `matrix()` functie. Om een matrix op de manier te maken die je wenst, kun je standaard 3 parameters geven aan de `matrix()` functie:

- `data` : Met welke data of met welke waarden de matrix gevuld moet worden;
- `nrow` : Het aantal rijen dat de matrix moet hebben;
- `ncol` : Het aantal kolommen dat de matrix moet hebben.

Om dit te demonstreren, maken we een matrix aan die de waarden 1 t/m 10 bevat, verdeeld in 2 kolommen en 5 rijen.

```
# de waarden 1 t/m 10 genereren
getallen <- 1:10

# de matrix "getallen" aanmaken met "getallen" als data
getallenMatrix <- matrix(getallen, nrow = 5, ncol = 2)

getallenMatrix
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

Zoals je kunt zien, wordt de matrix gevuld met de getallen die het `getallen` object bevat.

8.1.1 Een matrix vullen per rij of per kolom

Ook kun je zien dat dit per kolom opeenvolgend is gedaan. Als we de matrix willen vullen per rij in plaats van per kolom, kunnen we een `byrow` parameter toevoegen. Hiermee krijgen we het volgende resultaat.

```
# getallenMatrix2 aanmaken, in dit geval vullen per rij
getallenMatrix2 <- matrix(getallen, nrow = 5, ncol = 2, byrow = TRUE)

getallenMatrix2
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

Zoals je kunt zien, is de matrix nu per rij gevuld.

8.2 Lijsten (lists) maken in R

Net als bij een data frame, kan een matrix rij- en kolomnamen bevatten. Dit is heel handig als je bijvoorbeeld later data wilt opzoeken uit specifieke rijen en kolommen. Om dit te demonstreren, maken we een matrix waarin we voor een aantal personen een aantal eigenschappen bijhouden. Hierbij zetten we de namen van de personen op de rijen en de eigenschappen op de kolomnamen.

```
# lijsten aanmaken van persoonsgegevens

Harry <- list(
  voornaam = "Harry", achternaam = "de Hengst", lengte = 1.87,
  gewicht = 81.3, woonplaats = "Leiden", rookt = FALSE, studie = "Bedrijfskunde",
  geslacht = "M", leeftijd = 21
)
```

8.2.1 Key-value pair in lists

Lijsten (lists) zijn hele handige objecten in R. Lists bestaan namelijk uit vectoren die een naam hebben, in dit geval bijvoorbeeld `leeftijd`, `voornaam`, `rookt` enzovoort. Deze namen worden *keys* genoemd, de waarden achter deze namen worden *values* genoemd. Hierdoor spreek je over *key - value pairs*, of in Nederlands: de naam van de vector en de waarde in deze vector. Door deze opbouw, kun je net zoals bij een dataframe met de `$` operator de waarde achter een key oproepen.

8.2.2 Verschillende datatypes opslaan in lists

Een sterk punt van de `list` structuur is dat je waarden van verschillende datatypes er in kunt opslaan. Dit kan niet met een vector. Als je namelijk een vector maakt die allemaal numerieke waarden heeft en slechts een enkele `character` waarde, wordt de vector gelijk omgezet naar een `character` vector. Hierdoor worden alle numerieke waarden in de vector gelijk onbruikbaar. Dit probleem heb je met een `list` echter niet. Iedere waarde in een list heeft zijn eigenschappen.

```
waardenVector <- c(1, 2, 3, 4, "Woord")

waardenVector
```

```
## [1] "1"      "2"      "3"      "4"      "Woord"
```

```
class(waardenVector)
```

```
## [1] "character"
```

```
waardenLijst <- list(1, 2, 3, 4, "Woord")

waardenLijst
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
```

```
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] "Woord"
```

```
class(waardenLijst)
```

```
## [1] "list"
```

Je kunt aan de aanhalingstekens bij de nummers in de `waardenVector` zien dat de nummers omgezet zijn naar het `character` datatype. Dit is bij de `waardenLijst` echter niet zo.

```
class(Harry)
```

```
## [1] "list"
```

```
# de leeftijd van Harry terugvinden
```

```
Harry$leeftijd
```

```
## [1] 21
```

```
# de gehele 'Harry' lijst weergeven
```

```
Harry
```

```
## $voornaam
## [1] "Harry"
##
## $achternaam
## [1] "de Hengst"
##
## $lengte
## [1] 1.87
##
## $gewicht
## [1] 81.3
##
## $woonplaats
## [1] "Leiden"
##
## $rookt
## [1] FALSE
##
## $studie
## [1] "Bedrijfskunde"
```

```
##
## $geslacht
## [1] "M"
##
## $leeftijd
## [1] 21
```

Key-value pairs kom je in de Business Intelligence en Analytics wereld steeds meer tegen. Dit is een van de meeste gebruikte datastructuren die onder NoSQL categorie vallen. De JSON-datastructuur werkt ook met key-value pairs, daarom wordt JSON-data in R geïnterpreteerd als een *list* datatype.

Het is niet verplicht om de namen in een lijst een naam te geven. Het nadeel hiervan is echter dat je deze dan ook niet kunt oproepen zoals in het vorige voorbeeld.

```
Japie <- list("Japie","Krekel", 1.47, 65.3, "Utrecht", TRUE,"Rechten","M",23)
Roos <- list("Roos","Jansen", 1.67, 60, "Utrecht", FALSE,"Geneeskunde","V",25)
Hans <- list("Hans","Klock", 1.78, 78.2, "Utrecht", FALSE,"Bedrijfskunde","M",20)
Marco <- list("Marco","Polo", 1.89, 81.3, "Leiden", FALSE,"Rechten","M",19)
Erica <- list("Erica","Jansen", 1.53, 65.3, "Leiden", TRUE,"Geneeskunde","V",27)
```

8.3 Een matrix samenstellen op basis van lijsten

Nu we de data hebben gemaakt, kunnen we de matrix samenstellen. In dit geval willen we rij- en kolomnamen toevoegen aan de matrix. Deze rij- en kolomnamen zijn de namen van de personen en de namen van de eigenschappen. Hiervoor gebruiken we de parameter `dimnames` in de `matrix()` functie. Hiervoor moeten we een lijst aanmaken die twee reeksen aan namen bevat, een voor de rijnamen (als eerst) en een voor de kolomnamen (als tweede). Hiervoor wordt de `list()` functie gebruikt. In het onderstaande voorbeeld krijg je een beter beeld over hoe dit in zijn werk gaat.

```
# de personenMatrix maken op basis van lijsten

personenMatrix <-
  matrix(c(Harry, Japie, Roos, Hans, Marco, Erica),
        nrow = 6,
        ncol = 9,
        byrow = TRUE,
        dimnames = list(
          c("Harry","Japie","Roos","Hans","Marco","Erica"),
          c("Voornaam","Achternaam","Lengte","Gewicht","Woonplaats","Thuiswonend",
            "Studie", "Geslacht", "Leeftijd")))

personenMatrix
```

```
##      Voornaam Achternaam Lengte Gewicht Woonplaats Thuiswonend
## Harry "Harry"   "de Hengst" 1.87    81.3    "Leiden"   FALSE
## Japie "Japie"   "Krekel"   1.47    65.3    "Utrecht"  TRUE
## Roos  "Roos"   "Jansen"   1.67    60     "Utrecht"  FALSE
## Hans  "Hans"   "Klock"    1.78    78.2    "Utrecht"  FALSE
## Marco "Marco"   "Polo"     1.89    81.3    "Leiden"   FALSE
## Erica "Erica"   "Jansen"   1.53    65.3    "Leiden"   TRUE
##      Studie      Geslacht Leeftijd
```



```
## Harry "Bedrijfskunde" "M"      21
## Japie "Rechten"          "M"    23
## Roos  "Geneeskunde"      "V"    25
## Hans  "Bedrijfskunde"    "M"    20
## Marco "Rechten"          "M"    19
## Erica "Geneeskunde"      "V"    27
```

Zoals je kunt zien, heeft de matrix nu rij- en kolomnamen.

8.4 Data uit de matrix selecteren

Met matrixen kun je heel handig data opzoeken. Dit demonstreren we met de matrix die we in de vorige stap hebben gemaakt. Aangezien we rij- en kolomnamen hebben toegevoegd aan de matrix, is het makkelijker om specifieke datapunten op te zoeken.

8.4.1 Data uit een matrix opzoeken met rij- en kolomnummers

Net als bij een data frame, kunnen we de rij en kolomnummers invoeren om het datapunt op die plek in de matrix op te halen.

```
# de waarde uit rij 4 en kolom 7 bekijken
personenMatrix[4,7]
```

```
## [[1]]
## [1] "Bedrijfskunde"
```

8.4.2 Data uit een matrix opzoeken met rij- en kolomnamen

Wat zeer handig is van deze matrix, is dat we rij- en kolomnamen hebben toegevoegd. Hiermee kunnen we heel duidelijk een specifiek data punt, of meerdere dataverzamelingen uit de matrix terughalen. Het is net alsof je tegen de matrix spreekt, bijvoorbeeld: “Wat is de woonplaats van Erica?”.

```
personenMatrix["Erica","Woonplaats"]
```

```
## [[1]]
## [1] "Leiden"
```

Op deze manier kun je zien op welke manier gebruik van rijnummers veel gemak kan opleveren bij het opzoeken van data uit een matrix.

8.5 Calculaties maken met matrixen

Matrixen zijn zeer handig bij het toepassen van calculaties op meerdere waarden. Om dit te demonstreren, gebruiken we de matrixen die we in het begin van dit hoofdstuk hebben aangemaakt: `getallenMatrix` en `getallenMatrix2`.

8.5.1 Sommen toepassen op de gehele matrix

Het is zeer eenvoudig om een bepaalde berekening toe te passen op alle waarden in een matrix:

```
getallenMatrix
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```
# 3 optellen bij alle waarden in "getallenMatrix"
getallenMatrix + 3
```

```
##      [,1] [,2]
## [1,]    4    9
## [2,]    5   10
## [3,]    6   11
## [4,]    7   12
## [5,]    8   13
```

```
# alle waarden in "getallenMatrix" vermendigvuldigen met 2
getallenMatrix *2
```

```
##      [,1] [,2]
## [1,]    2   12
## [2,]    4   14
## [3,]    6   16
## [4,]    8   18
## [5,]   10   20
```

8.5.2 Matrixen met elkaar vermenigvuldigen

Je kunt ook een calculatie toepassen op een matrix op basis van de waarden uit een andere matrix. In het onderstaande voorbeeld worden alle waarden uit `getallenMatrix` vermendigvuldigt met de waarden uit `getallenMatrix2` die op dezelfde plek staan. Het voorbeeld daarna vermenigvuldigt alle waarden uit `getallenMatrix` met de waarden die op de eerste rij staan van `getallenMatrix2`.

```
getallenMatrix
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```
getallenMatrix2
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

```
getallenMatrix* getallenMatrix2
```

```
##      [,1] [,2]
## [1,]    1  12
## [2,]    6  28
## [3,]   15  48
## [4,]   28  72
## [5,]   45 100
```

```
getallenMatrix* getallenMatrix2[1,]
```

```
##      [,1] [,2]
## [1,]    1  12
## [2,]    4    7
## [3,]    3   16
## [4,]    8    9
## [5,]    5   20
```

Voor meer voorbeelden van matrix calculaties in R, kun je het artikel van [4] “R Library: Matrices and matrix computations in R” van IDRE, UCLA (2016) bekijken. Hier zijn ook nog veel meer voorbeelden te vinden over matrix calculaties in R.)

8.6 Samenvatting

8.6.1 Matrixen genereren

In R kun je met de `matrix()` functie op een eenvoudige manier matrixen maken zoals je die wenst. Je kunt zelf de dimensies bepalen. Als een matrix gegenereerd is, lijken deze verdacht veel op data frames. De verschillen tussen deze structuren zijn daarom ook minimaal. Over het algemeen wordt er gehanteerd dat als je een tabel hebt waar veel calculaties mee gedaan moeten worden, je voor het matrix datatype kiest. Voor data opslag kies je voor een data frame. Echter kun je prima data opslaan in matrixen of calculaties doen in data frames, maar door deze vuistregel te hanteren maak je het jezelf wel makkelijker.

8.6.2 Lijsten in R

Lists zijn zeer handige datastructuren in R. Ze zijn de juiste keuze als je meerdere waarden wilt opslaan die je later weer eenvoudig terug kan roepen. Door de *key-value pair* structuur is het een veel geschikter object dan een vector met meerdere waarden. Hierdoor hoeft je niet gelijk een hele data frame te maken van een enkele rij, dat zou namelijk overdreven zijn. Zoals je in het geval van de matrix hebt kunnen zien, kun je altijd nog een matrix of dataframe maken van alle lijsten waarbij iedere lijst gewoon een record is.

8.6.3 Matrix calculaties

Matrixen zijn prima te gebruiken om calculaties op meerdere waarden te doen in een bepaalde structuur. Door de namen van de kolommen te gebruiken in matrixen, kun je zeer handig calculaties toepassen op meerdere waarden van de juiste kolom. Je kunt uiteindelijk ook matrixen met matrixen calculeren. Door dit handig in te richten, heb je een zeer krachtig rekenwapen in handen.

8.7 Oefeningen

8.7.0.0.1 Maak een matrix die bestaat uit 3 kolommen en 4 rijen en uit willekeurige nummers bevat.

8.7.0.0.2 Maak lists over de gegevens van jouw familie, en stel deze samen toe een matrix

9 Het *factor* datatype

Het factor datatype zijn we al een aantal keren tegengekomen in deze handleiding. Het lijkt misschien overdreven om een heel hoofdstuk aan een datatype toe te wijden. Echter zal je dit datatype veel tegen komen als je werkt met R. Daarbij kan je er ook zeer handig gebruik van maken en is het in sommige gevallen zelfs onmisbaar. In deze handleiding wordt het datatype factor uitgebreid uitgelegd en kun je aan de hand van demonstraties zien hoe je van het factor datatype handig gebruik kunt maken.

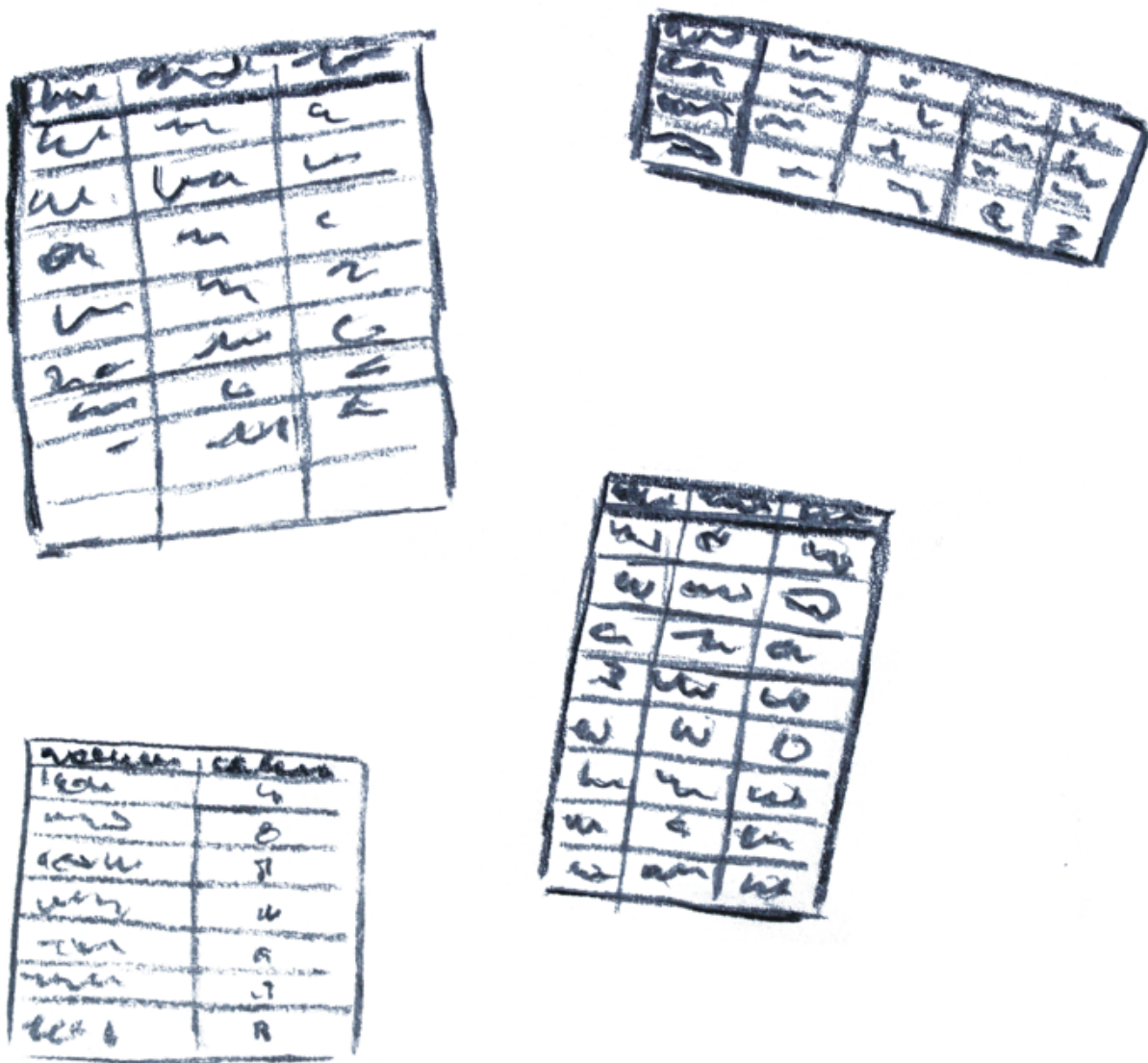


Figure 18:

9.1 Het “factor”-datatype in R

Naast de datatypes `character`, `numeric` en `boolean` is er ook het `factor` datatype. Dit datatype kun je op een eenvoudige manier beschrijven door te zeggen dat het een datatype is die qua eigenschappen tussen de `numeric` en `character` datatypes in zit. Het kan namelijk numerieke en character waarde bevatten, en er kan niet mee worden gerekend. Als we het `factor`-datatype beschrijven door vanuit de `character`-datatype te redeneren, kun je zeggen dat de `factor`-datatype een `character`-datatype is met vooraf gedefinieerde niveaus. Dit wordt aan de hand van een voorbeeld uitgelegd. Tijdens deze uitleg wordt gelijk geleerd hoe je handmatig een data frame kunt maken door vectoren samen te stellen.

Het artikel, [5] Factor variables, (Bruin, 2006) geeft een uitstekende uitleg over factor variabelen in R.

9.1.1 Handmatig data frames maken

We maken twee tabellen, in R dus `data frames`, een (1) data frame dat het salaris per persoon weergeeft en (2) een data frame die het aantal personen op kantoor weergeeft.

Om de `data frames` te maken, stellen we eerst `vectoren` op. Deze `vectoren` stellen we samen als `data frames`.

```
# Een vector maken met de namen van 10 personen
personen <- c("Jack","Rosemarijn","Arie","Marco","Koen","Noah",
             "Helga","Francis","Henk","Winston")

#personen is een character vector
class(personen)
```

```
## [1] "character"
```

```
# Een vector maken met 10 salarissen. Let op! Om de juiste salarissen bij
# de bijbehorende personen te
# houden, moeten de salarissen in dezelfde volgorde als de personen-vector
# worden gedefinieerd.

salarissen <- c(7000,2800,5000,2000,3600,5000,7000,3100,6000,2000)

#salarissen is een numeric vector
class(salarissen)
```

```
## [1] "numeric"
```

Zoals we in de output kunnen zien, hebben de vectoren `personen` en `salarissen` respectievelijk de datatypes `character` en `numeric`. Nu we de twee vectoren hebben gedefinieerd, kunnen we deze samenvoegen tot een data frame. Dit doen we eenvoudig met de `data.frame()` functie.

```
#De salarisPersonen dataframe maken
salarisPersonen <- data.frame(personen,salarissen)

salarisPersonen
```

```
##      personen salarissen
## 1      Jack      7000
## 2 Rosemarijn    2800
## 3      Arie     5000
## 4      Marco    2000
## 5      Koen     3600
## 6      Noah     5000
## 7      Helga    7000
## 8      Francis  3100
## 9      Henk     6000
## 10     Winston  2000
```

De data frame ziet er keurig uit. Laten we de datatypes van de data frame bekijken met de `class` functie. De verwachting is dat `personen` de `character` datatype heeft en `salarissen` de `numeric` datatype.

```
str(salarisPersonen)
```

```
## 'data.frame':  10 obs. of  2 variables:
## $ personen   : Factor w/ 10 levels "Arie","Francis",...: 5 9 1 7 6 8 3 2 4 10
## $ salarissen: num  7000 2800 5000 2000 3600 5000 7000 3100 6000 2000
```

Echter zien we dat `personen` in de kolom als data frame nu een `factor` is. We komen later hier op terug.

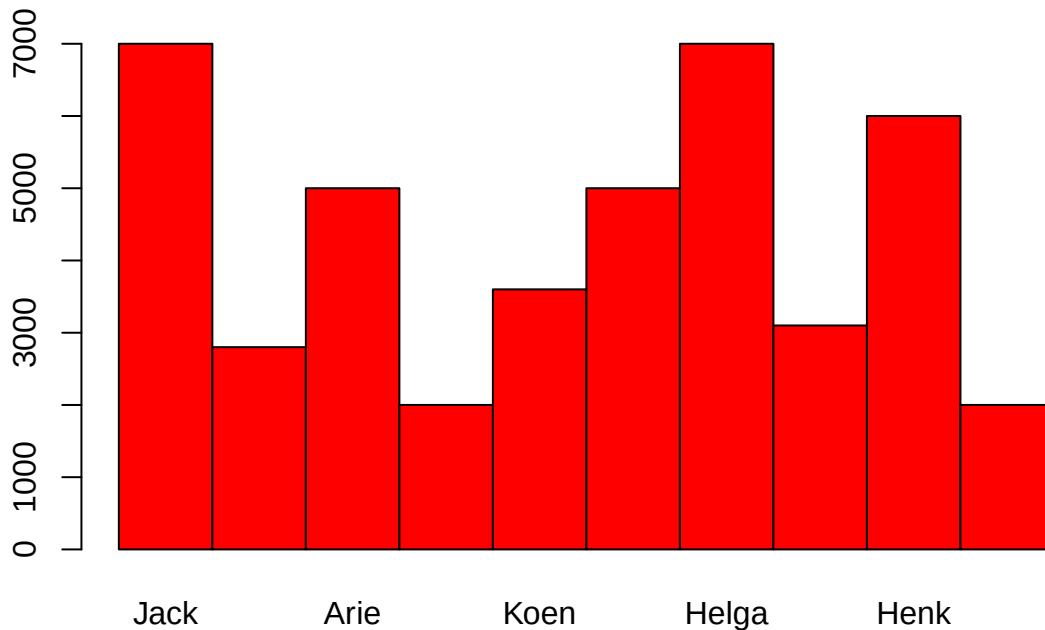
9.2 Een data frame plotten

Als we nu deze informatie in een grafiek willen laten zien, kunnen we een eenvoudige staaftdiagram maken door de personen op de x-as te plaatsen en de salarissen op de y-as. Dit doen we eenvoudig met de `barplot()` - functie.

Plot functies worden uitgebreid besproken in het hoofdstuk “Data importeren en plotten”.

```
#Een barplot maken met de volgende parameters: staven van grootte 3, geen
#ruimte tussen de staven, namen op de x-as, en de vulkleur rood.

barplot(salarisPersonen$salarissen, width = 3,space=0,
        names.arg = personen,col="red")
```



We zien dat het staafdiagram exact in dezelfde volgorde wordt weergegeven als we die hebben gedefinieerd bij de `personen` vector. Dit is waar het `factor` datatype zijn werk doet, namelijk ervoor zorgen dat er in ieder geval een vorm van volgorde aanwezig is in de data frame. Echter slaat deze volgorde nergens op, deze is gewoon door R gegenereerd. In deze situatie maakt de volgorde niet uit, dus maken we er verder geen probleem van.

9.3 Het factor-datatype definiëren

Als volgorde in een `character` datatype echter wel belangrijk is, maken we gebruik van de `factor` datatype. Een situatie waarbij dit handig is, is bijvoorbeeld namen van dagen en maanden. Laten we kijken wat de verschillen zijn als we hierbij geen gebruik maken van de `factor` datatype en wanneer we dat wel doen.

```
#character
dagen1 <- c("maandag","dinsdag","woensdag","donderdag",
            "vrijdag","zaterdag","zondag")

#factor
dagen2 <- factor(dagen1,levels=c("maandag","dinsdag","woensdag",
                                "donderdag","vrijdag","zaterdag",
                                "zondag"))

#numeric
aanwezigen <- c(9,8,6,9,3,1,0)

# De dataframe maken
bezetting <- data.frame(dagen1,dagen2,aanwezigen)

# De data frame laten zien
bezetting
```

```
##      dagen1    dagen2  aanwezigen
## 1  maandag  maandag          9
```



```
## 2   dinsdag   dinsdag           8
## 3   woensdag  woensdag          6
## 4   donderdag donderdag          9
## 5   vrijdag   vrijdag           3
## 6   zaterdag  zaterdag           1
## 7    zondag   zondag            0
```

```
# Eigenschappen van de data frame
str(bezetting)
```

```
## 'data.frame':   7 obs. of  3 variables:
## $ dagen1      : Factor w/ 7 levels "dinsdag","donderdag",...: 3 1 5 2 4 6 7
## $ dagen2      : Factor w/ 7 levels "maandag","dinsdag",...: 1 2 3 4 5 6 7
## $ aanwezig: num  9 8 6 9 3 1 0
```

We kunnen bij de laatste output al zien (achter de `str()` functie dat voor de kolom `dagen1` de volgorde er niet helemaal goed uitziet. Dit is bij `dagen2` echter wel zo. Dit komt omdat we bij `dagen2` expliciet hebben aangegeven dat we een **factor vector** willen hebben en daar duidelijk de volgorde in aangeven. Dit hebben we bij `dagen1` niet gedaan. R kiest hiervoor zelf de volgorde, die in dit geval alfabetisch is. Om dit nog duidelijker te demonstreren, laten we de twee vectoren nog een keer gesorteerd zien:

```
# de character vector 'dagen1' gesorteerd weergeven
sort(dagen1, decreasing = FALSE)
```

```
## [1] "dinsdag" "donderdag" "maandag" "vrijdag" "woensdag" "zaterdag"
## [7] "zondag"
```

```
# de character vector 'dagen2' gesorteerd weergeven
sort(dagen2, decreasing = FALSE)
```

```
## [1] maandag  dinsdag  woensdag  donderdag vrijdag  zaterdag  zondag
## Levels: maandag dinsdag woensdag donderdag vrijdag zaterdag zondag
```

Zoals je kunt zien, wordt de `dagen1` vector simpelweg weergegeven in alfabetische volgorde. De `dagen2` vector wordt echter weergegeven in de goede volgorde van de dagen in de week, zoals we dat bij de factor vector gedefinieerd hebben.

Voor de duidelijkheid geeft R ook het aantal niveaus aan in de factor vector. Dit kun je ook individueel weergeven met de `levels()` functie:

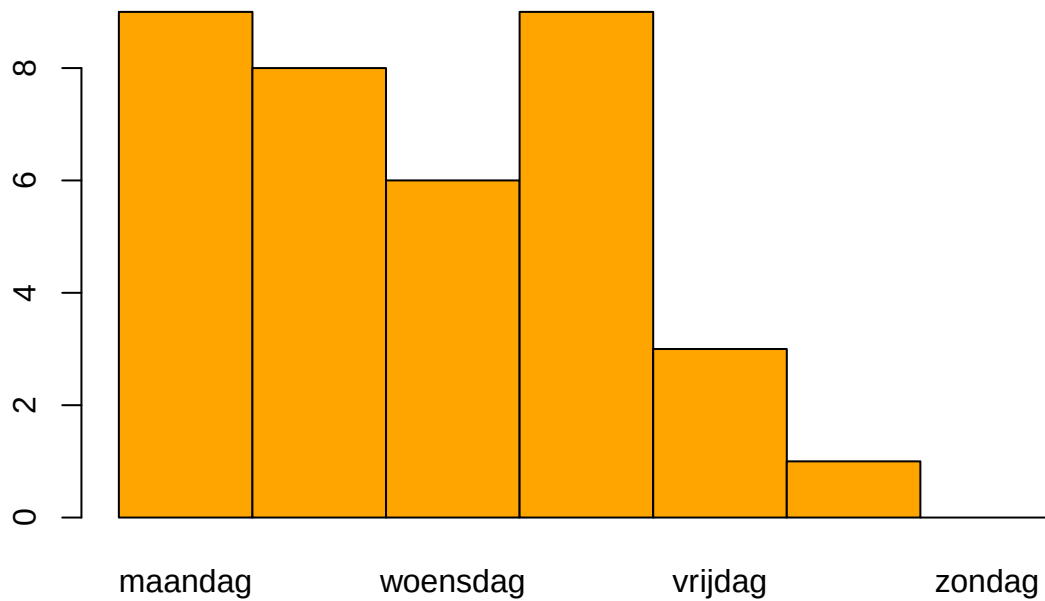
```
levels(dagen2)
```

```
## [1] "maandag" "dinsdag" "woensdag" "donderdag" "vrijdag" "zaterdag"
## [7] "zondag"
```

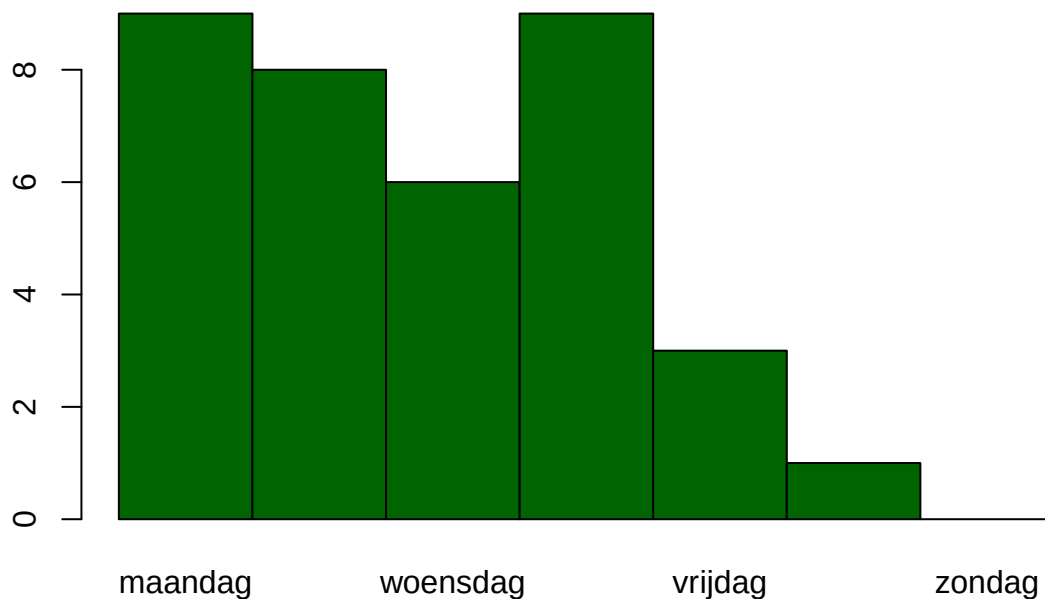
9.4 Data in de juiste volgorde plotten met factors

Het nadeel van **geen** gebruik maken van factors zien we pas echt als we grafieken maken. We maken twee visualisaties: (1) een staafdiagram die de `dagen1` vector weergeeft, dus waarvan we geen gebruik hebben gemaakt van **factors** en (2) een staafdiagram met `dagen2`, waar we wel goed gebruik hebben gemaakt van **factors**. We zullen zien dat dit voor een staafdiagram nog goed gaat:

```
#Barplot met de character vector dagen1  
barplot(bezetting$aanwezigen,width=3,space=0,names.arg = dagen1,col="orange")
```



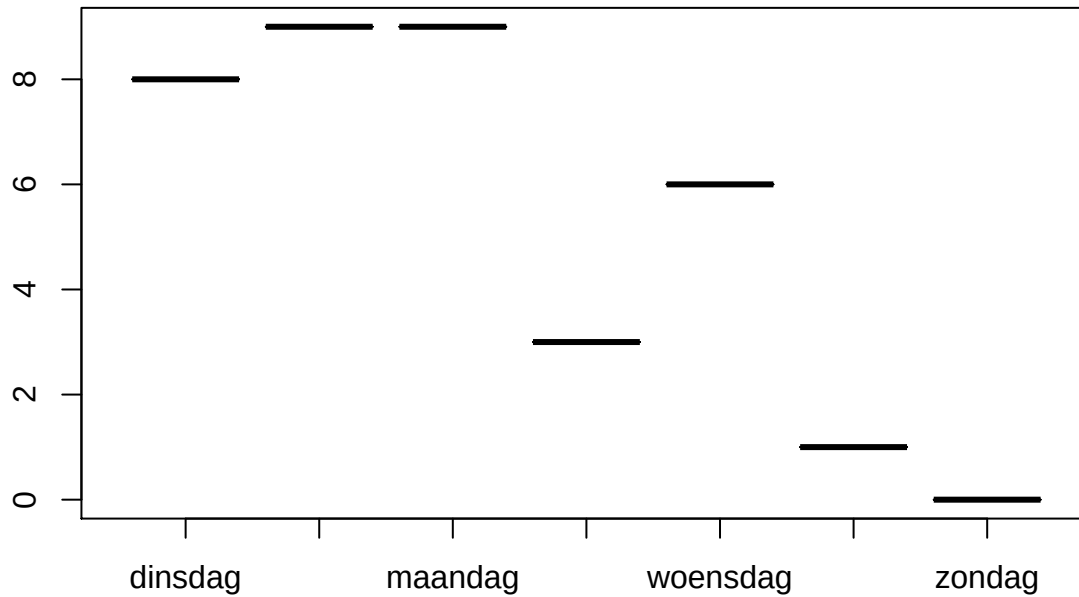
```
#Barplot met de factor vector dagen2  
barplot(bezetting$aanwezigen,width=3,space=0,names.arg = dagen2,col="darkgreen")
```



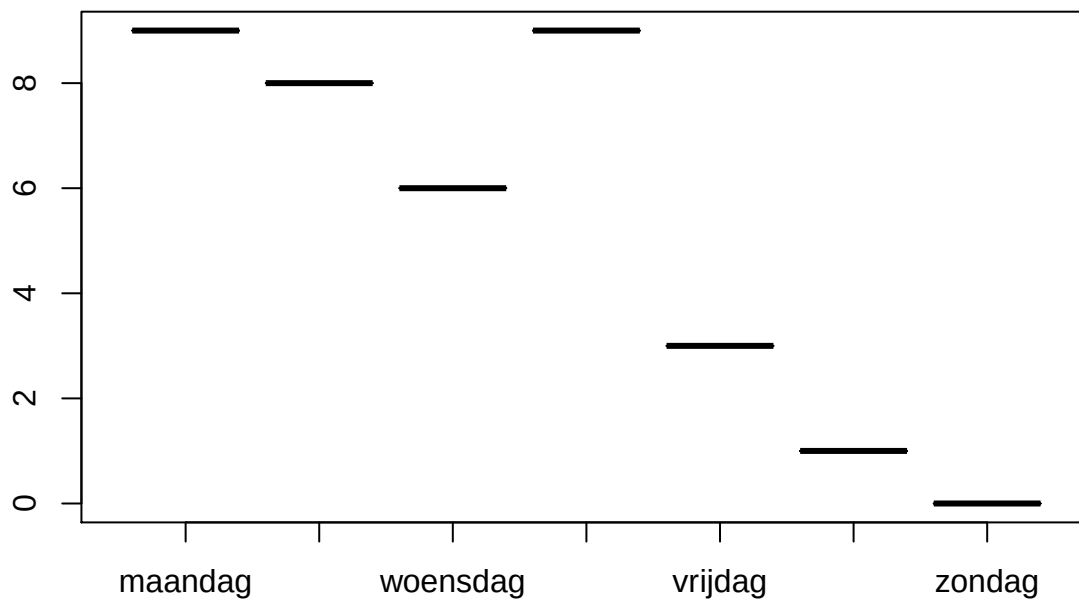
Beide grafieken zijn identiek omdat bij een `barplot` de numerieke waarde leidend is. De volgorde waarin deze numerieke waarde is gedefinieerd wordt dan gebruikt. Aangezien wij deze in de juiste volgorde hebben gedefinieerd, is er geen probleem.

Als we echter een grafiek willen maken, krijgen we wel een probleem. We doen nu hetzelfde testje voor een grafiek:

```
#Grafiek met de character vector dagen1  
plot(bezetting$dagen1, bezetting$aanwezigen, type="l")
```



```
#Grafiek met de factor vector dagen2  
plot(bezetting$dagen2, bezetting$aanwezigen, type="l")
```



Omdat we bij de eerste grafiek een `dagen1` hebben gebruikt en daardoor dus geen volgorde hebben kunnen genereren, kiest de `plot()` functie zelf een volgorde: welke in dit geval oplopend alfabetisch is en niet de goede volgorde van de dagen in de week.

9.5 Handig gebruik maken van factor vectoren

We hebben gezien dat het factor datatype in sommige gevallen handig kan zijn en soms zelfs nodig is om het gewenste resultaat te krijgen. Maak je geen zorgen, in veel gevallen volstaat de `character` datatype. Wil

je echter een volgorde of hiërarchie vastleggen in een **character** vector, maak dan gebruik van het **factor** datatype. Hiermee bespaar je veel sorteerwerk achteraf en zorg je ervoor dat je voorbewerking van de data staat als een huis.

Om nog een keer het gebruik van de **factor** datatype te laten zien, werken we nog 1 voorbeeld uit waarin we deze gebruiken. We maken opnieuw een klein werknemersbestandje, maar deze keer met de functies van de werknemers. Alhoewel ik het vrij ouderwets vind, bestaat er een bepaalde hiërarchie onder functies. We stellen deze volgorde vast door gebruik te maken van een factor en laten uiteindelijk een tabel zien waarin gesorteerd naar hiërarchie de maandsalarissen worden getoond per persoon.

```
# De namen van de werknemers aangeven
namenPersonen <- c("Jack","Rosemarijn","Arie","Marco","Koen","Noah",
                  "Helga","Francis","Henk","Winston")

# De salarissen van de werknemers definiëren
salarissenPersonen <- c(9000,2800,5000,2000,3600,5000,7000,3100,
                       6000,2000)

# De functies van de werknemers aangeven
functiesPersonen <- c("Managing partner","Secretariaat",
                    "Senior","Junior","Senior","Senior",
                    "Partner","Senior","Partner","Facilitair")

# De hiërarchie van functies aangeven
functiesPersonen2 <- factor(functiesPersonen,levels=c(
  "Managing partner","Partner","Senior","Junior","Secretariaat","Facilitair"))

werknemersbestand <- data.frame(
  namenPersonen,salarissenPersonen,functiesPersonen,functiesPersonen2)

werknemersbestand
```

##	namenPersonen	salarissenPersonen	functiesPersonen	functiesPersonen2
## 1	Jack	9000	Managing partner	Managing partner
## 2	Rosemarijn	2800	Secretariaat	Secretariaat
## 3	Arie	5000	Senior	Senior
## 4	Marco	2000	Junior	Junior
## 5	Koen	3600	Senior	Senior
## 6	Noah	5000	Senior	Senior
## 7	Helga	7000	Partner	Partner
## 8	Francis	3100	Senior	Senior
## 9	Henk	6000	Partner	Partner
## 10	Winston	2000	Facilitair	Facilitair

Zoals je kunt zien, hebben we een dataframe **werknemersbestand** aangemaakt die de namen, salarissen en functies van de personen bevat. Hierbij is **functiesPersonen** van het datatype **character** en **functiesPersonen2** van het type **factor**.

9.6 Een data frame *sorteren* rangschikken

Terminologie

Nu moeten we letten op de terminologie: onder sorteren verstaan we in de volksmond soms dat we iets aflopen of oplopend sorteren. Echter is sorteren het plaatsen van elementen in de daarvoor

gedefinieerde juiste doelen. Net zoals je als baby de juiste figuurtjes (driehoekje, blokje en rondje) in het juiste gat moest stoppen. Echter bedoelen we hier rangschikken: van hoog naar laag of van laag naar hoog.

Naast rangschikken op de salarissen kunnen we de data frame `werknemers` nu ook rangschikken op de kolom `functiesPersonen2`. Om te rangschikken in R, gebruik je de `order` functie. Aan de hand van de volgende voorbeelden zullen we zien dat rangschikken in R vrij eenvoudig is.

*In het artikel [6] **Sorting data**, (Kabacoff, 2014) wordt een handig overzicht gegeven over hoe je kunt sorteren in R.*

```
# het werknemersbestand oplopend rangschikken op salarissen
werknemersbestand[order(werknemersbestand$salarissenPersonen),]
```

```
##      namenPersonen salarissenPersonen functiesPersonen functiesPersonen2
## 4      Marco      2000      Junior      Junior
## 10     Winston      2000      Facilitair      Facilitair
## 2     Rosemarijn      2800      Secretariaat      Secretariaat
## 8      Francis      3100      Senior      Senior
## 5       Koen      3600      Senior      Senior
## 3       Arie      5000      Senior      Senior
## 6       Noah      5000      Senior      Senior
## 9       Henk      6000      Partner      Partner
## 7       Helga      7000      Partner      Partner
## 1       Jack      9000 Managing partner Managing partner
```

```
# het werknemersbestand aflopend rangschikken op salarissen
werknemersbestand[order(-werknemersbestand$salarissenPersonen),]
```

```
##      namenPersonen salarissenPersonen functiesPersonen functiesPersonen2
## 1       Jack      9000 Managing partner Managing partner
## 7       Helga      7000      Partner      Partner
## 9       Henk      6000      Partner      Partner
## 3       Arie      5000      Senior      Senior
## 6       Noah      5000      Senior      Senior
## 5       Koen      3600      Senior      Senior
## 8      Francis      3100      Senior      Senior
## 2     Rosemarijn      2800      Secretariaat      Secretariaat
## 4       Marco      2000      Junior      Junior
## 10     Winston      2000      Facilitair      Facilitair
```

```
# het werknemers bestand aflopend rangschikken op functie (zonder hierarchie)
werknemersbestand[order(-werknemersbestand$functiesPersonen),]
```

```
## Warning in Ops.factor(werknemersbestand$functiesPersonen): '-' not
## meaningful for factors
```

```
##      namenPersonen salarissenPersonen functiesPersonen functiesPersonen2
## 1       Jack      9000 Managing partner Managing partner
## 2     Rosemarijn      2800      Secretariaat      Secretariaat
## 3       Arie      5000      Senior      Senior
```

## 4	Marco	2000	Junior	Junior
## 5	Koen	3600	Senior	Senior
## 6	Noah	5000	Senior	Senior
## 7	Helga	7000	Partner	Partner
## 8	Francis	3100	Senior	Senior
## 9	Henk	6000	Partner	Partner
## 10	Winston	2000	Facilitair	Facilitair

```
# het werknemers bestand aflopend rangschikken op functie (met hierarchie)
werknemersbestand[order(werknemersbestand$functiesPersonen2),]
```

##	namenPersonen	salarissenPersonen	functiesPersonen	functiesPersonen2
## 1	Jack	9000	Managing partner	Managing partner
## 7	Helga	7000	Partner	Partner
## 9	Henk	6000	Partner	Partner
## 3	Arie	5000	Senior	Senior
## 5	Koen	3600	Senior	Senior
## 6	Noah	5000	Senior	Senior
## 8	Francis	3100	Senior	Senior
## 4	Marco	2000	Junior	Junior
## 2	Rosemarijn	2800	Secretariaat	Secretariaat
## 10	Winston	2000	Facilitair	Facilitair

Zoals je kunt zien, kunnen we sorteren op de functies in de volgorde die we voor de factor gedefinieerd hebben.

De laatste tabel is gerangschikt op de functie (de factor-versie). Als we naar de *Senior* werknemers kijken, is het misschien een goede toevoeging om binnen een bepaalde functie aan werknemers een rangschikking (aflopend) van de salarissen toe te voegen. Op dit moment zie je namelijk dat de salarissen nog niet zijn gerangschikt. Dit kunnen we eenvoudig doen door een extra argument aan de *order* functie toe te voegen. Namelijk *salarissenPersonen* als secundaire rangschikking.

```
# Het werknemers bestand aflopend rangschikken op functie (met hierarchie)
werknemersbestand[order(werknemersbestand$functiesPersonen2,
                        -werknemersbestand$salarissenPersonen),]
```

##	namenPersonen	salarissenPersonen	functiesPersonen	functiesPersonen2
## 1	Jack	9000	Managing partner	Managing partner
## 7	Helga	7000	Partner	Partner
## 9	Henk	6000	Partner	Partner
## 3	Arie	5000	Senior	Senior
## 6	Noah	5000	Senior	Senior
## 5	Koen	3600	Senior	Senior
## 8	Francis	3100	Senior	Senior
## 4	Marco	2000	Junior	Junior
## 2	Rosemarijn	2800	Secretariaat	Secretariaat
## 10	Winston	2000	Facilitair	Facilitair

Het eindresultaat is dat de werknemers op functies zijn gerangschikt, en binnen de functies op hun salaris zijn gerangschikt.

9.7 Samenvatting

9.7.1 Handmatig data frames maken

In de vorige hoofdstukken heb je kunnen zien hoe je handmatig een matrix kunt maken. Op dezelfde handige manier kun je ook een data frame handmatig samenstellen. Je kunt hiervoor de waarden stuk voor stuk invoeren, of de data gebruiken uit bestaande vectoren.

9.7.2 De extra eigenschap van het *factor* datatype

Het *factor* datatype is eigenlijk een *character* datatype met een extra eigenschap, namelijk de verschillende niveaus (*de levels*). Je moet in een R-sessie daarom voor jezelf goed beslissen of *character* vectoren echt als *characters* behandeld moeten worden of als je toch liever handig gebruik wilt maken van de eigenschap van het *factor* datatype. Het *factor* datatype gebruik je als je verschillende niveaus voor een *character* waarde wilt definiëren, zoals dagen, maanden, functie enzovoort. Als dit niet zo is, kun je beter bij het *character* datatype blijven.

9.7.3 Rangschikken

Voor een goed overzicht, is het in sommige gevallen gewenst dat de data gerangschikt is. Met een *numeric* datatype is dit eenvoudig te doen, het is namelijk zeer eenvoudig en vrij logisch om numerieke waarde te rangschikken. Met *character* waarden is dit natuurlijk een ander verhaal. R weet namelijk niet uit zichzelf welke logische volgorde er in een *character* vector zit. Hierbij maak je handig gebruik van het *factor* datatype. Het voorbeeld in dit hoofdstuk laat zien dat het sorteren op een *factor* datatype net zo eenvoudig gaat als bij een *numeric* datatype.

9.8 Oefeningen

9.8.1 Maak een gesorteerde *factor* van de maanden in het jaar en sorteer deze op de juiste manier.

9.8.2 Maak een dataframe die uit twee kolommen bestaat, *omzet*: bevat numerieke waarden over een denkbeeldige omzet per maand, en *maand*: een *factor* vector die de maanden van het jaar bevat (die in de vorige oefening is gemaakt).

9.8.3 Sorteer de data frame op de maanden oplopend. Maak daarbij ook een lijndiagram die de omzet per maand weergeeft.

10 Eenvoudige grafieken maken in R

In sommige gevallen is het handig om data te visualiseren in grafieken. R biedt uitstekende mogelijkheden om visualisaties te creëren. Zo goed als ieder denkbare grafiek is te maken met R, van simpele grafieken tot meer uitgebreide visualisaties. In dit hoofdstuk wordt behandeld hoe je in R eenvoudige grafieken kunt maken.

Data visualiseren kan op de meest uitgebreide manieren. De `plot()` -functie is slechts een enkele van de mogelijke visualisatie functies in R. Uitgebreid data visualiseren kan bijvoorbeeld ook met `ggplot`. Dit wordt behandeld in hoofdstuk 9.

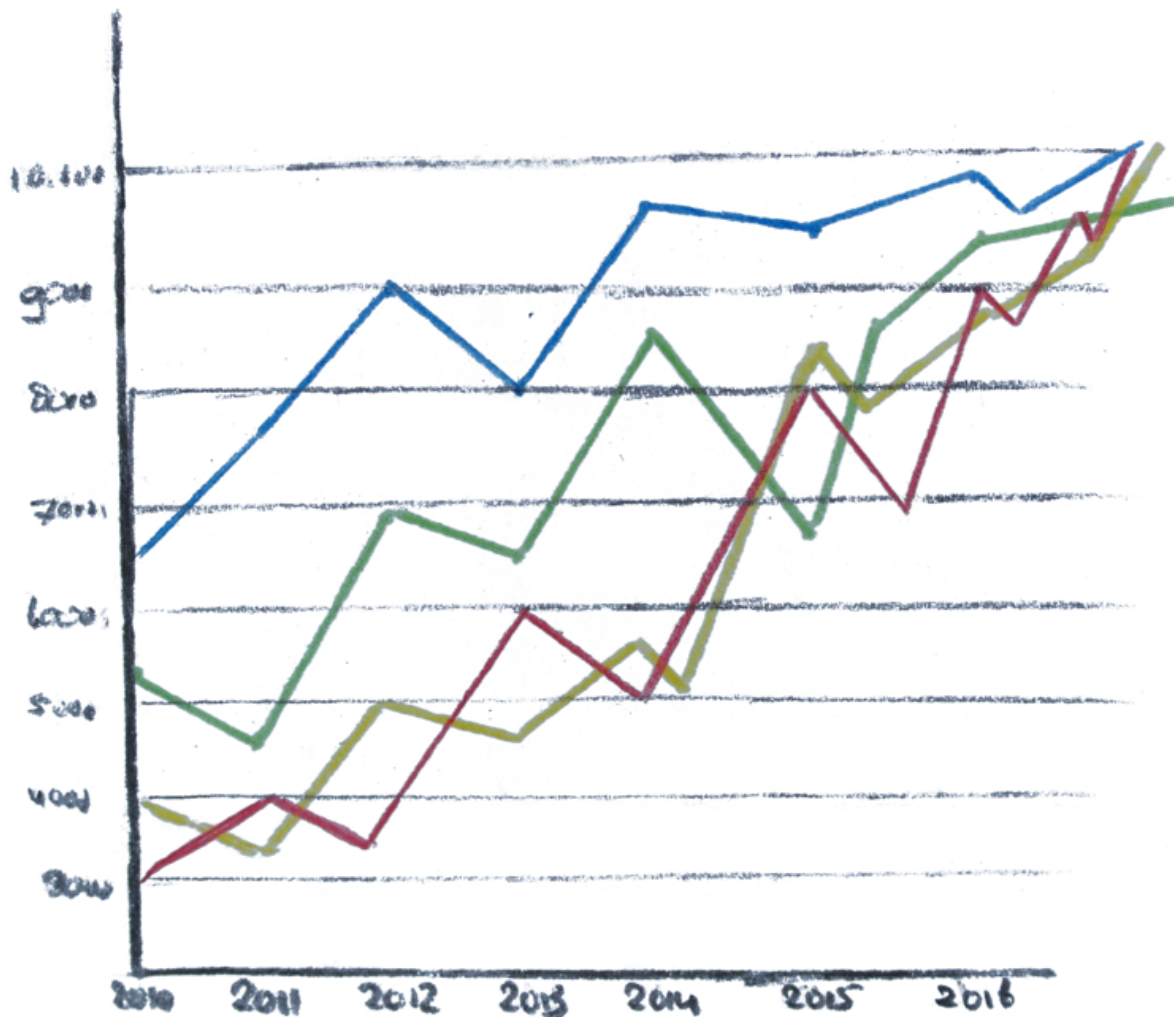


Figure 19:

10.1 De data frame voorbereiden

Door de data goed te controleren en voor te bereiden, zorgen we ervoor dat de visualisaties ook van een goede kwaliteit zijn en geen fouten bevatten. Met dat voorbereiden kun je denken aan:

- Controleren of de waarden kloppen;
- Lege waarden veranderen;
- De datatypes van de grafiek controleren.

10.1.1 De `attach()` functie gebruiken

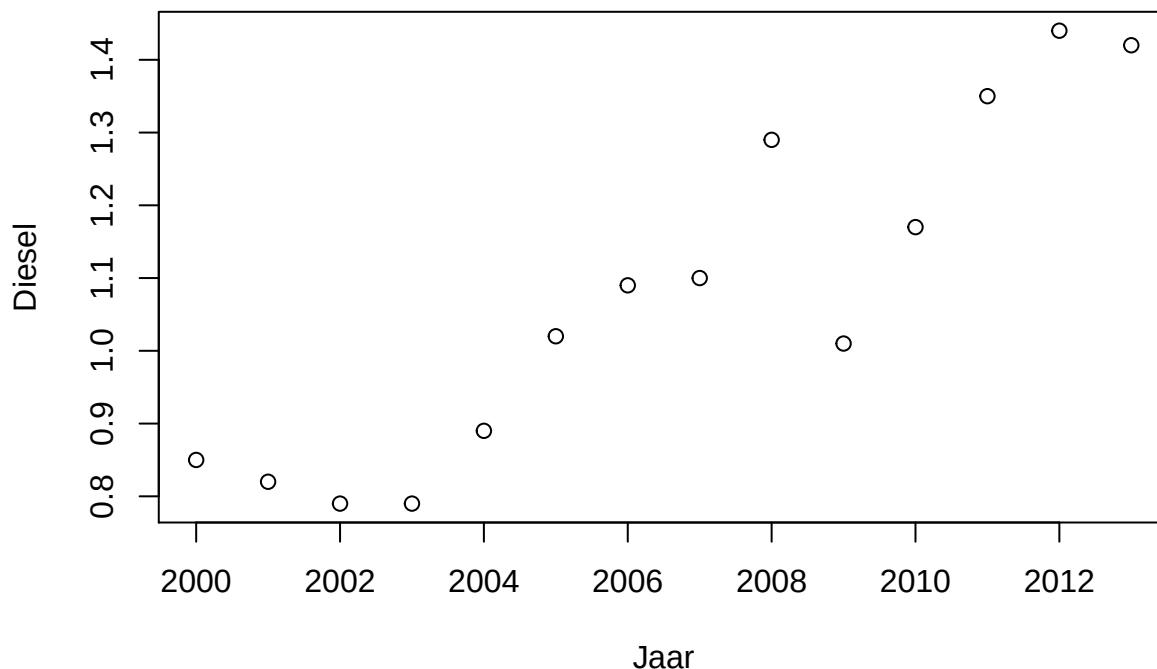
Door de `attach()` functie te gebruiken op een data frame, kunnen we voor de plot functies direct de namen van de variabelen noemen die we gevisualiseerd willen hebben.

Als je al eerder tijdens een sessie de `attach()` functie gebruikt hebt, hoeft je dat niet opnieuw te doen. Mocht het nodig zijn, kun je met de `detach()` functie een eerdere toepassing van de `attach()` functie ongedaan maken.

10.2 Een eenvoudige grafiek maken in R

Met de `plot()` functie kun je aangeven welke variabele je op de x-as en de y-as wilt hebben. Als dat de enige parameters zijn die je invoert, wordt er een eenvoudige grafiek gemaakt. Deze wordt in R een *scatter plot* genoemd.

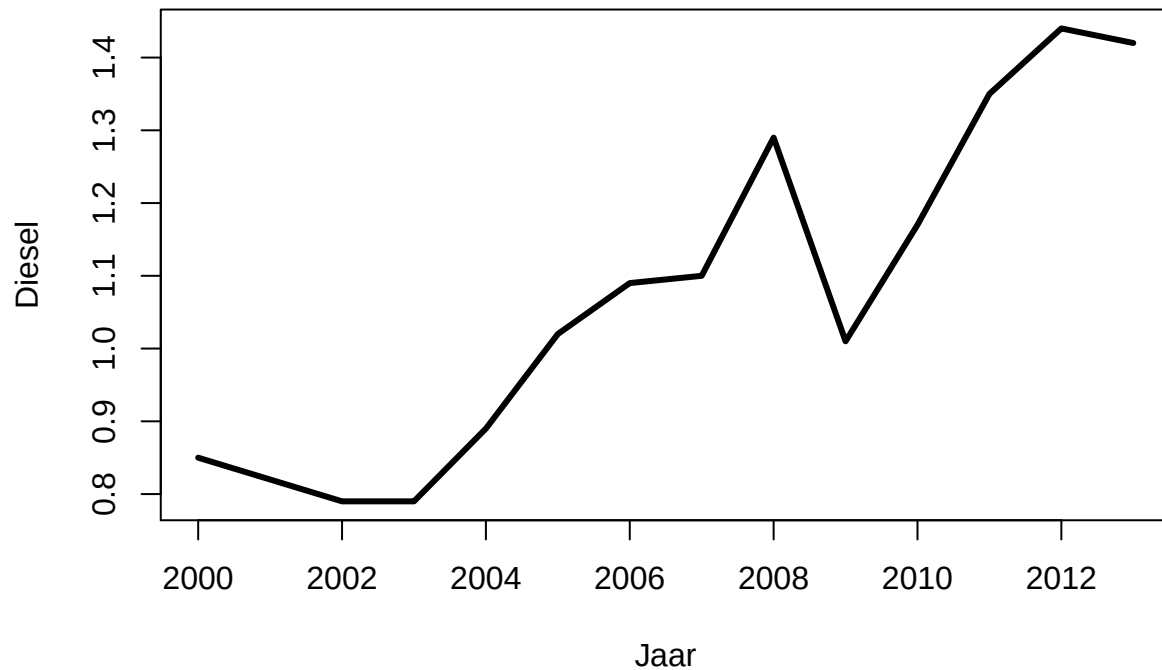
```
plot(Jaar,Diesel)
```



10.2.1 Het type van de grafiek in de `plot()` functie veranderen

Je kunt het type van de grafiek veranderen met de `type` en de `lwd` argumenten. Hierbij kun je met `type` aangeven waarmee je de grafiek wilt visualiseren en `lwd` hoe dik de lijn of de punten moeten zijn. Met het onderstaande command geven we aan dat we een lijn willen hebben met een dikte van 3.

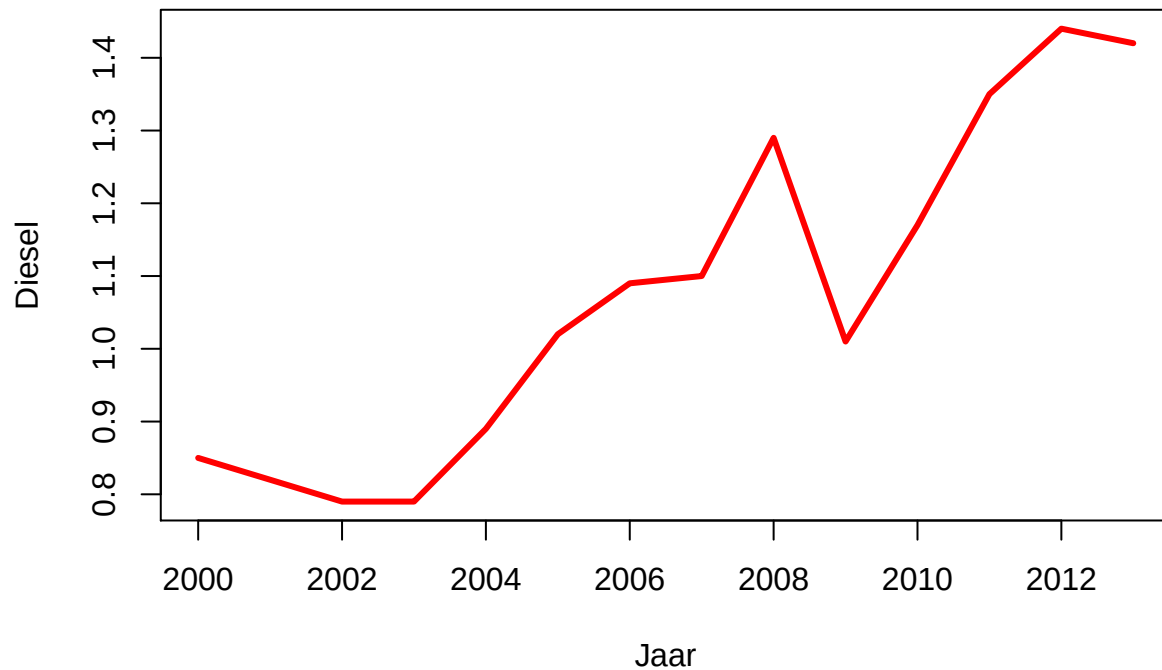
```
plot(Jaar,Diesel,type="l",lwd=3)
```



10.2.2 De kleur van de lijn veranderen

We kunnen een lijngrafiek maken in een gewenste kleur en een gewenste dikte als we de extra argumenten `type`, `lwd` en `col` toevoegen in de `plot()` functie.

```
# een rode lijndiagram maken  
plot(Jaar,Diesel,type="l",lwd=3,col="red")
```

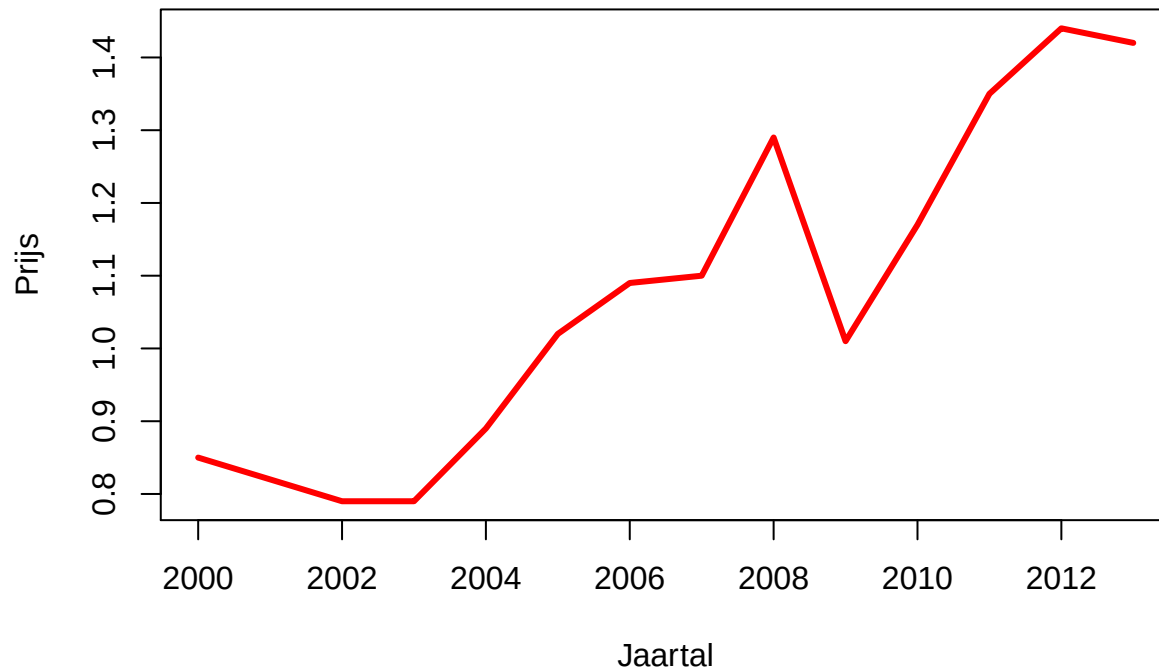


10.2.3 De titels van de x-as, y-as en de hoofdtitel veranderen

Met de `xlab`, `ylab` en `main` argumenten, kunnen we respectievelijk de titels van de x-as, y-as en hoofdtitel aangeven.

```
plot(Jaar,Diesel,type="l",lwd=3,col="red",  
     xlab="Jaartal",ylab="Prijs",  
     main="Ontwikkeling brandstofprijzen")
```

Ontwikkeling brandstofprijzen

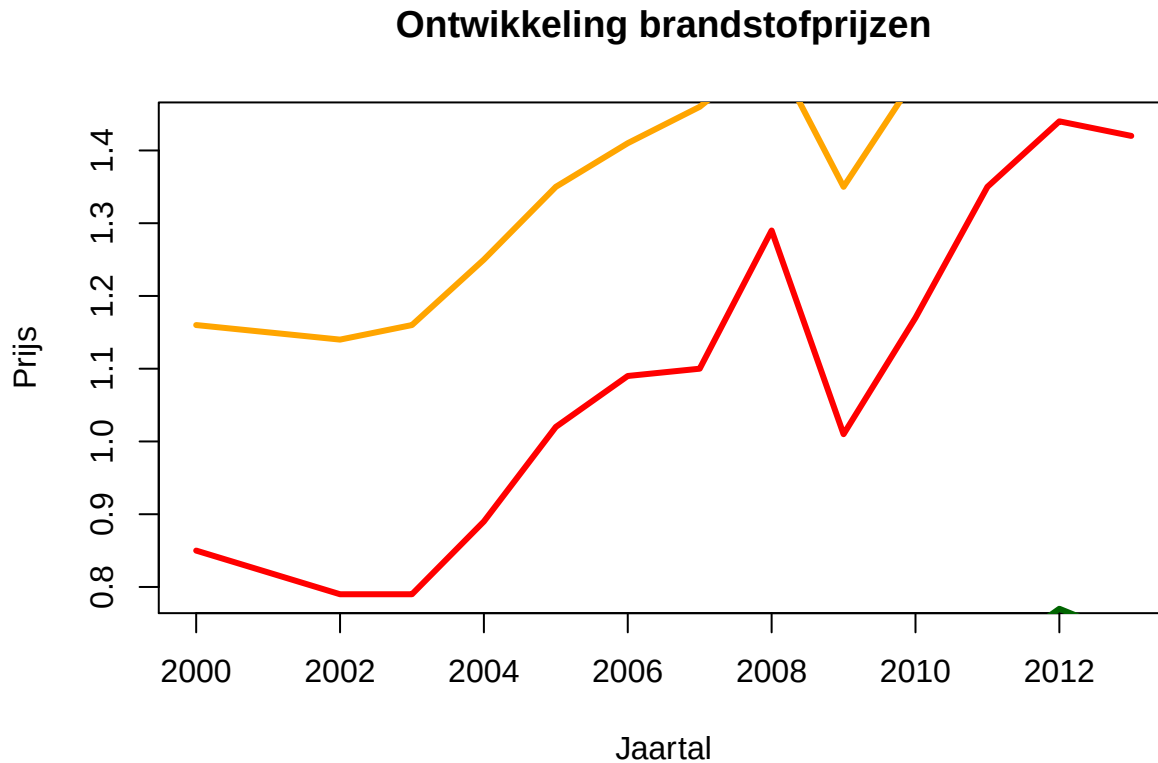


10.2.4 Lijnen van andere kolommen in de plot weergeven

Soms wil je meerdere lijnen in dezelfde grafiek weergeven, bijvoorbeeld om een vergelijking te maken. In dit geval vergelijken we de prijzen van Benzine, Diesel en LPG.

```
plot(Jaar,Diesel,
     type="l",
     lwd=3,
     col="red",
     xlab="Jaartal",
     ylab="Prijs",
     main="Ontwikkeling brandstofprijzen")

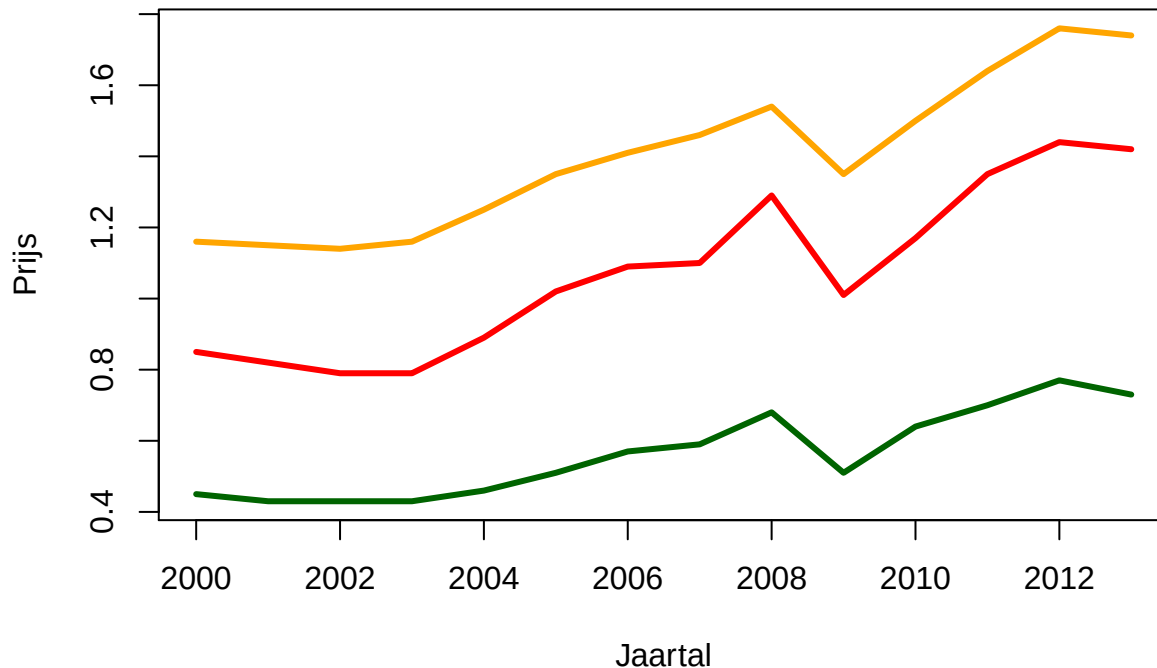
lines(Jaar,Benzine.Euro95,type="l",lwd=3,col="orange")
lines(Jaar,LPG,type="l",lwd=3,col="darkgreen")
```



Zoals je kunt zien, passen de lijnen van Benzine (de prijzen zijn te hoog) en de prijs van LPG (de prijzen zijn te laag) niet in de grafiek. Daarvoor moeten we de assen aanpassen. We kunnen dit slim oplossen. Voor de y-as stellen we namelijk als minimum de uitkomst van de `min()` functie op LPG en de `max()` functie op Benzine als instellingen van de y-as. Uiteindelijk kunnen we het gewenste resultaat met het volgende command samenstellen.

```
plot(Jaar,Diesel,type="l",lwd=3,
     col="red",xlab="Jaartal",ylab="Prijs",
     main="Ontwikkeling brandstofprijzen",ylim = c(min(LPG),max(Benzine.Euro95)))
lines(Jaar,Benzine.Euro95,type="l",lwd=3,col="orange")
lines(Jaar,LPG,type="l",lwd=3,col="darkgreen")
```

Ontwikkeling brandstofprijzen



10.3 Staafdiagrammen en histogrammen

Met R kun je ook staafdiagrammen en histogrammen maken. Deze visualisaties zijn handig als je de hoogtes en aantallen van bepaalde categorieën wilt visualiseren. Om dit te demonstreren, maken we handmatig een data frame aan met een aantal categorieën en waarden zodat we deze in de grafiek kunnen visualiseren.

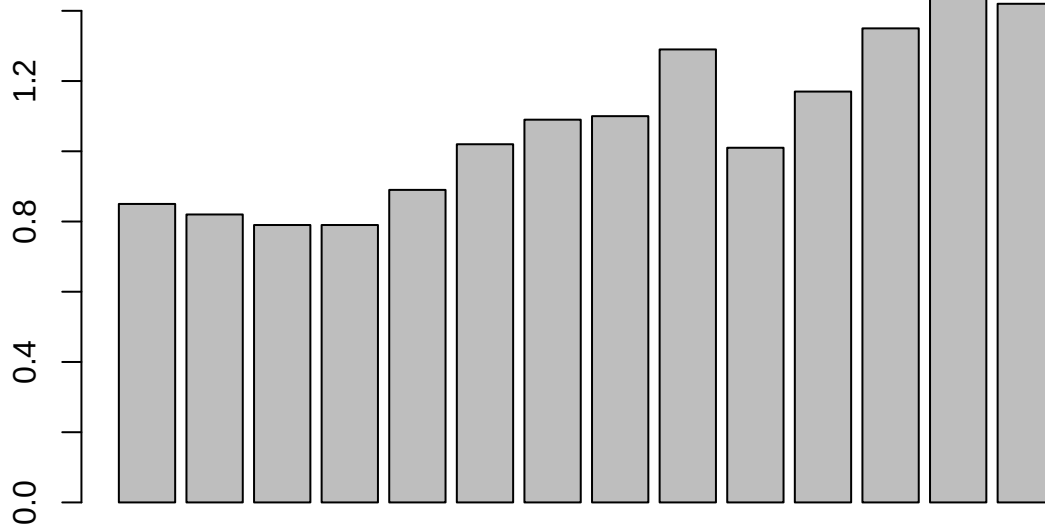
10.3.1 Een staafdiagram maken met de `barplot()` functie

De `barplot()` functie werkt op een vergelijkbare manier als de `plot` functie. Echter werken de parameters iets anders.

Om de parameters van een functie te bekijken, kun je de `help()` functie gebruiken.

Als we een enkel argument geven aan de `barplot()` functie, de data die we willen visualiseren, geeft het een standaard opmaak van het staafdiagram.

```
# Een staafdiagram maken met de prijzen van Diesel  
barplot(Diesel)
```

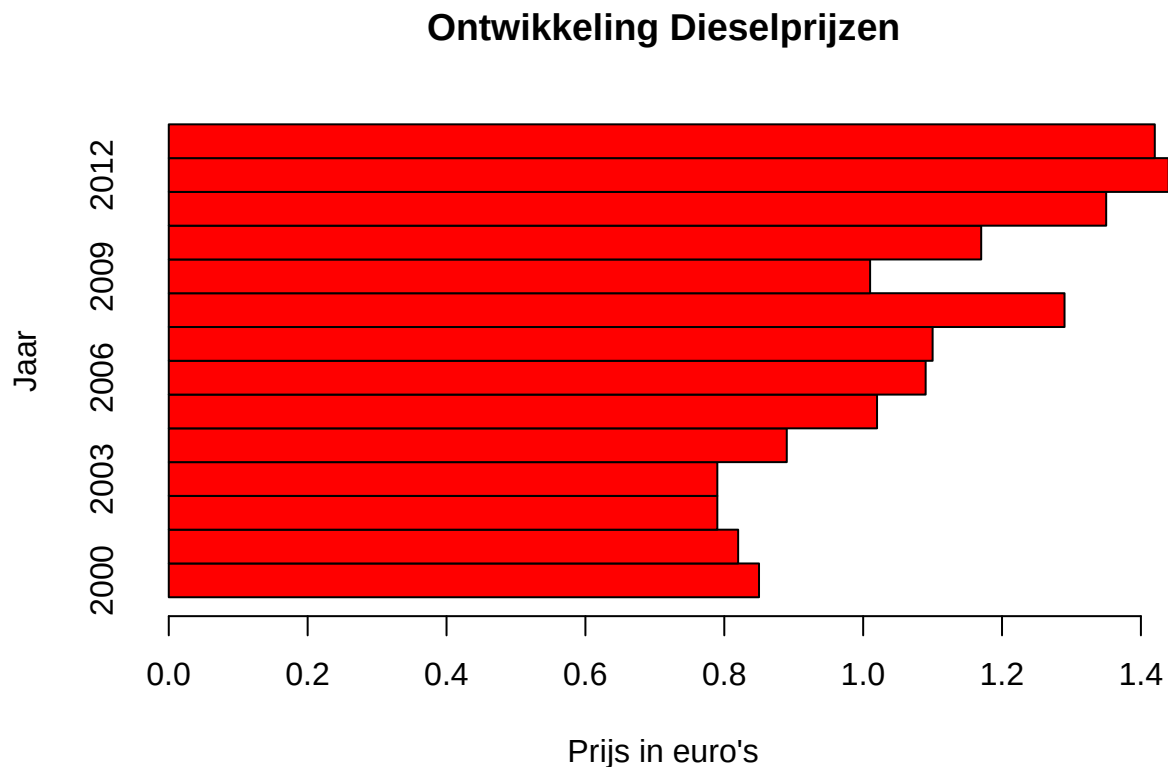


Zoals je kunt zien, wordt op volgorde van jaar de hoogte van de dieselprijzen weergegeven.

10.3.2 Extra parameters voor het staafdiagram

We gaan het staafdiagram verbeteren door er een aantal parameters aan toe te voegen:

```
barplot(Diesel, width = 3,
        space = 0,
        names.arg = Jaar, horiz = TRUE, col = "red",
        main = "Ontwikkeling Dieselprijzen",
        xlab = "Prijs in euro's", ylab = "Jaar")
```



Zoals je kunt zien, kun je door het gebruik van een aantal parameters de standaardopmaak van R opleuken. Voor een overzicht van de beschikbare kleuren in R, kun je de `colors()` functie gebruiken voor een overzicht.

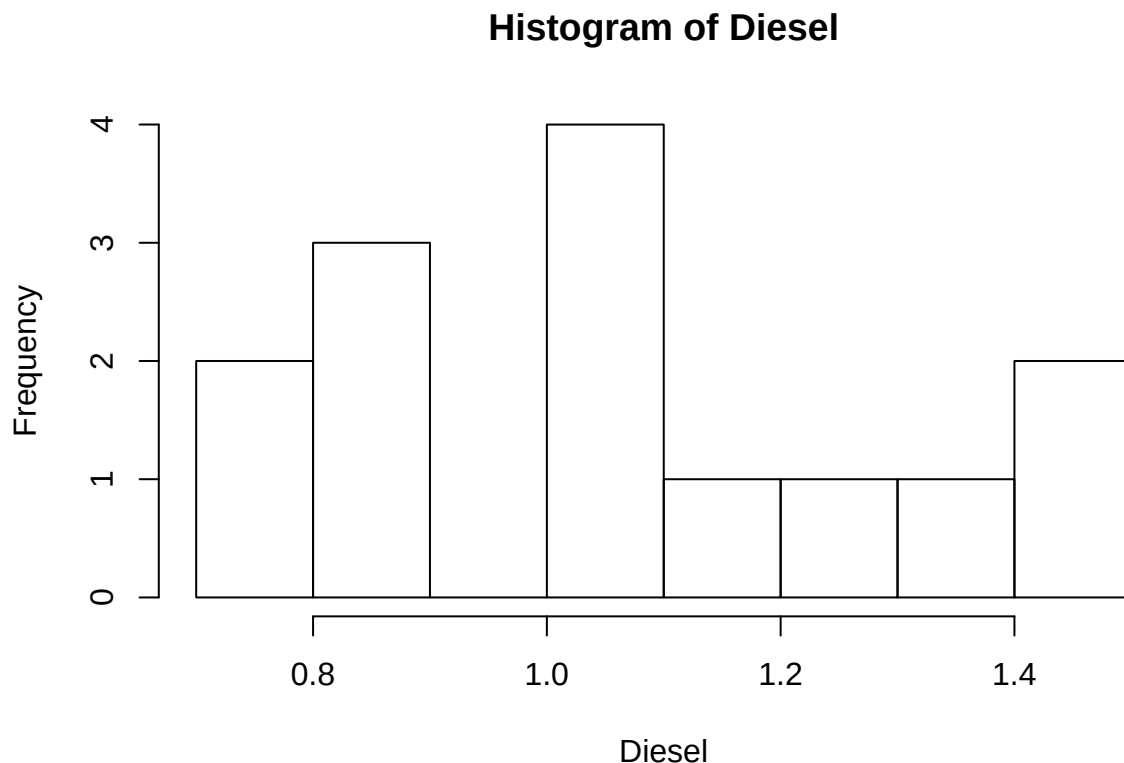
```
# De eerste 7 beschikbare kleuren in R weergeven  
head(colors())
```

```
## [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"  
## [5] "antiquewhite2" "antiquewhite3"
```

10.3.3 Een histogram maken met de `hist()` functie

Met een histogram wordt de frequentie van een bepaalde waarde gevisualiseerd. Vooral in de statistiek wereld is het een handige visualisatie om de verdeling van de waarden weer te geven. Met de `hist()` functie kun je in R een histogram visualiseren.

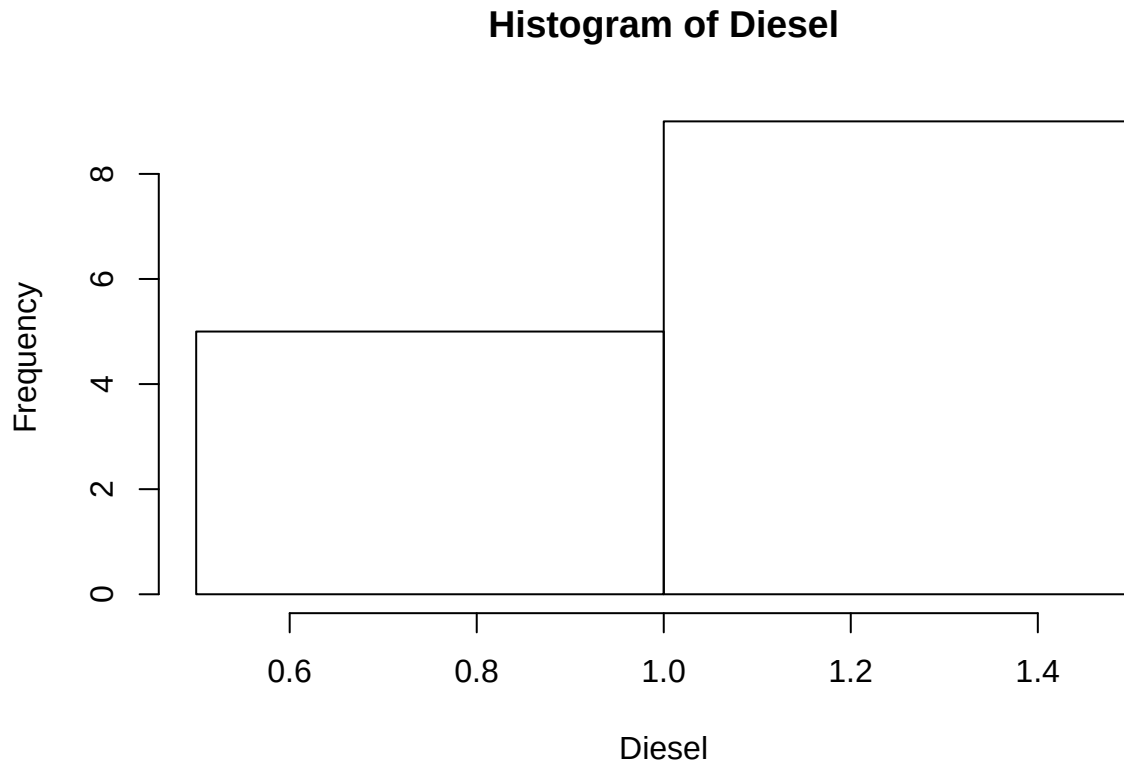
```
hist(Diesel)
```



In het histogram is o.a. af te lezen dat een prijs onder de 0.8 2 keer voor komt. Ook kun je aflezen dat een prijs tussen 1 en 1.1, (tussen de euro en 1.10 euro) 4 keer voor komt.

Als we een grovere verdeling willen maken, kunnen we een extra parameter toevoegen aan de `hist` functie. We gebruiken hier het `breaks` argument voor.

```
hist(Diesel, breaks = 2)
```

In de histogram is goed af te lezen dat prijzen onder de één euro 5 keer voor komen en boven de één euro 9 keer voor komen.

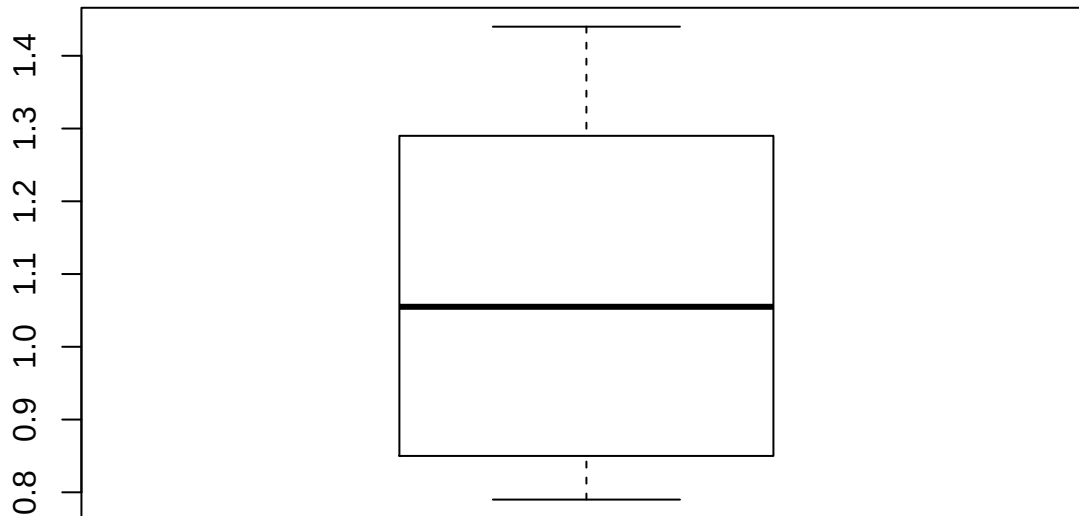
10.4 Een boxplot maken in R

Een boxplot is een andere visualisatie die in de statistiek ook veel wordt gebruikt en op een uitstekende manier duidelijke inzichten geeft over de waarden. Het visualiseert namelijk dezelfde statistieken die je krijgt bij het uitvoeren van het `summary` command, op het gemiddelde na. Dit zijn de `minimum`, `maximum`, `mediaan`, `1st Qu.` en `3rd Qu.` (eerste en derde kwartiel).

```
summary(Diesel)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.790   0.860   1.055   1.074   1.260   1.440
```

```
boxplot(Diesel)
```

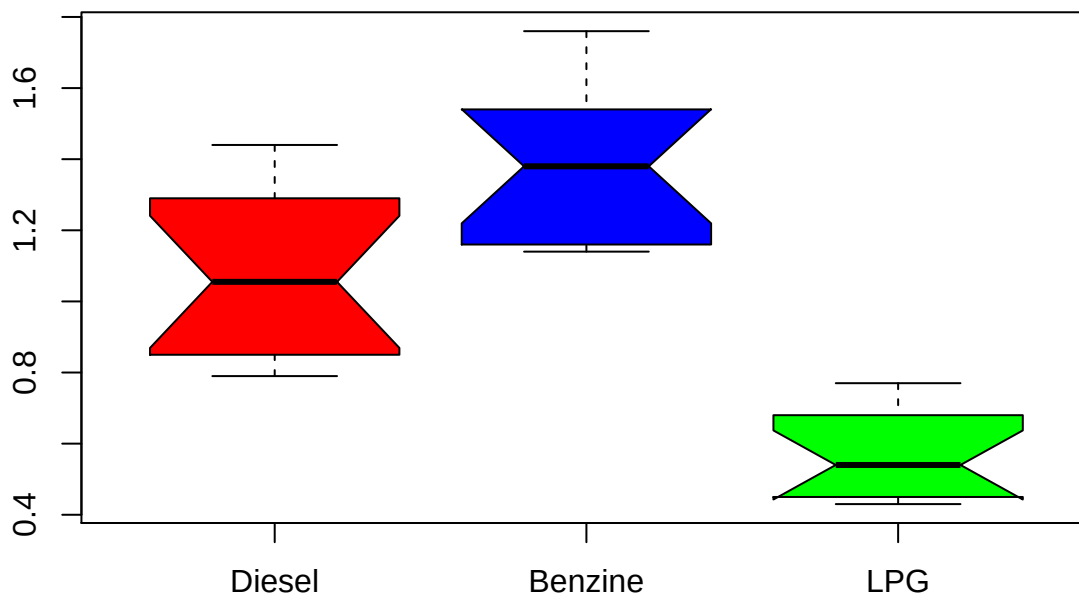


Door middel van een boxplot heb je deze informatie visueel en krijg je een korte maar effectieve statistische samenvatting van de data.

Het is ook mogelijk om meerdere boxplots te visualiseren in dezelfde visualisatie. Hiervoor voer je eenvoudig de naam van een andere variabele toe aan de functie. Om dit te demonstreren, visualiseren we **Diesel**, **LPG** en **Benzine** in dezelfde grafiek. Daarbij geven we een aantal andere handige parameters aan de `boxplot()` functie, deze worden hierna ook toegelicht.

```
boxplot(Diesel, Benzine.Euro95, LPG, notch = TRUE,
        col = c("red", "blue", "green"),
        names = c("Diesel", "Benzine", "LPG"))
```

```
## Warning in bxp(structure(list(stats = structure(c(0.79, 0.85, 1.055,
## 1.29, : some notches went outside hinges ('box'): maybe set notch=FALSE
```



Zoals je kunt zien, kun je door middel van extra parameters de standaard boxplot uitbreiden. In deze visualisatie kun je zien hoe de prijzen van de drie brandstoffen uit de dataset zich onderling verhouden. Zo kun je bijvoorbeeld zien dat het maximum van de prijs van LPG nog niet eens bij het minimum van de prijs

van **Diesel** komt. De inhammen die je ziet, worden *notches* genoemd. Volgens een paper van Chambers et al. (1983) (een statisticus waar van ik het citaat heb uit het onderstaande artikel) zijn de medianen van de variabelen significant verschillend als de *notches* van deze variabelen elkaar niet overlappen. In dit geval is het net aan het geval bij **Diesel** en **Benzine**: Je kunt zien dat de inhammen, dus de *notches* van **Diesel** en **Benzine** elkaar nog net overlappen.

*In het artikel [7] **Exploratory Data Analysis: Variations of Box Plots in R for Ozone Concentrations in New York City and Ozonopolis**, (Eric Cai - RBloggers, 2013) wordt uitgebreid uitgelegd hoe je boxplots kunt gebruiken en interpreteren voor data exploratie.*

10.5 Samenvatting

10.5.1 De attach functie gebruiken voor een data frame

Door de `attach()` functie te gebruiken, kun je zeer eenvoudig de kolomnamen (variabelen) uit een data frame oproepen. Je hoeft namelijk niet meer de `$` operator te gebruiken om eerst de naam van de data frame te selecteren. De data frame heb je namelijk al genoemd in de `attach()` functie. Met de `detach()` functie kun je het kiezen van de data frame weer ongedaan maken en moet je weer gebruik maken van de `$` operator.

Veel R gebruikers raden het gebruik van `attach()` af, waaronder ik zelf. Het is natuurlijk zo dat het toepassen van `attach()` op een data frame veel dingen makkelijker maakt. Echter kan het ervoor zorgen dat de R-sessie onoverzichtelijk wordt, je kan namelijk vergeten bij welke data frame de kolomnamen horen. Daarbij hebben normaal gesproken alleen objecten zoals een data frame, matrix of vector een naam en kolommen in een data frame niet. Je speelt op deze manier een beetje met de geordende structuur in R. Echter is de `attach()` functie er niet voor niks en is het slechts mijn mening. Voel je absoluut niet geremd om de `attach()` functie te gebruiken.

10.5.2 Grafieken maken in R

Data visualiseren is een van de sterkste wapens in R. Met zeer weinig code, en dus in heel weinig tijd, kun je een hele mooie en handige grafiek opbouwen. Met functies als `plot()`, `barplot()` en `boxplot` kun de visualisatie kiezen die bij jouw situatie past. Een grafiek heeft een vector als input. Dit kan een losse vector zijn of een kolom uit een data frame, dat tenslotte ook een vector is. Het maakt meestal voor R niet uit van welk datatype een vector is, R doet zijn best om er een grafiek van te maken. Als het echt niet kan, krijg je een foutmelding.

10.5.3 Een visualisatie volledig vormgeven naar jouw eigen voorkeur

Je hebt kunnen zien dat als we bijvoorbeeld alleen de namen van de vectoren in de `plot()` functie stoppen, R een kale en minimale versie van de grafiek genereert. Dit is erg prettig omdat je in het begin niet verplicht bent om veel argumenten te gebruiken om een visualisatie te maken. Aan de andere kant heb je juist de mogelijkheden om een visualisatie helemaal naar jouw eigen wensen op te bouwen. Op deze manier kun je langzamerhand van kale grafiek naar uitgebreide en prachtige visualisatie gaan die alles bevat wat je zelf wilt.

10.6 Oefeningen

10.6.0.0.1 Gebruik de `mtcars` dataset om de verhouding tussen paardenkracht (`hp`) en gewicht (`wt`) te visualiseren in een grafiek (je mag er zelf voor kiezen of je gebruikt maakt van de `attach()` functie of niet).

10.6.0.0.2 Maak een donkerblauwe histogram die horizontaal is weergegeven en informatie bevat over het aantal auto's gesorteerd per aantal versnellingen.

10.6.0.0.3 Maak dezelfde visualisatie als in de eerste opgave. Echter deze keer helemaal uitgewerkt met de volgende onderdelen: Een titel op de x-as, een titel op de y-as, een titel van de grafiek, een leuke vorm en een mooie kleur. Gebruik je creativiteit.

11 Case 1: Transactiedata analyseren

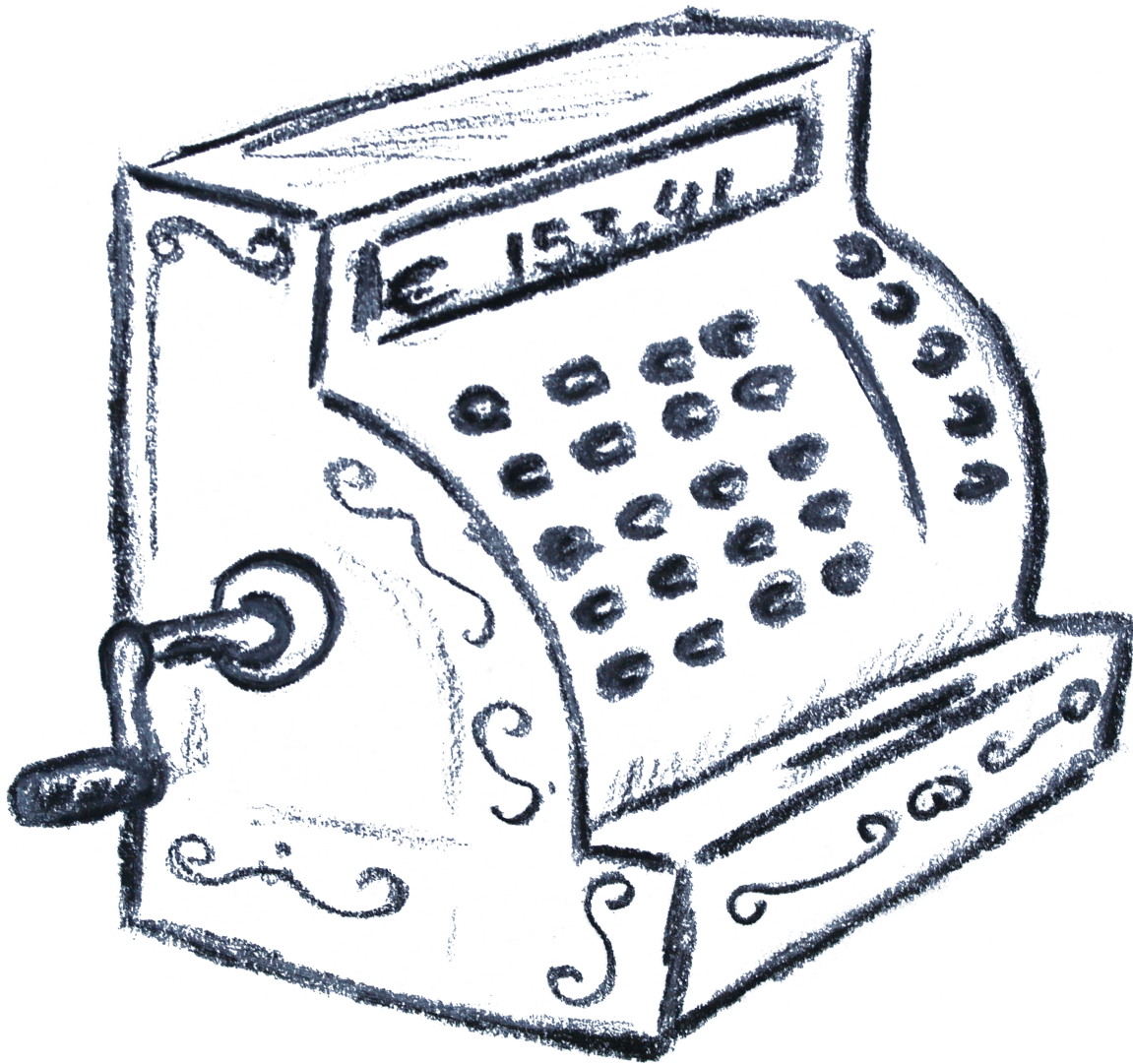


Figure 20:

In dit hoofdstuk wordt een case behandeld. In dit geval een situatie waarin je een set aan transactiegegevens tot je beschikking hebt en hier inzichten uit probeert te halen. Aan de hand van voorbeelden word je geholpen om een aantal inzichten uit deze dataset te halen. Voel je vrij om af te wijken van de voorbeelden in deze case en je creativiteit de vrije loop te laten gaan.

In deze case wordt de dataset CDNOW geanalyseerd. Dit is rauwe transactiedata uit een database.

De huidige dataset, CDNOW.csv komt van de website <http://brucehardie.com/datasets/> . Dit bestand is speciaal beschikbaar gesteld voor experimenten met data analyse en stamt uit de jaren 90. Deze dataset wordt voornamelijk in wetenschappelijke artikelen gebruikt. De dataset

bestaat uit transactiedata van aankopen van verschillende klanten. De eerste kolom geeft het klantnummer aan, de tweede kolom de datum van de transactie, de derde kolom het aantal producten dat gekocht is en de vierde kolom het totale bedrag dat is afgerekend. Succes!

11.1 De CDNOW-dataset importeren

We importeren de CDNOW dataset. Als de dataset is gedownload, kunnen we de *Import Dataset* knop van RStudio gebruiken.

Import Dataset

Name:

Encoding:

Heading: ☐ Yes ☒ No

Row names:

Separator:

Decimal:

Quote:

Comment:

na.strings:

☒ Strings as factors

Input File

00001	19970101	1	11.77
00002	19970112	1	12.00
00002	19970112	5	77.00
00003	19970102	2	20.76
00003	19970330	2	20.76
00003	19970402	2	19.54
00003	19971115	5	57.45
00003	19971125	4	20.96
00003	19980528	1	16.99
00004	19970101	2	29.33
00004	19970118	2	29.73
00004	19970802	1	14.96
00004	19971212	2	26.48
00005	19970101	2	29.33

Data Frame

V1	V2	V3	V4
1	19970101	1	11.77
2	19970112	1	12.00
2	19970112	5	77.00
3	19970102	2	20.76
3	19970330	2	20.76
3	19970402	2	19.54
3	19971115	5	57.45
3	19971125	4	20.96
3	19980528	1	16.99
4	19970101	2	29.33
4	19970118	2	29.73
4	19970802	1	14.96
4	19971212	2	26.48

Figure 21: Import CDNOW

Merk op dat de dataset geen kolomnamen heeft. In dit geval kiezen we daarom voor *No* bij *Headings*. De kolomnamen moet we later dus handmatig toevoegen. De volgende code wordt door het importeren automatisch door RStudio gegenereerd.

De CDNOW dataset is nu een **data frame** in R. Het is een slim idee om deze dataset te kopiëren door een nieuw object te maken met dezelfde dataset. Hierdoor hebben we altijd nog de originele dataset voor het geval we het verpesten tijdens onze analyse. We noemen het object waar we mee verder gaan werken `CDNOW.df`.

```
CDNOW.df <- CDMOW
```

11.2 De eigenschappen van de dataset bekijken

Als volgende stap bekijken we de eigenschappen van de dataset zodat we weten met welke datatypes we te maken hebben. Hiervoor kunnen we de `str()` functie gebruiken.

```
# de str() functie gebruiken om de structuur van de dataset te bekijken
str(CDMOW.df)
```

```
## 'data.frame':    69659 obs. of  4 variables:
##  $ V1: int  1 2 2 3 3 3 3 3 4 ...
##  $ V2: int  19970101 19970112 19970112 19970102 19970330 19970402 19971115 19971125 19980528 19970101 ...
##  $ V3: int  1 1 5 2 2 2 5 4 1 2 ...
##  $ V4: num  11.8 12 77 20.8 20.8 ...
```

We zien dat de dataset bestaat uit 69659 rijen (observaties/transacties) en 4 variabelen (kolommen). Daarvan zijn alle variabelen **integer** (hele getallen) en **numeric** (getallen met decimalen).

11.3 De dataset de juiste vorm geven

Er is dus werk aan de winkel. Als eerst geven kolomnamen aan de dataset converteren we de kolom met de datum (**Datum**) naar het **Date** datatype.

11.3.1 Kolomnamen geven aan de kolomnaam-loze dataset

Met de `names()` functie kunnen we namen geven aan de `CDNOW.df` data frame.

```
# namen toevoegen aan de CDMOW-dataset
names(CDMOW.df) <- c("Klant", "Datum", "Aantal", "Bedrag")

# opnieuw de dataset bekijken
str(CDMOW.df)
```

```
## 'data.frame':    69659 obs. of  4 variables:
##  $ Klant : int  1 2 2 3 3 3 3 3 4 ...
##  $ Datum : int  19970101 19970112 19970112 19970102 19970330 19970402 19971115 19971125 19980528 19970101 ...
##  $ Aantal: int  1 1 5 2 2 2 5 4 1 2 ...
##  $ Bedrag: num  11.8 12 77 20.8 20.8 ...
```

11.3.2 Een kolom converteren naar het date (datum) datatype

Met de `as.Date` functie kunnen we een waarde omzetten naar het **date** datatype. Hiervoor moeten met twee dingen rekening houden:

- De `as.Date` functie accepteert een **character** datatype;
- We moeten de structuur van de waarde aangeven die als datum geconverteerd moet worden.

Dit doen we op de volgende manier:

```
# om eerst proef te draaien, dus te voorkomen dat we onze goeie kolommen
# overschrijven

x <- as.Date(as.character(CDNOW.df$Datum), '%Y%m%d')
head(x)
```

```
## [1] "1997-01-01" "1997-01-12" "1997-01-12" "1997-01-02" "1997-03-30"
## [6] "1997-04-02"
```

```
class(x)
```

```
## [1] "Date"
```

We hebben het gewenste resultaat, we kunnen de kolom `Datum` uit de `CDNOW.df` data frame overschrijven met onze nieuwe kolom:

```
CDNOW.df$Datum <- as.Date(as.character(CDNOW.df$Datum), '%Y%m%d')
```

11.3.3 Gebruik maken van het Date datatype

Het `Date` datatype is anders dan het `character` datatype, al lijkt deze er erg veel op. Met het `Date` datatype kun je namelijk informatie halen uit de datum, en deze datum bewerken. Aan de hand van een aantal voorbeelden wordt dit gedemonstreerd.

Deze demonstratie past eigenlijk niet helemaal bij een case, maar ook in een echte situatie dwaal je af en toe af door met de functionaliteit te spelen.

```
# de head functie wordt gebruikt om steeds slechts een kleine sample
#van de data uit de variabele te laten zien

head(CDNOW.df$Datum)
```

```
## [1] "1997-01-01" "1997-01-12" "1997-01-12" "1997-01-02" "1997-03-30"
## [6] "1997-04-02"
```

```
head(CDNOW.df$Datum) + 31
```

```
## [1] "1997-02-01" "1997-02-12" "1997-02-12" "1997-02-02" "1997-04-30"
## [6] "1997-05-03"
```

```
head(CDNOW.df$Datum) + 31
```

```
## [1] "1997-02-01" "1997-02-12" "1997-02-12" "1997-02-02" "1997-04-30"
## [6] "1997-05-03"
```

```
head(months(CDNOW.df$Datum))
```

```
## [1] "januari" "januari" "januari" "januari" "maart" "april"
```



```
head(weekdays(CDNOW.df$Datum))
```

```
## [1] "woensdag" "zondag" "zondag" "donderdag" "zondag" "woensdag"
```

```
head(quarters((CDNOW.df$Datum)))
```

```
## [1] "Q1" "Q1" "Q1" "Q1" "Q1" "Q2"
```

```
head(as.numeric(format(CDNOW.df$Datum,'%Y')))
```

```
## [1] 1997 1997 1997 1997 1997 1997
```

```
(CDNOW.df$Datum[6] - CDNOW.df$Datum[1])
```

```
## Time difference of 91 days
```

Je kunt zien dat het in R mogelijk is om berekeningen te maken met datums.

11.4 Spelenderwijs de data verkennen

Om wat meer gevoel te krijgen met de data, is het goed om een aantal functies uit te proberen om de data te verkennen. Doe dit vooral op een manier die jou het beste lijkt. Hier volgen een aantal suggesties:

```
# de dataset
```

```
attach(CDNOW.df)
```

```
summary(CDNOW.df)
```

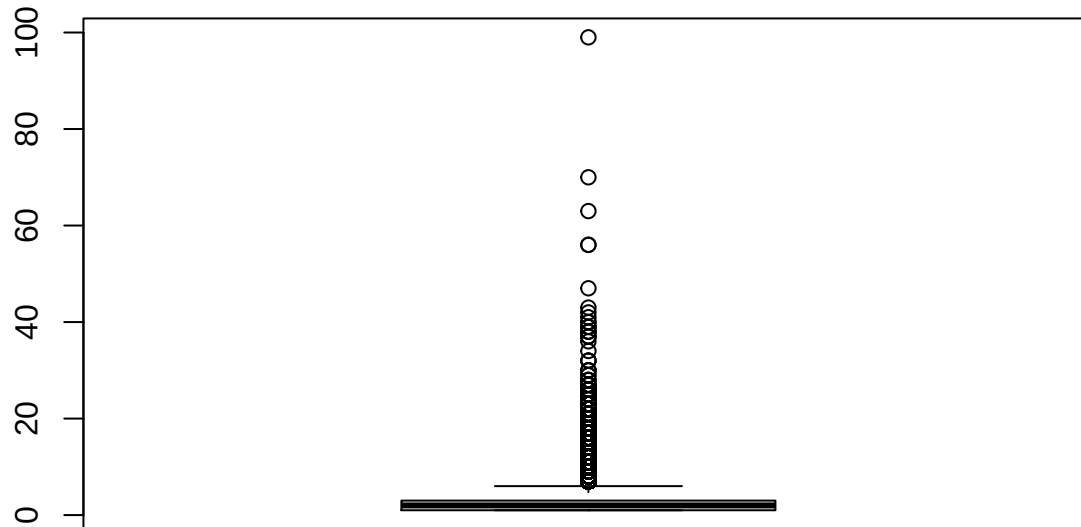
```
##      Klant      Datum      Aantal      Bedrag
## Min.   :    1  Min.   :1997-01-01  Min.   : 1.00  Min.   :  0.00
## 1st Qu.: 5506  1st Qu.:1997-02-22  1st Qu.: 1.00  1st Qu.: 14.49
## Median :11410  Median :1997-04-24  Median : 2.00  Median : 25.98
## Mean   :11471  Mean   :1997-07-02  Mean   : 2.41  Mean   : 35.89
## 3rd Qu.:17273  3rd Qu.:1997-11-07  3rd Qu.: 3.00  3rd Qu.: 43.70
## Max.   :23570  Max.   :1998-06-30  Max.   :99.00  Max.   :1286.01
```

```
# aantallen
```

```
mean(Aantal)
```

```
## [1] 2.41004
```

```
boxplot(Aantal)
```

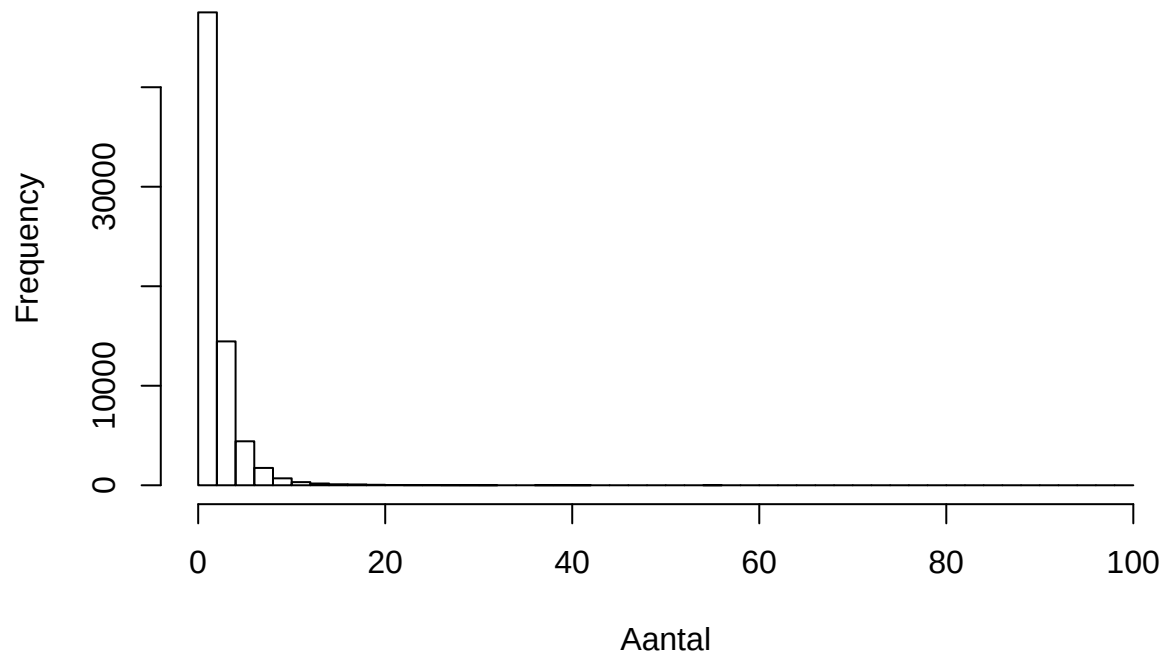


```
table(Aantal)
```

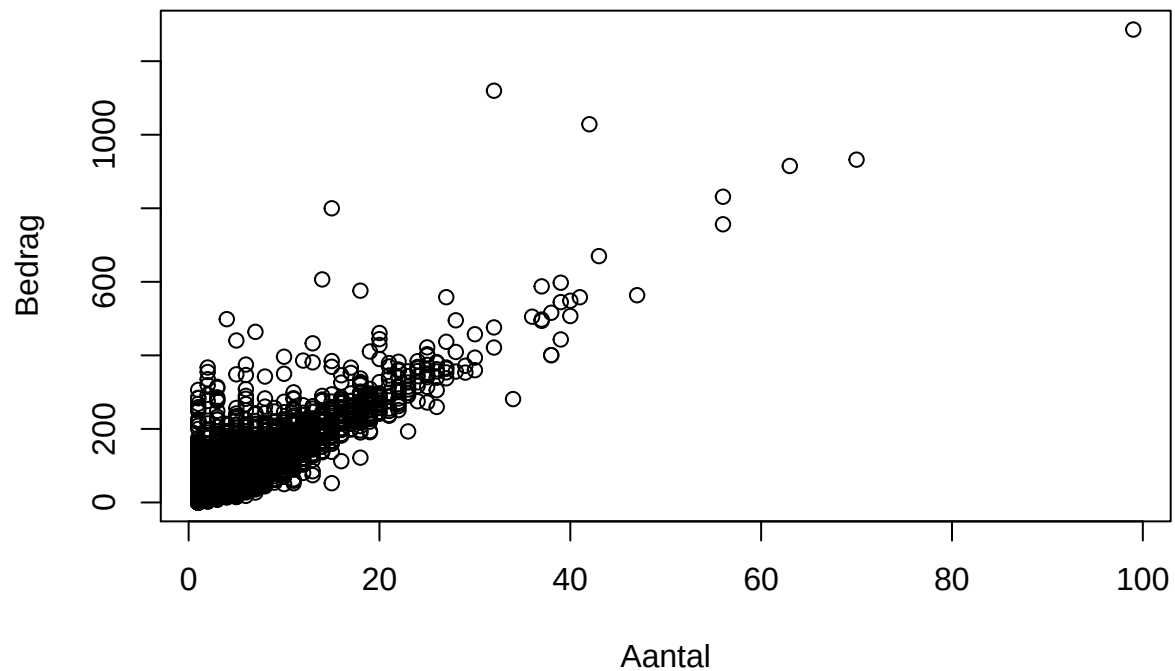
```
## Aantal
##      1      2      3      4      5      6      7      8      9     10     11     12
## 31454 16070  9444  5015  2792  1627  1057   684   403   293   180   136
##      13     14     15     16     17     18     19     20     21     22     23     24
##   108     70     59     39     34     48     26     22     12     15      8      8
##      25     26     27     28     29     30     32     34     36     37     38     39
##      9      7      6      3      2      3      3      1      1      3      3      3
##      40     41     42     43     47     56     63     70     99
##      2      1      1      1      1      2      1      1      1
```

```
hist(Aantal, breaks = length(unique(Aantal)))
```

Histogram of Aantal



```
plot(Aantal,Bedrag)
```



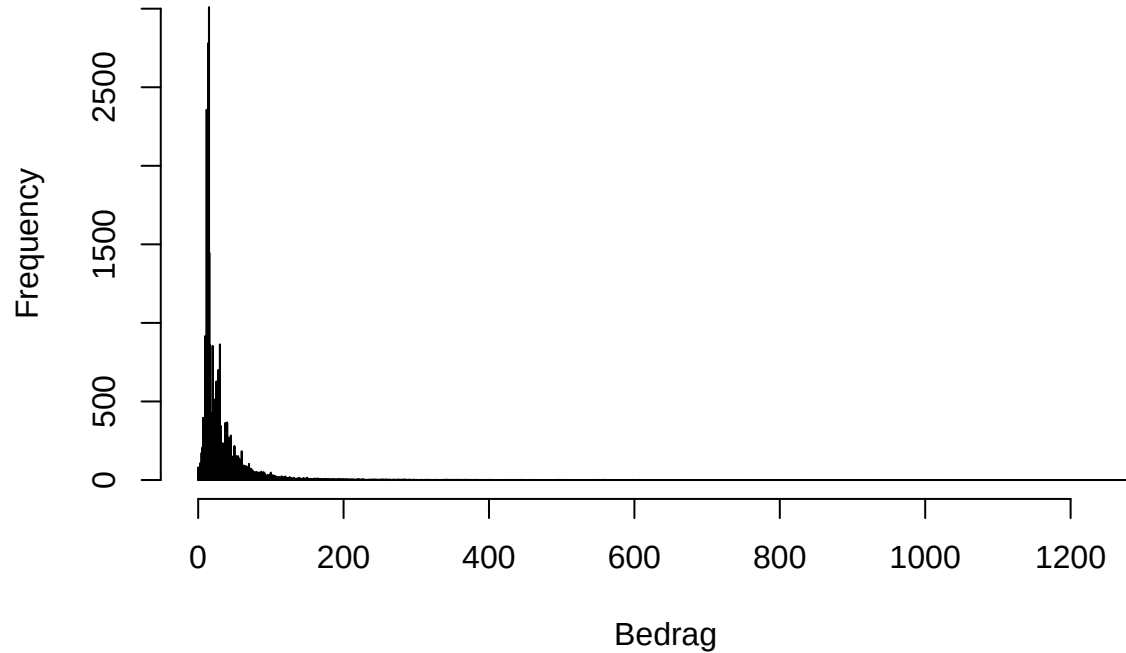
```
# bedragen
```

```
sum(Bedrag)
```

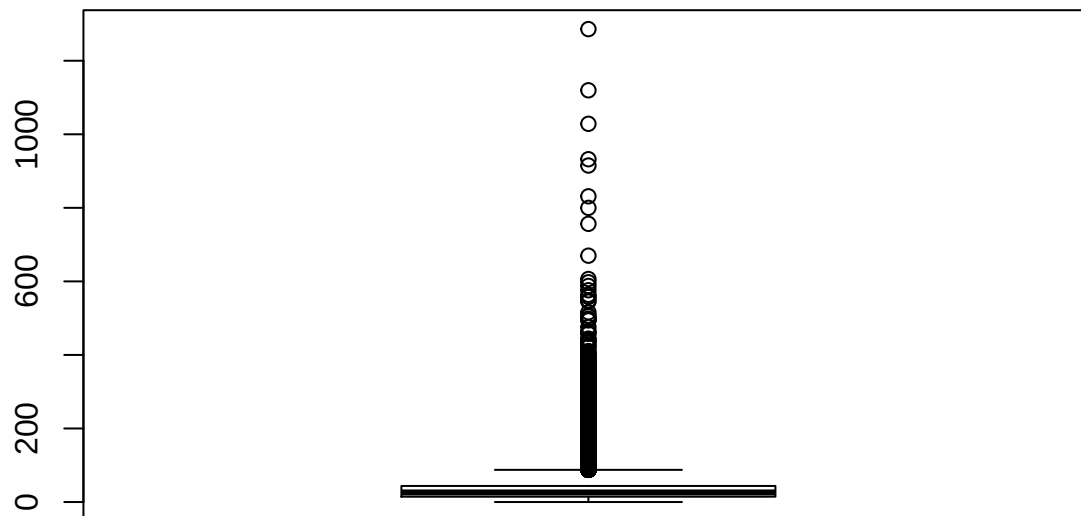
```
## [1] 2500316
```

```
hist(Bedrag, breaks = length(unique(Bedrag)))
```

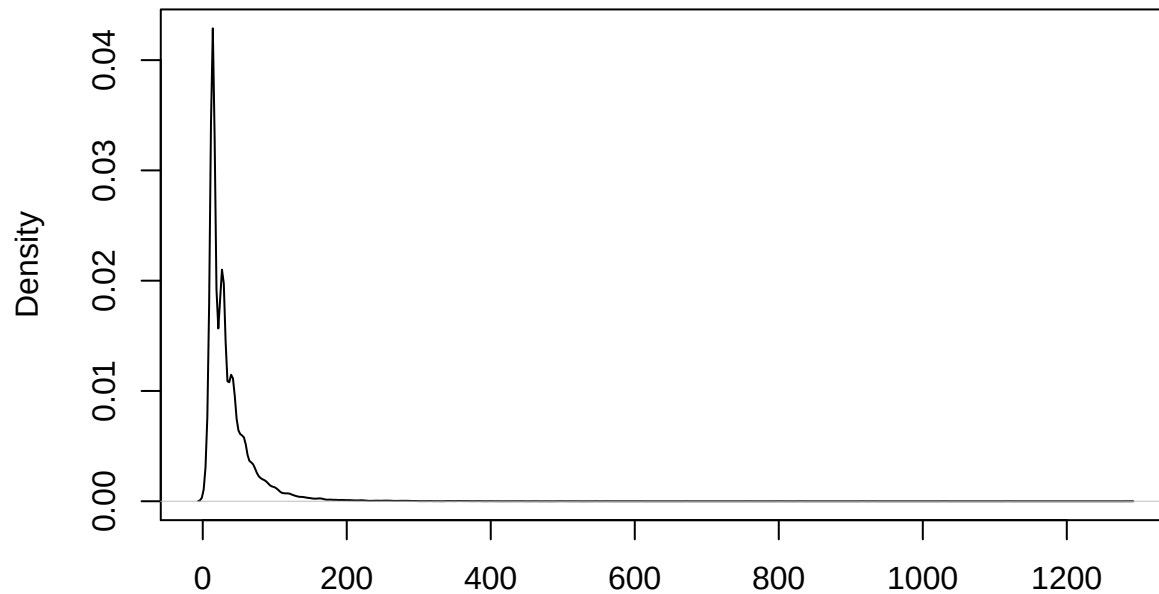
Histogram of Bedrag



```
boxplot(Bedrag)
```



```
plot(density(Bedrag))
```

density.default(x = Bedrag)

N = 69659 Bandwidth = 2.109

```
# datums
CDNOW.df$Maand <- months(Datum)
CDNOW.df$Kwartaal <- quarters(CDNOW.df$Datum)

detach(CDNOW.df)
attach(CDNOW.df)

by(CDNOW.df[, 3:4], Maand, colMeans)
```

```
## Maand: april
##   Aantal   Bedrag
## 2.549214 36.942218
## -----
## Maand: augustus
##   Aantal   Bedrag
## 2.521983 38.089522
## -----
## Maand: december
##   Aantal   Bedrag
## 2.563099 38.169868
## -----
## Maand: februari
##   Aantal   Bedrag
## 2.275605 34.342532
## -----
## Maand: januari
##   Aantal   Bedrag
## 2.253102 34.289868
```

```
## -----
## Maand: juli
##   Aantal   Bedrag
## 2.763766 41.495201
## -----
## Maand: juni
##   Aantal   Bedrag
## 2.469688 36.198778
## -----
## Maand: maart
##   Aantal   Bedrag
## 2.334098 34.891628
## -----
## Maand: mei
##   Aantal   Bedrag
## 2.495492 36.664541
## -----
## Maand: november
##   Aantal   Bedrag
## 2.840727 41.981324
## -----
## Maand: oktober
##   Aantal   Bedrag
## 2.421155 35.043236
## -----
## Maand: september
##   Aantal   Bedrag
## 2.495209 35.691986
```

```
by(CDNow.df[, 3:4], Kwartaal, colMeans)
```

```
## Kwartaal: Q1
##   Aantal   Bedrag
## 2.291004 34.532054
## -----
## Kwartaal: Q2
##   Aantal   Bedrag
## 2.506523 36.613209
## -----
## Kwartaal: Q3
##   Aantal   Bedrag
## 2.607965 38.686871
## -----
## Kwartaal: Q4
##   Aantal   Bedrag
## 2.614253 38.486024
```

11.4.1 Data verkennen zonder outliers

In de vorige voorbeelden heb je kunnen zien dat er veel outliers (extreme waarden) in de data zitten. Hierdoor zien de visualisaties er een beetje raar uit. Op verschillende manieren kun met outliers omgaan, bijvoorbeeld door ze weg te laten.

```

CDNOW.df$outlier <- ifelse(CDNOW.df$Bedrag
  > mean(CDNOW.df$Bedrag)+(3*(sd(CDNOW.df$Bedrag)))
  | CDNOW.df$Bedrag < mean(CDNOW.df$Bedrag)-(3*(sd(CDNOW.df$Bedrag))),
  "outlier",
  "geen_outlier")

CDNOW.df.zo <- subset(CDNOW.df, CDNOW.df$outlier != "outlier")

summary(CDNOW.df.zo)

```

```

##      Klant      Datum      Aantal      Bedrag
## Min.   :    1  Min.   :1997-01-01  Min.   : 1.000  Min.   : 0.00
## 1st Qu.: 5497  1st Qu.:1997-02-22  1st Qu.: 1.000  1st Qu.: 14.42
## Median :11402  Median :1997-04-23  Median : 2.000  Median : 25.74
## Mean   :11465  Mean   :1997-07-02  Mean   : 2.238  Mean   : 32.84
## 3rd Qu.:17263  3rd Qu.:1997-11-07  3rd Qu.: 3.000  3rd Qu.: 42.50
## Max.   :23570  Max.   :1998-06-30  Max.   :18.000  Max.   :144.72
##      Maand      Kwartaal      outlier
## Length:68493    Length:68493    Length:68493
## Class :character Class :character Class :character
## Mode  :character Mode  :character Mode  :character
##
##
##

```

```

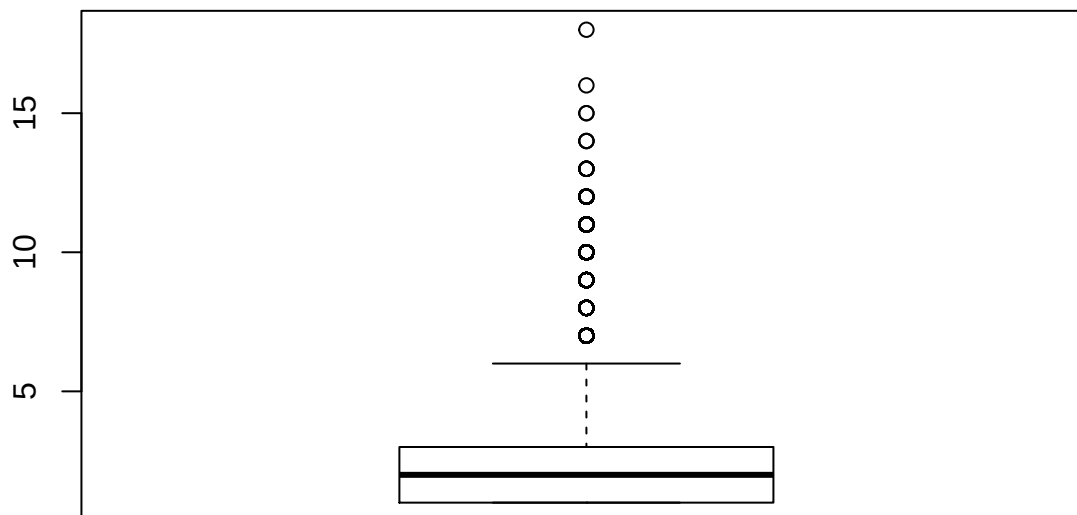
# aantallen

mean(CDNOW.df.zo$Aantal)

```

```
## [1] 2.23782
```

```
boxplot(CDNOW.df.zo$Aantal)
```

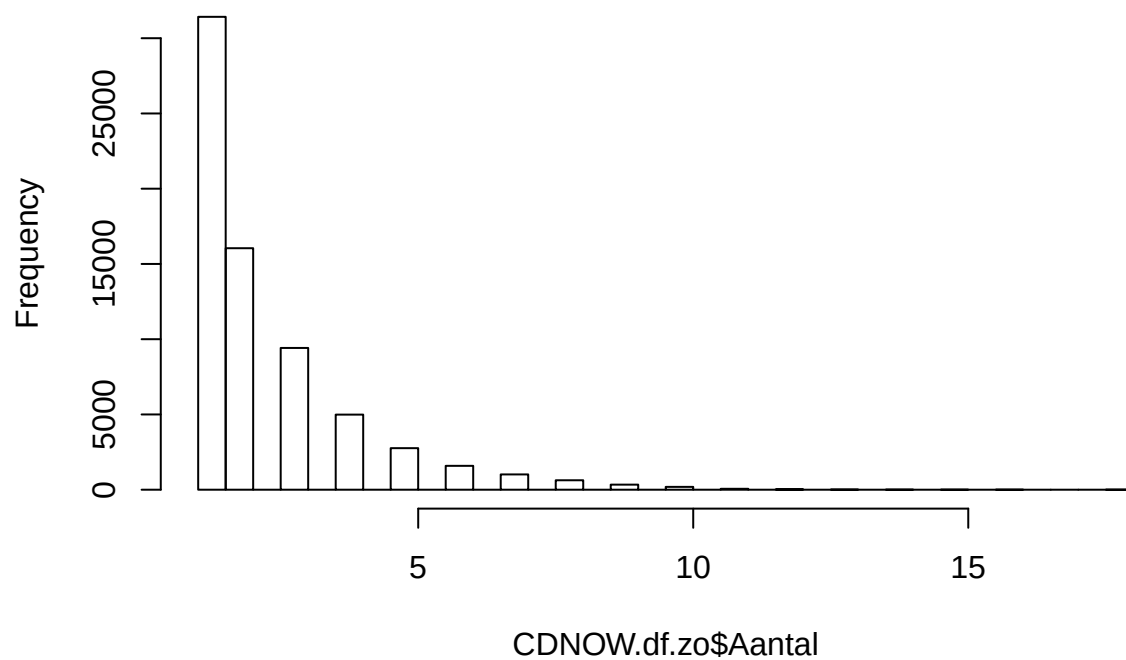


```
table(CDNOW.df.zo$Aantal)
```

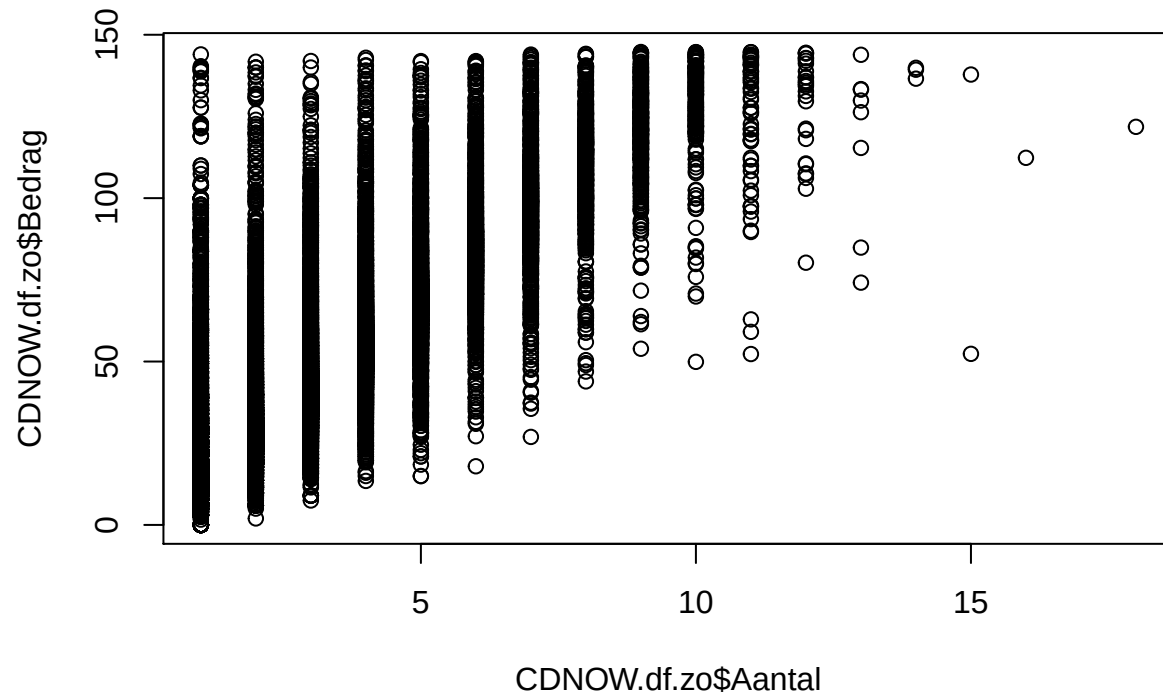
```
##
##      1      2      3      4      5      6      7      8      9     10     11     12
## 31422 16047  9419  4989  2761  1586  1016   627   337   187    60    27
##      13     14     15     16     18
##       8      3      2      1      1
```

```
hist(CDNOW.df.zo$Aantal, breaks = length(unique(Aantal)))
```

Histogram of CDNOW.df.zo\$Aantal



```
plot(CDNOW.df.zo$Aantal,CDNOW.df.zo$Bedrag)
```

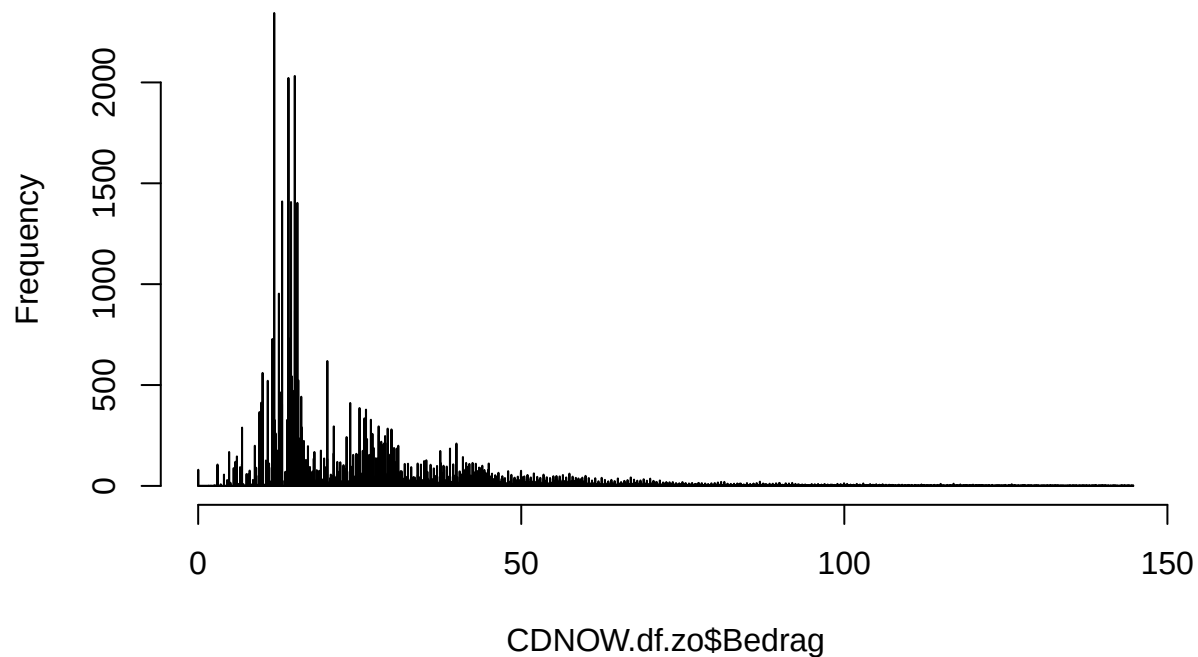
```
# bedragen
```

```
sum(CDNOW.df.zo$Bedrag)
```

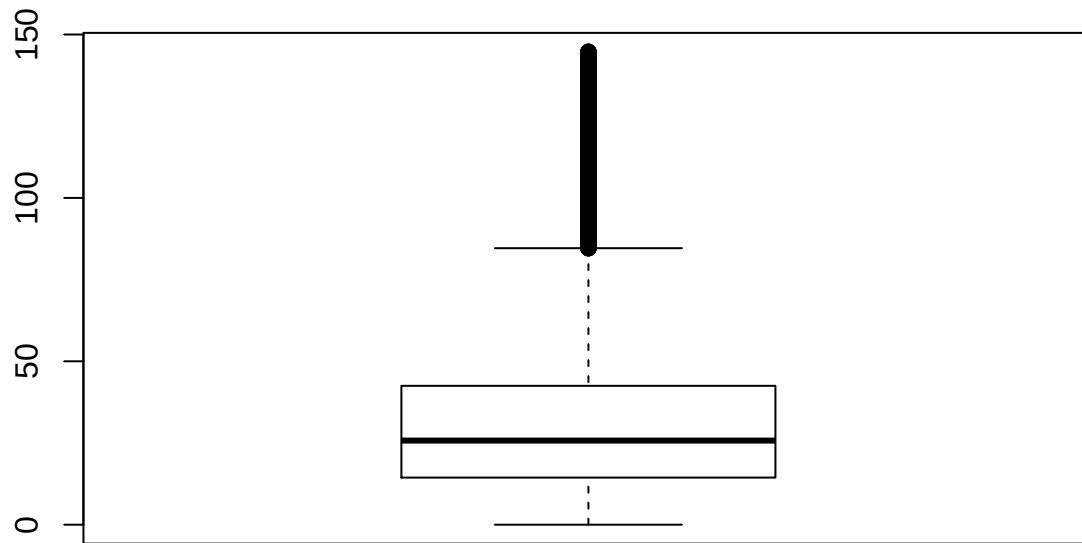
```
## [1] 2249614
```

```
hist(CDNOW.df.zo$Bedrag, breaks = length(unique(CDNOW.df.zo$Bedrag)))
```

Histogram of CDNOW.df.zo\$Bedrag

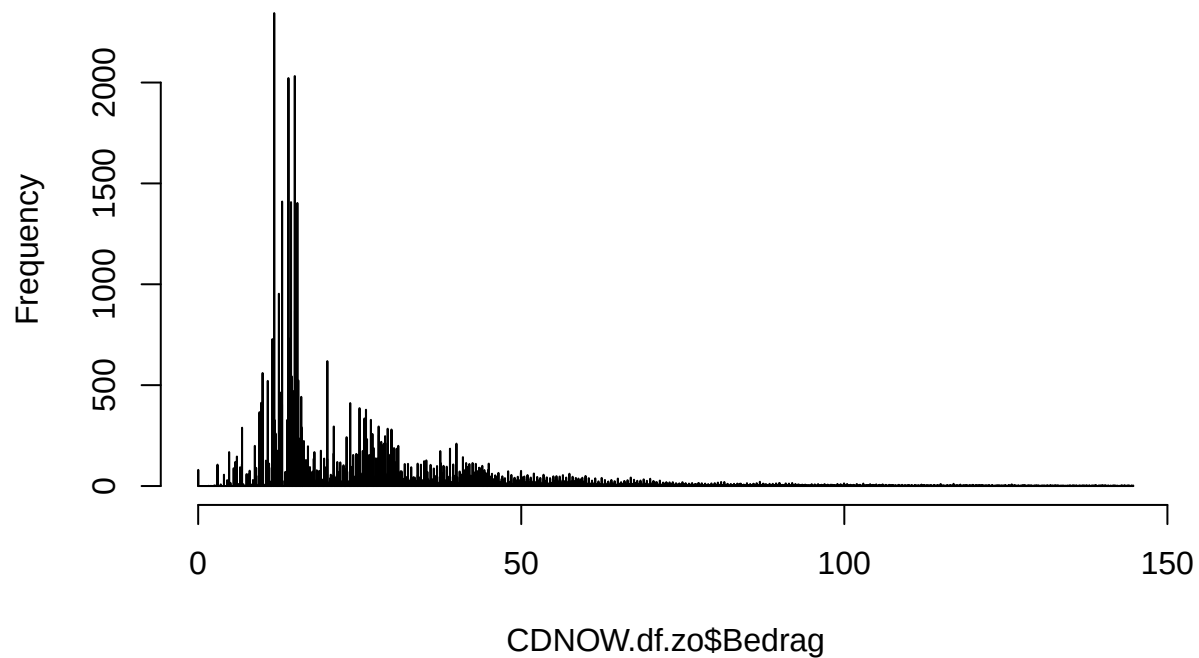


```
boxplot(CDNOW.df.zo$Bedrag)
```

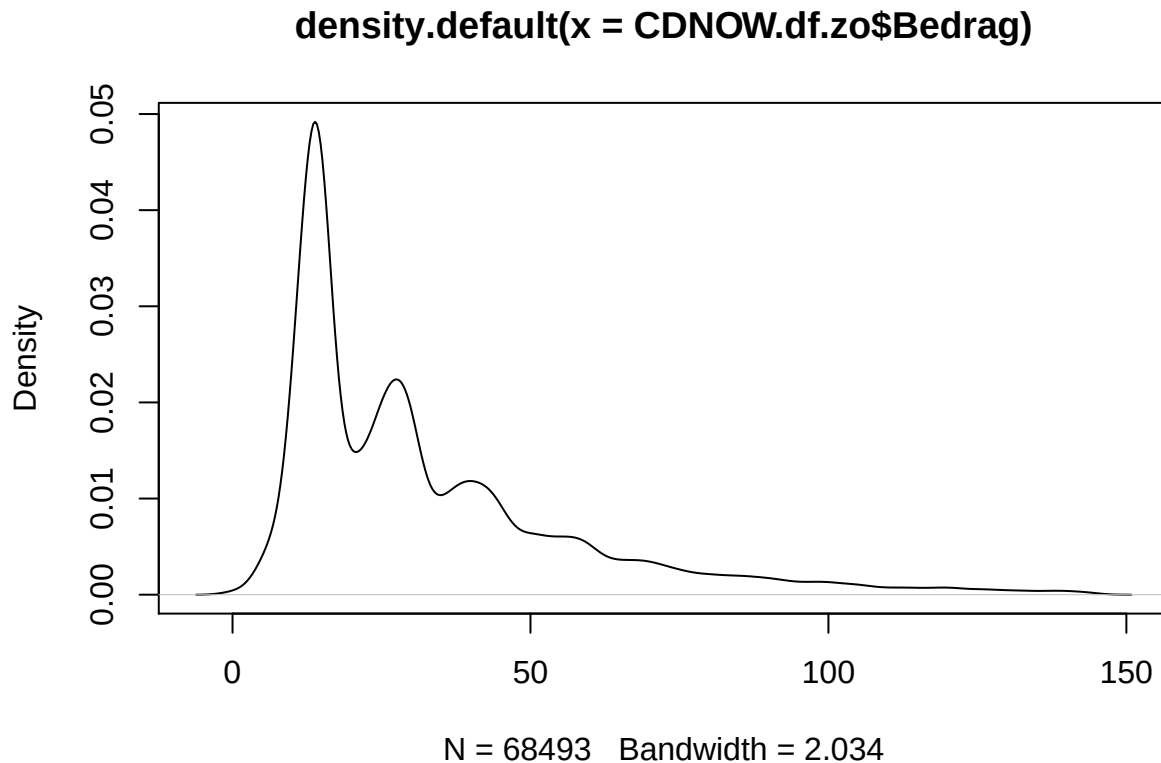


```
hist(CDNOW.df.zo$Bedrag, breaks = length(unique(CDNOW.df.zo$Bedrag)))
```

Histogram of CDNOW.df.zo\$Bedrag



```
plot(density(CDNOW.df.zo$Bedrag))
```



11.5 Totale uitgave per klant berekenen

De dataset bestaat uit aparte transacties. Hierbij dus ook voor dezelfde klant op verschillende koopmomenten. Als we de totale klantwaarde willen weten, moeten we per Klant alle Bedrag-waarden bij elkaar optellen. Dit doen we als volgt.

```
# een object aanmaken "TKB" dat staat voor "Totaal Klant Bedrag"
TKB <- aggregate(CDNOW.df$Bedrag ~ CDNOW.df$Klant, CDNOW.df, sum)

# checken hoe de waarden er uit zien
head(TKB)
```

```
##   CDNOW.df$Klant CDNOW.df$Bedrag
## 1             1          11.77
## 2             2           89.00
## 3             3          156.46
## 4             4          100.50
## 5             5          385.61
## 6             6           20.99
```

We kunnen zien dat we het gewenste resultaat hebben bereikt. We zien namelijk dat klant nummer 1 slechts 1 transactie heeft gedaan van 11,77 en klant nummer 2 transacties heeft gedaan die bij elkaar een waarde hebben van 89. We hebben nu dus de klantwaarde per klant berekend.

11.6 Klanten Segmenteren

We willen klantgroepen segmenteren op basis van het totale bedrag dat ze hebben uitgegeven. Als we een samenvatting geven van de variabele die we aangemaakt hebben met de totale klantwaarden, zien we de algemene statistieken: We veranderen de namen van de variabelen in de dataset TKB:

- `CDNOW.df$Klant` wordt `TBKant`;
- `CDNOW.df$Bedrag` wordt `TBedrag`.

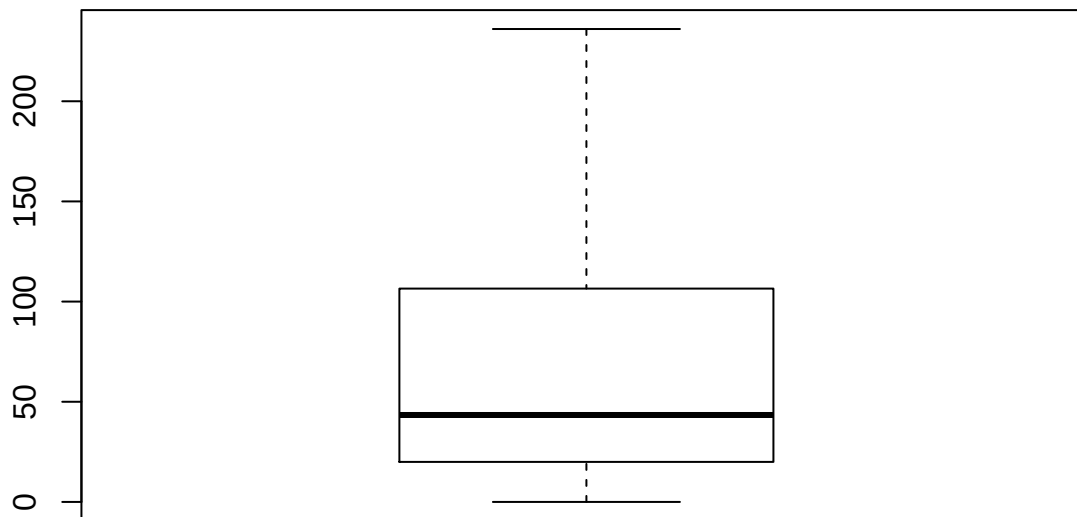
```
# de namen van TKB veranderen
names(TKB) <- c("TKlant", "TBedrag")
```

We definiëren de klantgroepen **Kleine Klanten**, **Midden Klanten** en **Grote Klanten** op basis van hun waarde. Met behulp van de samenvatting kunnen we een goede groepering maken.

```
summary(TKB$TBedrag)
```

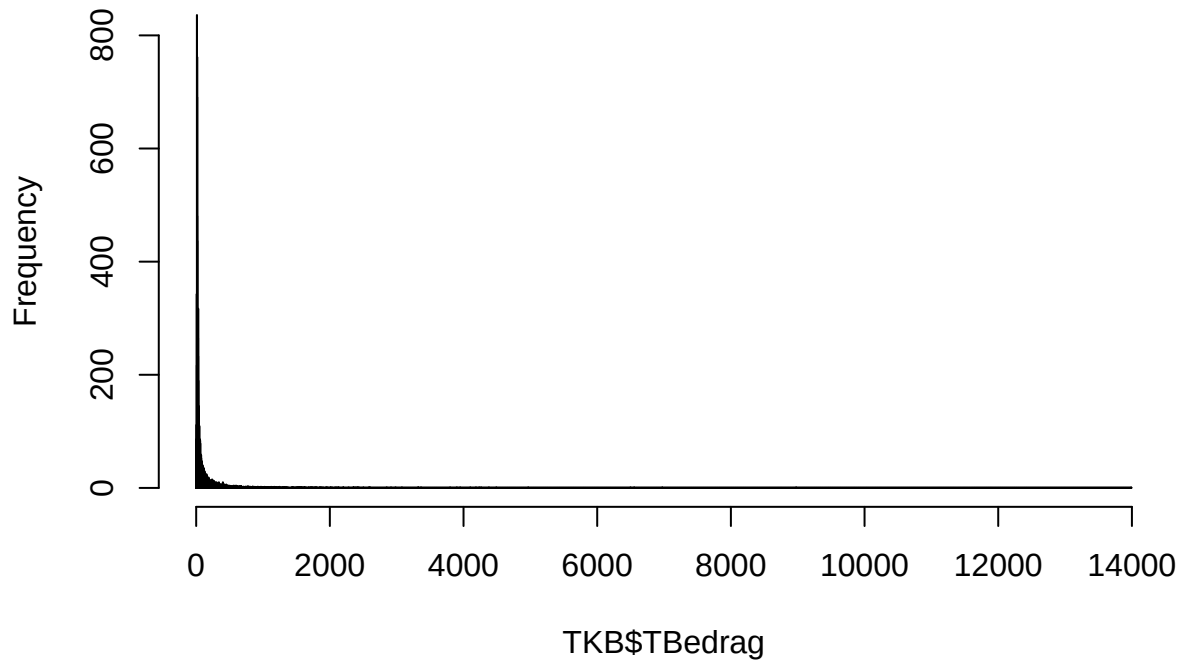
```
##      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.
##    0.00    19.97    43.40    106.10    106.50   13990.00
```

```
boxplot(TKB$TBedrag, outline = FALSE)
```



```
hist(TKB$TBedrag, breaks = nrow(TKB))
```

Histogram of TKB\$TBedrag



- Maak een aparte groep met *Kleine klanten*, klanten die minder dan 43.40 uitgegeven hebben;
- *Midden klanten*, tussen de 43.40 en 106.08 hebben uitgegeven;
- *Grote klanten*, meer dan 106,08 hebben uitgegeven.

```
# subsets maken voor de verschillende klantgroepen
KlKl <- subset(TKB, TKB$TBedrag < 43.40)

MiKl <- subset(TKB, TKB$TBedrag >= 43.40 & TKB$TBedrag <= 106.08)

GrKl <- subset(TKB, TKB$TBedrag > 106.08)
```

11.6.1 Een samenvatting geven van alle klantsegmenten

```
summary(KlKl)
```

```
##      TKlant      TBedrag
## Min.   :    1  Min.   : 0.00
## 1st Qu.: 6188  1st Qu.:13.97
## Median :12064  Median :19.97
## Mean   :11986  Mean   :22.08
## 3rd Qu.:17867  3rd Qu.:29.74
## Max.   :23569  Max.   :43.38
```

```
summary(MiKl)
```

```
##      TKlant      TBedrag
## Min.   :    2   Min.   : 43.41
## 1st Qu.: 5786   1st Qu.: 53.30
## Median :11718   Median : 65.72
## Mean   :11703   Mean    : 68.48
## 3rd Qu.:17569   3rd Qu.: 82.21
## Max.   :23570   Max.    :106.07
```

```
summary(GrKl)
```

```
##      TKlant      TBedrag
## Min.   :    3   Min.   : 106.1
## 1st Qu.: 5405   1st Qu.: 141.9
## Median :11425   Median : 202.9
## Mean   :11468   Mean    : 310.8
## 3rd Qu.:17272   3rd Qu.: 333.0
## Max.   :23568   Max.    :13990.9
```

11.7 Klantnummers verzamelen per klantgroep

Om de eigenschappen van de klanten per klantgroep later goed te kunnen bekijken, worden de nummers van de klanten per klantgroep verzameld.

```
KKN <- K1Kl$TKlant
MKN <- M1Kl$TKlant
GKN <- GrKl$TKlant
```

Ten slotte willen we alle data verzamelen per klantgroep. Dus de hele dataset onderverdeeld in 3 groepen.

```
KKD <- subset(CDNOW.df, CDNOW.df$Klant %in% KKN)
MKD <- subset(CDNOW.df, CDNOW.df$Klant %in% MKN)
GKD <- subset(CDNOW.df, CDNOW.df$Klant %in% GKN)
```

11.8 Informatie uit transactiedata presenteren

We vatten de informatie samen in een nette tabel die qua formaat direct in een rapport past. In de onderstaande demonstratie kun je zien hoeveel vrijheid je hebt in het combineren van functies. Er worden bijvoorbeeld combinaties gebruikt van de `paste()`, `length()`, `nrow()`, `round()`, en `mean()` functies.

```
aantalKlanten <- c(length(KKN), length(MKN), length(GKN))

gemiddeldTransactieBedrag <- c(paste('€ ',
                                     round(mean(KKD$Bedrag), 2)),
                              paste('€ ', round(mean(MKD$Bedrag), 2)) ,
                              paste('€ ', round(mean(GKD$Bedrag), 2)))

gemiddeldAantal <- c(round(mean(KKD$Aantal), 1),
                    round(mean(MKD$Aantal), 1),
                    round(mean(GKD$Aantal), 1))

aantalTransacties <- c(nrow(KKD), nrow(MKD), nrow(GKD))
```

```

klantgroepen <- c("Kleine klanten", "Midden klanten", "Grote klanten")

TransactieTabel <- data.frame(aantalKlanten,
                              gemiddeldTransactieBedrag,
                              gemiddeldAantal,
                              aantalTransacties,
                              row.names = klantgroepen)

names(TransactieTabel) <- c("Aantal_klanten",
                             "Gemiddeld_transactiebedrag",
                             "Gemiddeld_aantal",
                             "Aantal_transacties")

TransactieTabel

```

```

##           Aantal_klanten Gemiddeld_transactiebedrag Gemiddeld_aantal
## Kleine klanten          11785                € 18.45              1.3
## Midden klanten           5872                € 29.36              2.0
## Grote klanten           5913                € 43.91              2.9
##           Aantal_transacties
## Kleine klanten          14109
## Midden klanten          13697
## Grote klanten           41853

```

Buiten deze analyse zijn er nog veel meer analyses mogelijk. Denk bijvoorbeeld aan het uitzoeken voor welk bepaald percentage omzet welke klantgroep verantwoordelijk is.

Merk op dat het een vrij ongebalanceerde segmentatie is. Normaal gesproken is de middengroep het grootst en de groep met grote klanten veel kleiner.

11.9 Informatie presenteren over de perioden

We sluiten deze case af met een presentatie van de data over perioden. Hiervoor werken we voornamelijk met de `Datum` kolom.

11.9.1 De totale omzet per dag berekenen

```

totaleOmzet <- aggregate(CDNow.df$Bedrag ~ CDNow.df$Datum, CDNow.df, sum)
head(totaleOmzet)

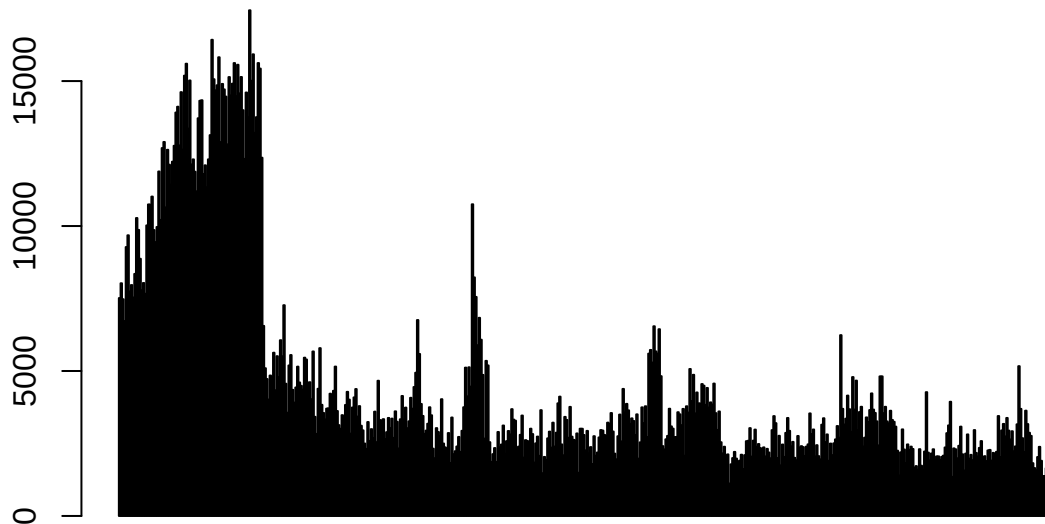
```

```

##   CDNow.df$Datum CDNow.df$Bedrag
## 1 1997-01-01      7515.35
## 2 1997-01-02      8025.95
## 3 1997-01-03      7475.04
## 4 1997-01-04      6722.93
## 5 1997-01-05      9274.80
## 6 1997-01-06      9680.55

```

```
names(totaleOmzet) <- c("Datum", "Bedrag")
barplot(totaleOmzet$Bedrag)
```



11.9.2 De totale aantallen per dag

```
totaalAantal <- aggregate(CDNow.df$Aantal ~ CDNow.df$Datum, CDNow.df, sum)
head(totaalAantal)
```

```
##      CDNow.df$Datum CDNow.df$Aantal
## 1      1997-01-01             494
## 2      1997-01-02             530
## 3      1997-01-03             495
## 4      1997-01-04             404
## 5      1997-01-05             590
## 6      1997-01-06             620
```

```
names(totaalAantal) <- c("Datum", "Aantal")
```

*Naast de analyses in deze case kunnen er natuurlijk meer uitgebreide analyses worden toegepast op deze dataset. Je kunt bijvoorbeeld algoritmes uit verschillende packages op deze dataset toepassen om tot meer waardevolle inzichten te komen. Een voorbeeld hiervan is een artikel dat ik heb geplaatst op LinkedIn [17] **Transactiedata analyseren voor voorspellende inzichten in klantgedrag**, waarbij ik het package *BTYD* (Buy till you die) gebruik om met de algoritmes in dit package waardevolle inzichten over de Customer Lifetime Value (CLV) van klanten te voorspellen.*

12 Correlaties, functies en loops in R

Dit ziet eruit als een rare combinatie van onderwerpen om te behandelen in een boek waarin R wordt uitgelegd. Eerlijk gezegd is dat ook zo. Echter is dit een handige manier om met deze onderwerpen kennis te maken. Dit hoofdstuk geeft namelijk een echt probleem weer waar je een functie of een loop voor zou kunnen toepassen. Daarbij worden deze onderwerpen uitgebreid behandeld.

Een correlatie geeft een indicatie van de samenhang tussen twee variabelen. Dit interpreteer je als: “Als A sterk omhoog gaat, gaat B ook sterk omhoog.” Je spreekt dan van een sterke positieve correlatie. Deze correlaties geven je goede handvatten voor het data-analyse proces. Functies zijn een van de belangrijkste bouwstenen van R. Deze functies kun je zelf schrijven door een reeks aan commando's in een functie te plaatsen. Hierdoor kan je de functie de reeks aan commando's automatisch laten uitvoeren, maar dan voor bijvoorbeeld een andere waarde.

Loops zijn handig als je een bepaalde actie simpelweg wilt herhalen. Je kunt bij loops aangeven hoe vaak je iets wilt herhalen of voor welke variabelen je een herhaling wilt toepassen. Hierbij moet je wel oppassen dat je het eindpunt van een loop aangeeft. Zonder eindpunt gaat de loop voor eeuwig door (*een infinite loop*) en crasht de R-sessie. Maak je geen zorgen, de instructies zijn duidelijk en het is in R niet heel makkelijk om een infinite loop te schrijven.

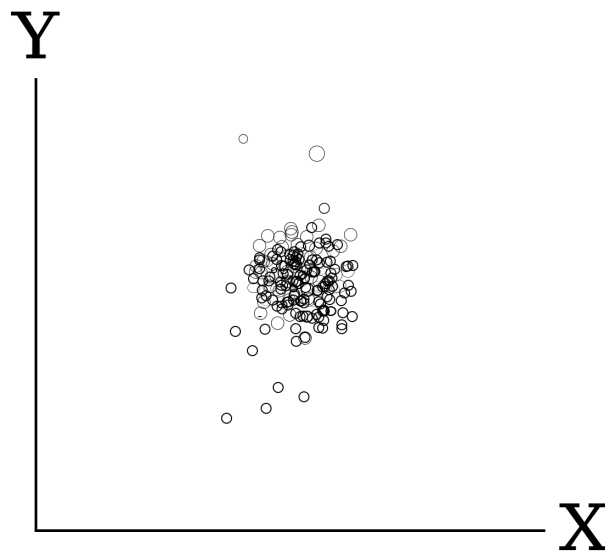


Figure 22: Correlatie

12.1 De definitie van een correlatie

Een correlatie wordt in de statistiek gebruikt om de mate van samenhang tussen twee variabelen te beoordelen. Hier is een formule voor opgesteld. De waarde die uit deze formule komt varieert tussen de -1 en 1. Deze waarde wordt de **correlatiecoëfficiënt** genoemd. Een correlatie van 1 geeft een sterke positieve samenhang aan. Een correlatie van -1 geeft een sterke negatieve samenhang aan. In beide gevallen is er sprake van een sterke samenhang, positief of negatief.

Een voorbeeld van een sterke positieve samenhang (of dus correlatie) is: “Des te gladder het is op de snelweg, des te meer ongelukken er op de snelweg plaatsvinden.”

Een voorbeeld van een sterke negatieve correlatie is: “Des te hoger de prijzen voor vakanties, des te minder klanten er een vakantie boeken.”

*Een artikel [8] **Correlatiecoëfficiënt**, (M Zandee, 2006) wordt het principe en de toepassing van correlaties uitgebreid, vooral vanuit de statistische hoek, uitgelegd.*

12.2 Correlaties berekenen

12.2.1 Correlaties berekenen uit handmatige data frames

Gelukkig hoeft je in R deze formule niet handmatig in te vullen, maar kun je gebruik maken van de `cor()` functie. De `cor` functie neemt twee variabelen als argumenten op. In het kader van de zojuist gegeven voorbeelden zou het er in R bijvoorbeeld als volgt uit kunnen zien:

```
#Vector maken met dagen
dag <- c(1:13)

#Vector met ongelukken
ongelukken <- c(0,1,0,3,2,6,7,5,8,3,1,3,1)

#Vector met gladheid in scores 1 t/m 5
gladheid <- c(1,1,2,1,4,3,4,4,5,4,2,2,1)

#Ongelukken data frame
ongelukkenGladheid <- data.frame(dag,ongelukken,gladheid)

ongelukkenGladheid
```

##	dag	ongelukken	gladheid
## 1	1	0	1
## 2	2	1	1
## 3	3	0	2
## 4	4	3	1
## 5	5	2	4
## 6	6	6	3
## 7	7	7	4
## 8	8	5	4
## 9	9	8	5
## 10	10	3	4
## 11	11	1	2
## 12	12	3	2
## 13	13	1	1

```
attach(ongelukkenGladheid)
cor(ongelukken, gladheid)
```

```
## [1] 0.7450994
```

```
detach(ongelukkenGladheid)
```

Zoals je kunt zien, een vrij hoge correlatie van 0.745. Des te hoger de gladheid score, des te meer ongelukken.

De gladheidsscore is slechts een verzinsel om de intensiteit van gladheid aan te geven. Naar mijn weten bestaat er niet iets als een gladheidsscore.

12.2.2 Een correlatie berekenen uit bestaande data frames

Naast zelfgemaakte data frames, kunnen er natuurlijk ook correlaties berekend worden uit bestaande data frames. Ook hier gaan we de `mtcars` dataset voor gebruiken.

We bekijken even de variabelen die er zijn.

```
names(mtcars)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

```
attach(mtcars)
```

Laten we de correlatie berekenen tussen `hp` (horse power/paardenkracht) en `mpg` (miles per gallon/zuinigheid).

```
cor(hp,mpg)
```

```
## [1] -0.7761684
```

De uitkomst is een zeer lage correlatie. Dit kunnen we interpreteren als een sterk negatief verband: Des te meer paardenkracht een auto heeft, des te minder zuinig deze auto is.

12.3 Correlatiematrix maken

12.3.1 Alleen numerieke waarden in een correlatie

Naast individueel de correlatie uitrekenen tussen twee variabelen, kunnen we ook een matrix maken die tussen alle variabelen de correlatie uitrekend. Dit doen we gewoon door de `cor()` functie te gebruiken, maar in plaats van twee variabelen daar in de plaatsen, wordt de naam van de **data frame** er in geplaatst. Voorwaarde voor een correlatiematrix is echter wel dat alle variabelen numeriek zijn. Dat checken we eerst even. Hier gebruiken we de `str()` functie voor.

```
str(CP)
```

```
## 'data.frame': 14 obs. of 59 variables:
## $ Jaar : int 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 ...
## $ Benzine.Euro95 : num 1.16 1.15 1.14 1.16 1.25 1.35 1.41 1.46 1.54 1.35 ...
## $ Diesel : num 0.85 0.82 0.79 0.79 0.89 1.02 1.09 1.1 1.29 1.01 ...
## $ LPG : num 0.45 0.43 0.43 0.43 0.46 0.51 0.57 0.59 0.68 0.51 ...
## $ Frikandel : num 0.89 0.96 1.04 1.06 1.08 1.1 1.12 1.16 1.21 1.25 ...
## $ Glas.bier : num 1.28 1.41 1.54 1.6 1.62 1.67 1.71 1.79 1.86 1.94 ...
## $ Glas.rode.huiswijn : num 1.83 2.01 2.18 2.26 2.31 2.36 2.38 2.47 2.64 2.79 ...
## $ Kopje.zwarte.koffie : num 1.18 1.28 1.41 1.47 1.51 1.55 1.57 1.62 1.7 1.81 ...
## $ Pizza : num 6.25 6.42 6.85 6.97 7.02 7 7.16 7.46 7.99 8.19 ...
## $ Aardappelen : num 0.63 0.75 0.87 0.79 0.73 0.58 0.86 0.81 0.77 0.72 ...
## $ Appelmoes : num 0.41 0.45 0.53 0.58 0.53 0.49 0.47 0.61 0.75 0.81 ...
## $ Bananen : num 1.51 1.62 1.74 1.7 1.54 1.62 1.52 1.53 1.57 1.6 ...
## $ Bier : num 1.17 1.29 1.38 1.43 1.29 1.28 1.31 1.37 1.46 1.6 ...
## $ Boterhamworst : num 6.31 7.49 7.81 7.99 7.69 7.61 7.45 7.12 7.17 7.39 ...
## $ Brood : num 1.29 1.34 1.38 1.39 1.41 1.43 1.37 1.35 1.48 1.46 ...
## $ Champignons : num 0.89 0.91 0.98 0.93 0.9 0.87 0.88 0.91 0.93 0.92 ...
## $ Cornflakes : num 2.39 2.67 2.85 2.95 2.97 2.97 3.25 3.53 3.75 3.88 ...
## $ Diepvries.patates.frites : num 1.06 1.09 1.18 1.14 1.06 0.98 1.02 1.31 1.33 1.26 ...
## $ Eieren : num 1.4 1.4 1.4 1.9 1.6 1.4 1.4 1.4 1.5 1.5 ...
## $ Hagelslag : num 1.17 1.23 1.3 1.38 1.22 1.18 1.15 1.13 1.21 1.29 ...
## $ Hamlappen : num 7.63 8.81 8.12 8 7.75 7.55 7.93 7.73 7.84 8.29 ...
## $ Kaas : num 7.01 7.34 7.58 7.67 7.34 7.18 7.24 7.11 8.39 8.15 ...
## $ Kipfilet : num 7.65 8.65 8.59 8.34 7.67 7.48 7.32 7.27 7.44 7.55 ...
## $ Koffie : num 6.16 5.96 5.76 5.54 4.94 5.5 5.8 6 6.6 6.58 ...
## $ Komkommer : num 0.81 0.8 0.91 0.87 0.75 0.77 0.89 0.91 0.83 0.84 ...
## $ Koolzuurhoudend.mineraalwater : num 0.25 0.25 0.25 0.26 0.21 0.2 0.31 0.35 0.37 0.41 ...
## $ Krentenbol : num 0.25 0.26 0.27 0.29 0.29 0.29 0.3 0.31 0.32 0.33 ...
## $ Macaroni : Factor w/ 13 levels " 0.82"," 0.88",...: 4 7 8 6 2 1 2 3 5 9 ...
## $ Margarine : num 0.64 0.69 0.72 0.76 0.73 0.68 0.68 0.7 0.78 0.79 ...
## $ Mayonaise : num 0.77 0.82 0.85 0.9 0.87 0.84 0.8 0.89 1.09 1.12 ...
## $ Melk : num 0.51 0.57 0.61 0.62 0.6 0.6 0.6 0.58 0.7 0.71 ...
## $ Mineraalwater.zonder.koolzuur : num 0.29 0.36 0.36 0.31 0.29 0.28 0.27 0.27 0.29 0.28 ...
## $ Olijfolie : num 5.39 5.18 5.04 4.98 4.75 4.62 5.37 5.41 5.24 5.13 ...
## $ Pasta : num 1.49 1.65 1.66 1.54 1.2 1.12 1.2 1.3 1.51 1.84 ...
## $ Plantaardige.olie : num 1.03 1.06 1.19 1.15 1.05 0.88 0.87 0.85 1.22 1.42 ...
## $ Riblappen : num 8.65 9.27 9.4 9.01 8.59 8.6 9 9.02 9.19 9.29 ...
## $ Rijst : num 2.12 2.21 2.27 2.27 1.81 1.5 1.39 1.48 1.73 2.03 ...
## $ Roomboter : num 1.09 1.11 1.12 1.09 1.05 1.01 1.01 1.08 1.22 1.15 ...
## $ Rundergehakt : num 5.87 6.08 6.02 5.71 5.05 5.19 5.18 5.02 5.22 5.4 ...
## $ Shampoo : num 1.72 1.78 1.84 1.84 1.62 1.49 2.05 2.07 2.17 2.15 ...
## $ Sinaasappelsap : num 0.84 0.86 0.87 0.89 0.8 0.73 0.74 0.91 0.95 0.89 ...
## $ Sperziebonen.in.blik : Factor w/ 12 levels " 0.52"," 0.54",...: 3 6 6 5 5 4 1 2 7 8 ...
## $ Suiker : num 0.88 0.89 0.88 0.85 0.82 0.79 0.78 0.8 0.78 0.79 ...
## $ Tandpasta : num 2.39 2.71 2.81 2.74 2.46 2.34 2.31 2.31 2.34 2.51 ...
## $ Thee : num 1.02 1.1 1.15 1.12 0.97 0.87 0.85 0.84 0.9 0.91 ...
## $ Toilet papier : num 2.9 3.06 3.18 3.29 3.05 2.94 3 3.04 3.16 3.15 ...
## $ Tomatenketchup : num 1.55 1.61 1.64 1.72 1.8 1.81 1.84 1.88 2.07 2.16 ...
## $ Tonijn.in.blik : num 1.09 1.15 1.25 1.27 1.21 1.21 1.23 1.1 1.2 1.33 ...
## $ Vanillevla : num 0.84 0.94 0.99 0.98 0.92 0.9 0.86 0.87 0.95 0.87 ...
## $ Zelfrijzend.bakmeel : num 0.99 1.02 1.12 1.08 1.07 1.01 1.07 1.1 1.14 1.07 ...
## $ APK.keuring : Factor w/ 13 levels " 35.03"," 38.53",...: 8 11 12 10 9 7 6 5 4 3 ...
## $ Autorijles : num 29.5 32.3 34.5 36.3 37.5 ...
## $ Bioscoopkaartje : num 6.72 6.99 8.1 7.35 7.56 7.8 7.99 8.14 8.28 8.5 ...
```

```
## $ Car.wash           : num  7.15 7.63 8.17 8.26 8.49 8.65 8.83 8.87 9.25 9.53 ...
## $ Dameskapper        : num  25.6 27 28.9 30.6 31.3 ...
## $ Herenkapper        : num  13.4 14.4 15.3 16.1 16.9 ...
## $ Krant              : num  1.03 1.06 1.11 1.17 1.22 1.25 1.22 1.25 1.26 1.39 ...
## $ Testament          : Factor w/ 14 levels ".","217.46","226.21",...: 2 3 4 5 6 7 8 9 10 11
## $ Zwemles            : num  5.23 6.51 7.43 7.92 7.88 8.55 9.05 9.33 9.43 9.92 ...
```

Merk op dat de producten `Macaroni`, `Sperziebonen.in.blik`, `APK.keuring`, `Testament` de datatype `factor` hebben. Echter kunnen we alleen een correlatie maken met variabelen die de datatype `numeric` hebben, dit geldt dus ook voor een correlatiematrix. We krijgen namelijk de foutmelding `'x' must be numeric` als we dit niet doen:

Daarom willen we `Macaroni` en `Sperziebonen.in.blik` omzetten in een numerieke variabelen.

12.3.2 In de knoop met datatypes in R

De volgende serie aan commands laat zien hoe datatypes in elkaar zitten en hoe we deze kunnen manipuleren:

```
#De huidige datatype van Macaroni bekijken
class(Macaroni)
```

```
## [1] "factor"
```

Het datatype van `Macaroni` is `factor`. De eigenschappen kun je zien als we deze weergeven. Er wordt aangegeven dat er sprake is van 13 levels, dit zijn gewoon de verschillende prijzen die er hier voor `Macaroni` zijn op volgorde.

```
Macaroni
```

```
## [1] 1.09 1.21 1.22 1.13 0.88 0.82 0.88 0.95 1.10 1.34
## [11] 1.36 1.35 1.38 .
## 13 Levels: 0.82 0.88 0.95 1.09 1.10 1.13 1.21 ... .
```

12.3.3 De integer-datatype: Zegt vaarwel tegen decimalen

Als we grofweg de `as.numeric()` of `is.numeric()` functie gebruiken, wordt `Macaroni` als integer opgeslagen. Het nadeel hiervan is dat het datatype “integer” staat voor **volledige** getallen en daarom de decimalen verwijderd.

```
as.numeric(Macaroni)
```

```
## [1] 4 7 8 6 2 1 2 3 5 9 11 10 12 13
```

```
class(as.numeric(Macaroni))
```

```
## [1] "numeric"
```

Als we echter naar de data kijken, zien we dat het niet eens meer lijkt op de prijzen van `Macaroni`.

12.3.4 Omzetten naar een numeriek datatype door gebruik te maken van `matrix`

Een manier om de data numeriek te maken zonder data te verliezen, is door gebruik te maken van het `matrix` formaat. We bekijken waarom dit zo is.

```
#De variabele "Macaroni" als matrix weergeven
as.matrix(Macaroni)
```

```
##      [,1]
## [1,] "  1.09"
## [2,] "  1.21"
## [3,] "  1.22"
## [4,] "  1.13"
## [5,] "  0.88"
## [6,] "  0.82"
## [7,] "  0.88"
## [8,] "  0.95"
## [9,] "  1.10"
## [10,] "  1.34"
## [11,] "  1.36"
## [12,] "  1.35"
## [13,] "  1.38"
## [14,] "  ."
```

Let op de indicaties van de datapunten in de matrix. Datapunt `[7,1]` geeft bijvoorbeeld de prijs 0.88 aan. Het zijn de juiste prijzen, maar echter in het verkeerde formaat. De aanhalingstekens " verklappen namelijk dat het geen numerieke waarden zijn. Numerieke waarden worden namelijk aangegeven zonder aanhalingstekens. De volgende stap is daarom het omzetten van de waarden in de matrix in het `numeric` formaat.

```
as.numeric(as.matrix(Macaroni))
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 1.09 1.21 1.22 1.13 0.88 0.82 0.88 0.95 1.10 1.34 1.36 1.35 1.38 NA
```

NAs introduced by coercion. Dit is een waarschuwing die je vrij vaak gaat tegenkomen in R. Het zegt namelijk dat tijdens de conversie (coercion: dwingen van een waarde om in een bepaald formaat te veranderen) de waarde verloren is gegaan. In dit geval is dat gebeurd bij de laatste waarde in de matrix, `[14,1]`. Dit was namelijk een lege *character* waarde, een . of een spatie/lege ruimte. R kan deze niet omzetten naar de waarde 0 en geeft daarom de waarde **NA** aan dat datapunt, wat staat voor **Not Available**. Dit is voor ons niet erg want we weten dat het om een leeg datapunt gaat. Daarom zetten we deze later om in een andere waarde.

Nu willen we er zeker van zijn dat deze waarden in de data frame CP terecht komen voor de variabele Macaroni. We schrijven de huidige waarde van `CP$Macaroni` over met onze zojuist aangemaakte waarden.

```
CP$Macaroni <- as.numeric(as.matrix(Macaroni))
```

```
## Warning: NAs introduced by coercion
```

```
class(CP$Macaroni)
```

```
## [1] "numeric"
```

```
CP$Macaroni
```

```
## [1] 1.09 1.21 1.22 1.13 0.88 0.82 0.88 0.95 1.10 1.34 1.36 1.35 1.38 NA
```

Gelukt! De waarden van de variabele `Macaroni` hebben de datatype `numeric`. Echter hebben we nu een nieuw probleem, de NA-waarde.

12.3.5 Omgaan met NAs (missing values) bij berekeningen in R

Een datacollectie of vector met een NA-waarde, zorgt ervoor dat er niet op de standaard manier statistische- of rekenfuncties kunnen worden toegepast. Als we nu bijvoorbeeld het gemiddelde willen berekenen van ons zojuist aangepaste variabele `Macaroni` (uit de CP data frame), krijgen we het volgende resultaat:

```
mean(CP$Macaroni)
```

```
## [1] NA
```

```
min(CP$Macaroni)
```

```
## [1] NA
```

Een omweg hiervoor is gebruik maken van de `summary` functie. Deze laat ook de minimumwaarde en het gemiddelde zien.

```
summary(CP$Macaroni)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.     NA's
##    0.820   0.950   1.130   1.132   1.340   1.380         1
```

De `mean` wordt in dit geval berekend zonder de NA waarde:

$$\bar{x} = \frac{\sum x_i}{n - 1}$$

Waarbij X_i alle waarden zijn in `Macaroni` zonder de NA waarde:

$$1.131538 = \frac{1.09 + 1.21 + 1.22 + 1.13 + 0.88 + 0.82 + 0.95 + 1.10 + 1.34 + 1.36 + 1.35 + 1.38}{14 - 1}$$

=

$$1.131538 = \frac{14.71}{13}$$

We krijgen de waarde NA in plaats van het gemiddelde. Gelukkig kunnen we de `mean()` functie een extra argument geven waar in we aangeven hoe om te gaan met NAs. Dit is het `na.rm` argument, dat staat voor “NA remove” en aangeeft dat we voor het berekenen van de uitkomst de NAs even weglaten (voor de calculatie even “removen”). Deze zetten we op `TRUE`.

Als we dit toepassen en de berekeningen opnieuw proberen uit te rekenen, krijgen we de volgende resultaten:

```
mean(CP$Macaroni, na.rm = TRUE)
```

```
## [1] 1.131538
```

```
min(CP$Macaroni, na.rm = TRUE)
```

```
## [1] 0.82
```

We krijgen het gewenste resultaat en weten op welke manier deze berekend is.

12.3.6 Een lege waarde (NA/missing value) vervangen door een berekende waarde (de `ifelse()` functie)

De lege waarde negeren tijdens de berekening is een manier. Wat we ook kunnen doen, is de lege waarde vervangen door een berekende waarde. Een manier is door de lege waarde te vervangen door het gemiddelde van alle andere waarden.

We kunnen hier de `ifelse()` functie voor gebruiken. De `ifelse()` functie is als volgt opgebouwd: “Als *object* gelijk is aan *deze waarde*, doe dan *dit*, doe anders *dat*”

In plaats voor “is gelijk aan” (==), kun je ook andere condities en operatoren zoals “kleiner dan” (<), “groter dan” (>) of “is niet” (!=). Een goed overzicht van de beschikbare operators in R kun je vinden in een artikel van [9] “Operators”, (Kabacoff, 2014).

bijvoorbeeld:

```
waarde1 <- 5
waarde2 <- 4

antwoord <- ifelse(waarde1 < waarde2, "Lager", "Hoger")
antwoord
```

```
## [1] "Hoger"
```

Het antwoord geeft `Hoger` terug omdat `waarde1` hoger is dan `waarde2`.

Nu we de `ifelse()` functie begrijpen, kunnen we deze gebruiken om de lege waarde in de `Macaroni` te vullen.

```
CP$Macaroni <- ifelse(is.na(CP$Macaroni), mean(CP$Macaroni, na.rm=TRUE), CP$Macaroni)
CP$Macaroni
```

```
## [1] 1.090000 1.210000 1.220000 1.130000 0.880000 0.820000 0.880000
```

```
## [8] 0.950000 1.100000 1.340000 1.360000 1.350000 1.380000 1.131538
```

We hebben het gewenste resultaat, het gemiddelde is 1.131538. In de `ifelse()` functie hebben we de volgende dingen aangegeven:

1. Zoek alle NAs op met de `is.na` functie;
2. Als je een NA tegenkomt, vervang die met de volgende berekening;

3. `mean(CP$Macaroni, na.rm=TRUE)`, waarmee het gemiddelde wordt berekend zonder de NAs;
4. Waarden die geen NAs zijn, laat die hun eigen waarden behouden.

Het eindresultaat is dat we een `CP$Macaroni` variabele hebben die de juiste datatype heeft en geen NAs meer bevat.

```
class(CP$Macaroni)
```

```
## [1] "numeric"
```

```
CP$Macaroni
```

```
## [1] 1.090000 1.210000 1.220000 1.130000 0.880000 0.820000 0.880000
## [8] 0.950000 1.100000 1.340000 1.360000 1.350000 1.380000 1.131538
```

12.3.7 Tijd besparen door herhaaldelijk werk te vervangen met functies: `function()` gebruiken in R

Om de vorige acties stuk voor stuk te doen kost veel tijd. Bijvoorbeeld als er veel meer kolommen zijn die we moeten veranderen. In deze situatie zijn het 5 variabelen waarbij we deze stappen moeten doen, dat valt nog wel te doen. Echter als het er bijvoorbeeld 50 zijn, is dat niet meer te doen en hebben we behoefte aan een mechanisme die dit herhaaldelijke werk voor ons doet. We zijn namelijk aan het programmeren dus dit moet niet raar klinken.

Voor dit probleem maken we gebruik van een van de sterkste wapens van programmeertalen, namelijk **functies**.

We gaan een functie schrijven die voor elke variabele in de CP data frame dezelfde stappen uitvoert die we zojuist hebben gedaan. Dit doen we met de `function()` functie.

12.3.7.1 Functies voorbeeld

Om eerst een beter beeld te krijgen van `function()`, gebruiken we eerst een simpel voorbeeld.

We maken een functie die 3 argumenten heeft: `getal1`, `getal2` en `getal3`. De waarden van `getal1` en `getal2` moeten worden opgeteld. Het resultaat daarvan moet worden vermenigvuldigd met de waarde van `getal3`:

```
# een rekenfunctie maken

rekenFunctie <- function(getal1, getal2, getal3){
  som <- getal1 + getal2
  som2 <- som * getal3
  som2
}
```

Het begint met het definiëren van een naam, voor de `<-`. Na de `<-` geven we aan de het een functie moet bevatten door `function` aan te geven. Tussen de `()` geven we aan welke waarden er steeds veranderen, in dit geval `getal1`, `getal2` en `getal3`. Tussen de `{ }` geven we alle handelingen aan, in dit geval de calculatie.

Nu we de functie hebben gebouwd, kunnen we deze gebruiken door er simpelweg nieuwe waarden in te voeren.

```
rekenFunctie(3,6,2)
```

```
## [1] 18
```

```
# Andere waarden
rekenFunctie(123432,13256,78653)
```

```
## [1] 10750921264
```

Het grote voordeel is dat we het stuk van `som`, `som2` etc. niet steeds hoeven in te voeren.

12.3.8 Stappen laten herhalen met een loops in r, de for functie

Voor onze functie moeten we ook een aantal zaken herhalen. Hier kunnen we een zogenaamde **loop** voor gebruiken. Het woord zegt het al, het voert een functie uit en eindigt daarna weer bij het begin (net als een looping bij een vliegtuig). Je geeft bij een loop het aantal keer aan dat die looping moet worden uitgevoerd, of wanneer het moet stoppen.

Het volgend korte voorbeeld laat zien hoe we een loop kunnen gebruiken in R. We schrijven een loop door een aantal personen te groeten.

```
namen <- c("Piet","Marco","Dirk","Arie","Rocco","Helga")

for (i in namen){
  print(paste("Hallo ", i, ", alles goed?"))
}
```

```
## [1] "Hallo Piet , alles goed?"
## [1] "Hallo Marco , alles goed?"
## [1] "Hallo Dirk , alles goed?"
## [1] "Hallo Arie , alles goed?"
## [1] "Hallo Rocco , alles goed?"
## [1] "Hallo Helga , alles goed?"
```

Je kunt de loop ook duidelijker maken door gebruik te maken van meervoud en enkelvoud.

```
namen <- c("Piet","Marco","Dirk","Arie","Rocco","Helga")

for (naam in namen){
  print(paste("Hallo ", naam, ", alles goed?"))
}
```

```
## [1] "Hallo Piet , alles goed?"
## [1] "Hallo Marco , alles goed?"
## [1] "Hallo Dirk , alles goed?"
## [1] "Hallo Arie , alles goed?"
## [1] "Hallo Rocco , alles goed?"
## [1] "Hallo Helga , alles goed?"
```

In deze loop is `naam` slechts een “placeholder”. In een for loop kunnen we neerzetten wat we willen als placeholder, bijvoorbeeld `geit`

```
namen <- c("Rosemarijn","Marco","Henk","Arie","Rocco","Helge")

for (geit in namen){
  print(paste("Heey", geit))
}
```

```
## [1] "Heey Rosemarijn"
## [1] "Heey Marco"
## [1] "Heey Henk"
## [1] "Heey Arie"
## [1] "Heey Rocco"
## [1] "Heey Helge"
```

12.3.9 Loops en functies combineren

In sommige functies willen we een aantal stappen herhalen, daar gebruiken we ook loops voor. Op deze manier bouwen we deze loops in een functie. Met een simpel voorbeeld laten we zien hoe we een functie en een loop kunnen combineren. We maken een groet functie die iedereen groet met voor en achternaam en een welkomstboodschap geeft.

```
groetFunctie <- function(namen,achternamen){
  for (i in 1:length(namen)){
    print(paste("Hallo ",namen[i]," ",achternamen[i],
               " welkom!"))
  }
}

namen <- c("Jan","Pieter","Klaas")
achternamen <- c("Jansen","Post","Vaak")

groetFunctie(namen,achternamen)
```

```
## [1] "Hallo Jan Jansen welkom!"
## [1] "Hallo Pieter Post welkom!"
## [1] "Hallo Klaas Vaak welkom!"
```

Dit werkt ook als we andere collecties van namen gebruiken:

```
deelnemers <- c("James","Willie","Jaap","Winston")
achternaampjes <- c("Bond","Wortel","De Wit","Churchill")

groetFunctie(deelnemers,achternaampjes)
```

```
## [1] "Hallo James Bond welkom!"
## [1] "Hallo Willie Wortel welkom!"
## [1] "Hallo Jaap De Wit welkom!"
## [1] "Hallo Winston Churchill welkom!"
```

Meer informatie over loops kun je vinden in een artikel van Datacamp door Carlo Fanara [10] “A Tutorial on Loops in R - Usage and Alternatives” (Fanara, 2015).

12.3.10 Een functie bouwen om waarden numeriek te maken

We bouwen nu een functie waarin alle acties worden uitgevoerd die we hebben gedaan voor de `CP$Macaroni` variabele. Het moet een functie zijn waarbij we alleen maar de naam van de variabele hoeven in te voeren. We noemen de functie `numericValues`.

```
attach(CP)
numericValues <- function(variabele){
  variabele2 <- as.numeric(as.matrix(variabele))
  variabele3 <- ifelse(is.na(variabele2),mean(variabele2,na.rm=TRUE),
                      variabele2)
  variabele3
}
class(Testament)
```

```
## [1] "factor"
```

```
Testament <- numericValues(Testament)
class(Testament)
```

```
## [1] "numeric"
```

```
Testament
```

```
## [1] 217.4600 226.2100 247.7400 260.5800 276.3900 287.8100 294.3000
## [8] 314.1500 335.0800 356.2300 370.4900 381.9300 400.5100 305.2985
```

We bekijken nu welke variabelen veranderd moeten worden in numeriek:

```
str(CP)
```

```
## 'data.frame': 14 obs. of 59 variables:
## $ Jaar : int 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 ...
## $ Benzine.Euro95 : num 1.16 1.15 1.14 1.16 1.25 1.35 1.41 1.46 1.54 1.35 ...
## $ Diesel : num 0.85 0.82 0.79 0.79 0.89 1.02 1.09 1.1 1.29 1.01 ...
## $ LPG : num 0.45 0.43 0.43 0.43 0.46 0.51 0.57 0.59 0.68 0.51 ...
## $ Frikandel : num 0.89 0.96 1.04 1.06 1.08 1.1 1.12 1.16 1.21 1.25 ...
## $ Glas.bier : num 1.28 1.41 1.54 1.6 1.62 1.67 1.71 1.79 1.86 1.94 ...
## $ Glas.rode.huiswijn : num 1.83 2.01 2.18 2.26 2.31 2.36 2.38 2.47 2.64 2.79 ...
## $ Kopje.zwarte.koffie : num 1.18 1.28 1.41 1.47 1.51 1.55 1.57 1.62 1.7 1.81 ...
## $ Pizza : num 6.25 6.42 6.85 6.97 7.02 7 7.16 7.46 7.99 8.19 ...
## $ Aardappelen : num 0.63 0.75 0.87 0.79 0.73 0.58 0.86 0.81 0.77 0.72 ...
## $ Appelmoes : num 0.41 0.45 0.53 0.58 0.53 0.49 0.47 0.61 0.75 0.81 ...
## $ Bananen : num 1.51 1.62 1.74 1.7 1.54 1.62 1.52 1.53 1.57 1.6 ...
## $ Bier : num 1.17 1.29 1.38 1.43 1.29 1.28 1.31 1.37 1.46 1.6 ...
## $ Boterhamworst : num 6.31 7.49 7.81 7.99 7.69 7.61 7.45 7.12 7.17 7.39 ...
## $ Brood : num 1.29 1.34 1.38 1.39 1.41 1.43 1.37 1.35 1.48 1.46 ...
## $ Champignons : num 0.89 0.91 0.98 0.93 0.9 0.87 0.88 0.91 0.93 0.92 ...
## $ Cornflakes : num 2.39 2.67 2.85 2.95 2.97 2.97 3.25 3.53 3.75 3.88 ...
## $ Diepvries.patates.frites : num 1.06 1.09 1.18 1.14 1.06 0.98 1.02 1.31 1.33 1.26 ...
```

```
## $ Eieren : num 1.4 1.4 1.4 1.9 1.6 1.4 1.4 1.4 1.5 1.5 ...
## $ Hagelslag : num 1.17 1.23 1.3 1.38 1.22 1.18 1.15 1.13 1.21 1.29 ...
## $ Hamlappen : num 7.63 8.81 8.12 8 7.75 7.55 7.93 7.73 7.84 8.29 ...
## $ Kaas : num 7.01 7.34 7.58 7.67 7.34 7.18 7.24 7.11 8.39 8.15 ...
## $ Kipfilet : num 7.65 8.65 8.59 8.34 7.67 7.48 7.32 7.27 7.44 7.55 ...
## $ Koffie : num 6.16 5.96 5.76 5.54 4.94 5.5 5.8 6 6.6 6.58 ...
## $ Komkommer : num 0.81 0.8 0.91 0.87 0.75 0.77 0.89 0.91 0.83 0.84 ...
## $ Koolzuurhoudend.mineraalwater : num 0.25 0.25 0.25 0.26 0.21 0.2 0.31 0.35 0.37 0.41 ...
## $ Krentenbol : num 0.25 0.26 0.27 0.29 0.29 0.29 0.3 0.31 0.32 0.33 ...
## $ Macaroni : num 1.09 1.21 1.22 1.13 0.88 0.82 0.88 0.95 1.1 1.34 ...
## $ Margarine : num 0.64 0.69 0.72 0.76 0.73 0.68 0.68 0.7 0.78 0.79 ...
## $ Mayonaise : num 0.77 0.82 0.85 0.9 0.87 0.84 0.8 0.89 1.09 1.12 ...
## $ Melk : num 0.51 0.57 0.61 0.62 0.6 0.6 0.6 0.6 0.58 0.7 0.71 ...
## $ Mineraalwater.zonder.koolzuur : num 0.29 0.36 0.36 0.31 0.29 0.28 0.27 0.27 0.29 0.28 ...
## $ Olijfolie : num 5.39 5.18 5.04 4.98 4.75 4.62 5.37 5.41 5.24 5.13 ...
## $ Pasta : num 1.49 1.65 1.66 1.54 1.2 1.12 1.2 1.3 1.51 1.84 ...
## $ Plantaardige.olie : num 1.03 1.06 1.19 1.15 1.05 0.88 0.87 0.85 1.22 1.42 ...
## $ Riblappen : num 8.65 9.27 9.4 9.01 8.59 8.6 9 9.02 9.19 9.29 ...
## $ Rijst : num 2.12 2.21 2.27 2.27 1.81 1.5 1.39 1.48 1.73 2.03 ...
## $ Roomboter : num 1.09 1.11 1.12 1.09 1.05 1.01 1.01 1.08 1.22 1.15 ...
## $ Rundergehakt : num 5.87 6.08 6.02 5.71 5.05 5.19 5.18 5.02 5.22 5.4 ...
## $ Shampoo : num 1.72 1.78 1.84 1.84 1.62 1.49 2.05 2.07 2.17 2.15 ...
## $ Sinaasappelsap : num 0.84 0.86 0.87 0.89 0.8 0.73 0.74 0.91 0.95 0.89 ...
## $ Sperziebonen.in.blik : Factor w/ 12 levels " 0.52"," 0.54",...: 3 6 6 5 5 4 1 2 7 8 ...
## $ Suiker : num 0.88 0.89 0.88 0.85 0.82 0.79 0.78 0.8 0.78 0.79 ...
## $ Tandpasta : num 2.39 2.71 2.81 2.74 2.46 2.34 2.31 2.31 2.34 2.51 ...
## $ Thee : num 1.02 1.1 1.15 1.12 0.97 0.87 0.85 0.84 0.9 0.91 ...
## $ Toilet papier : num 2.9 3.06 3.18 3.29 3.05 2.94 3 3.04 3.16 3.15 ...
## $ Tomatenketchup : num 1.55 1.61 1.64 1.72 1.8 1.81 1.84 1.88 2.07 2.16 ...
## $ Tonijn.in.blik : num 1.09 1.15 1.25 1.27 1.21 1.21 1.23 1.1 1.2 1.33 ...
## $ Vanillevla : num 0.84 0.94 0.99 0.98 0.92 0.9 0.86 0.87 0.95 0.87 ...
## $ Zelfrijzend.bakmeel : num 0.99 1.02 1.12 1.08 1.07 1.01 1.07 1.1 1.14 1.07 ...
## $ APK.keuring : Factor w/ 13 levels " 35.03"," 38.53",...: 8 11 12 10 9 7 6 5 4 3 .
## $ Autorijles : num 29.5 32.3 34.5 36.3 37.5 ...
## $ Bioscoopkaartje : num 6.72 6.99 8.1 7.35 7.56 7.8 7.99 8.14 8.28 8.5 ...
## $ Car.wash : num 7.15 7.63 8.17 8.26 8.49 8.65 8.83 8.87 9.25 9.53 ...
## $ Dameskapper : num 25.6 27 28.9 30.6 31.3 ...
## $ Herenkapper : num 13.4 14.4 15.3 16.1 16.9 ...
## $ Krant : num 1.03 1.06 1.11 1.17 1.22 1.25 1.22 1.25 1.26 1.39 ...
## $ Testament : Factor w/ 14 levels ".","217.46","226.21",...: 2 3 4 5 6 7 8 9 10 11
## $ Zwemles : num 5.23 6.51 7.43 7.92 7.88 8.55 9.05 9.33 9.43 9.92 ...
```

De variabelen `Sperziebonen.in.blik`, `APK.keuring` en `Testament` hebben de datatype `factor` en gaan we met onze zelfgemaakte `numericValues()` gebruiken.

```
class(CP$Sperziebonen.in.blik)
```

```
## [1] "factor"
```

```
CP$Sperziebonen.in.blik <- numericValues(CP$Sperziebonen.in.blik)
class(CP$Sperziebonen.in.blik)
```

```
## [1] "numeric"
```

```
CP$APK.keuring <- numericValues(CP$APK.keuring)
CP$Testament <- numericValues(CP$Testament)
str(CP)
```

```
## 'data.frame': 14 obs. of 59 variables:
## $ Jaar : int 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 ...
## $ Benzine.Euro95 : num 1.16 1.15 1.14 1.16 1.25 1.35 1.41 1.46 1.54 1.35 ...
## $ Diesel : num 0.85 0.82 0.79 0.79 0.89 1.02 1.09 1.1 1.29 1.01 ...
## $ LPG : num 0.45 0.43 0.43 0.43 0.46 0.51 0.57 0.59 0.68 0.51 ...
## $ Frikandel : num 0.89 0.96 1.04 1.06 1.08 1.1 1.12 1.16 1.21 1.25 ...
## $ Glas.bier : num 1.28 1.41 1.54 1.6 1.62 1.67 1.71 1.79 1.86 1.94 ...
## $ Glas.rode.huiswijn : num 1.83 2.01 2.18 2.26 2.31 2.36 2.38 2.47 2.64 2.79 ...
## $ Kopje.zwarte.koffie : num 1.18 1.28 1.41 1.47 1.51 1.55 1.57 1.62 1.7 1.81 ...
## $ Pizza : num 6.25 6.42 6.85 6.97 7.02 7 7.16 7.46 7.99 8.19 ...
## $ Aardappelen : num 0.63 0.75 0.87 0.79 0.73 0.58 0.86 0.81 0.77 0.72 ...
## $ Appelmoes : num 0.41 0.45 0.53 0.58 0.53 0.49 0.47 0.61 0.75 0.81 ...
## $ Bananen : num 1.51 1.62 1.74 1.7 1.54 1.62 1.52 1.53 1.57 1.6 ...
## $ Bier : num 1.17 1.29 1.38 1.43 1.29 1.28 1.31 1.37 1.46 1.6 ...
## $ Boterhamworst : num 6.31 7.49 7.81 7.99 7.69 7.61 7.45 7.12 7.17 7.39 ...
## $ Brood : num 1.29 1.34 1.38 1.39 1.41 1.43 1.37 1.35 1.48 1.46 ...
## $ Champignons : num 0.89 0.91 0.98 0.93 0.9 0.87 0.88 0.91 0.93 0.92 ...
## $ Cornflakes : num 2.39 2.67 2.85 2.95 2.97 2.97 3.25 3.53 3.75 3.88 ...
## $ Diepvries.patates.frites : num 1.06 1.09 1.18 1.14 1.06 0.98 1.02 1.31 1.33 1.26 ...
## $ Eieren : num 1.4 1.4 1.4 1.9 1.6 1.4 1.4 1.4 1.5 1.5 ...
## $ Hagelslag : num 1.17 1.23 1.3 1.38 1.22 1.18 1.15 1.13 1.21 1.29 ...
## $ Hamlappen : num 7.63 8.81 8.12 8 7.75 7.55 7.93 7.73 7.84 8.29 ...
## $ Kaas : num 7.01 7.34 7.58 7.67 7.34 7.18 7.24 7.11 8.39 8.15 ...
## $ Kipfilet : num 7.65 8.65 8.59 8.34 7.67 7.48 7.32 7.27 7.44 7.55 ...
## $ Koffie : num 6.16 5.96 5.76 5.54 4.94 5.5 5.8 6 6.6 6.58 ...
## $ Komkommer : num 0.81 0.8 0.91 0.87 0.75 0.77 0.89 0.91 0.83 0.84 ...
## $ Koolzuurhoudend.mineraalwater : num 0.25 0.25 0.25 0.26 0.21 0.2 0.31 0.35 0.37 0.41 ...
## $ Krentenbol : num 0.25 0.26 0.27 0.29 0.29 0.29 0.3 0.31 0.32 0.33 ...
## $ Macaroni : num 1.09 1.21 1.22 1.13 0.88 0.82 0.88 0.95 1.1 1.34 ...
## $ Margarine : num 0.64 0.69 0.72 0.76 0.73 0.68 0.68 0.7 0.78 0.79 ...
## $ Mayonaise : num 0.77 0.82 0.85 0.9 0.87 0.84 0.8 0.89 1.09 1.12 ...
## $ Melk : num 0.51 0.57 0.61 0.62 0.6 0.6 0.6 0.58 0.7 0.71 ...
## $ Mineraalwater.zonder.koolzuur : num 0.29 0.36 0.36 0.31 0.29 0.28 0.27 0.27 0.29 0.28 ...
## $ Olijfolie : num 5.39 5.18 5.04 4.98 4.75 4.62 5.37 5.41 5.24 5.13 ...
## $ Pasta : num 1.49 1.65 1.66 1.54 1.2 1.12 1.2 1.3 1.51 1.84 ...
## $ Plantaardige.olie : num 1.03 1.06 1.19 1.15 1.05 0.88 0.87 0.85 1.22 1.42 ...
## $ Riblappen : num 8.65 9.27 9.4 9.01 8.59 8.6 9 9.02 9.19 9.29 ...
## $ Rijst : num 2.12 2.21 2.27 2.27 1.81 1.5 1.39 1.48 1.73 2.03 ...
## $ Roomboter : num 1.09 1.11 1.12 1.09 1.05 1.01 1.01 1.08 1.22 1.15 ...
## $ Rundergehakt : num 5.87 6.08 6.02 5.71 5.05 5.19 5.18 5.02 5.22 5.4 ...
## $ Shampoo : num 1.72 1.78 1.84 1.84 1.62 1.49 2.05 2.07 2.17 2.15 ...
## $ Sinaasappelsap : num 0.84 0.86 0.87 0.89 0.8 0.73 0.74 0.91 0.95 0.89 ...
## $ Sperziebonen.in.blik : num 0.55 0.63 0.63 0.6 0.6 0.56 0.52 0.54 0.73 0.81 ...
## $ Suiker : num 0.88 0.89 0.88 0.85 0.82 0.79 0.78 0.8 0.78 0.79 ...
## $ Tandpasta : num 2.39 2.71 2.81 2.74 2.46 2.34 2.31 2.31 2.34 2.51 ...
## $ Thee : num 1.02 1.1 1.15 1.12 0.97 0.87 0.85 0.84 0.9 0.91 ...
## $ Toilet papier : num 2.9 3.06 3.18 3.29 3.05 2.94 3 3.04 3.16 3.15 ...
## $ Tomatenketchup : num 1.55 1.61 1.64 1.72 1.8 1.81 1.84 1.88 2.07 2.16 ...
## $ Tonijn.in.blik : num 1.09 1.15 1.25 1.27 1.21 1.21 1.23 1.1 1.2 1.33 ...
```

```
## $ Vanillevla           : num  0.84 0.94 0.99 0.98 0.92 0.9 0.86 0.87 0.95 0.87 ...
## $ Zelfrijzend.bakmeel  : num  0.99 1.02 1.12 1.08 1.07 1.01 1.07 1.1 1.14 1.07 ...
## $ APK.keuring          : num  59 80.3 80.3 78.2 60.9 ...
## $ Autorijles           : num  29.5 32.3 34.5 36.3 37.5 ...
## $ Bioscoopkaartje      : num  6.72 6.99 8.1 7.35 7.56 7.8 7.99 8.14 8.28 8.5 ...
## $ Car.wash             : num  7.15 7.63 8.17 8.26 8.49 8.65 8.83 8.87 9.25 9.53 ...
## $ Dameskapper          : num  25.6 27 28.9 30.6 31.3 ...
## $ Herenkapper          : num  13.4 14.4 15.3 16.1 16.9 ...
## $ Krant                : num  1.03 1.06 1.11 1.17 1.22 1.25 1.22 1.25 1.26 1.39 ...
## $ Testament            : num  217 226 248 261 276 ...
## $ Zwemles              : num  5.23 6.51 7.43 7.92 7.88 8.55 9.05 9.33 9.43 9.92 ...
```

Er kan natuurlijk ook gebruik gemaakt worden van een loop, zoals we dat behandeld hebben in het vorige gedeelte. Hier zijn heel veel manieren voor.

12.4 De correlatiematrix maken

Nu alle variabelen in de CP data frame numeriek zijn, kunnen we de correlatiematrix maken. Hiervoor kun je ook eenvoudig de `cor()` functie gebruiken.

```
Correlatiematrix <- cor(CP)

# met het volgende commando schrijf je een matrix of data frame naar een Excel-bestand.

write.table(Correlatiematrix,"Correlatiematrix.xls")
```

Het bestand wordt uiteindelijk opgeslagen in dezelfde map als waar dit R project is ingesteld. Om het pad naar dit bestand terug te halen, maak je gebruik van het `getwd()` command:

```
getwd()

## [1] "/Users/arietwigt/Dropbox/Data-analyseren-met-R-master"
```

12.5 Samenvatting

12.5.1 Correlaties

Correlaties geven een mate van samenhang weer. Deze samenhang kan zwak of sterk zijn en positief of negatief zijn. De correlatie wordt berekend aan de hand van een statistische formule, deze draait achter de schermen van de `cor()` functie in R. Bij het analyseren van data maak je gebruik van correlaties om te bekijken tussen welke verschillende variabelen er iets speelt. De volgende stap die je kunt nemen is verder kijken naar het verband, bijvoorbeeld onderzoek op welke manieren variabelen elkaar beïnvloeden. In R kun je echter alleen maar correlaties berekeningen met vectoren van het `numeric` of `integer` datatype.

12.5.2 Datatypes

Zoals je misschien al eerder gemerkt hebt tijdens het werken met R, kom je soms in de knoop met de verschillende datatypes. In dit hoofdstuk heb je kunnen zien dat dit zelfs kan gebeuren op momenten dat je het niet verwacht. Het is namelijk logisch om ervan uit te gaan dat een dataset met consumentenprijzen alleen maar kolommen bevat met numerieke waarden. Data-kwaliteit issues zoals lege waarden kunnen dit echter verpesten. Gelukkig zijn er genoeg handige manieren in R om met deze onhandigheden om te gaan, van het handmatig omzetten van een waarde tot het gebruik maken van handige functies.

12.5.3 Functies en loops

Functies zijn zeer krachtige functies in R. Door gebruik te maken van functies kun je namelijk veel herhaaldelijk werk besparen. Daarbij zijn functies eigenlijk de belangrijkste bouwstenen R. Alle packages in R zijn namelijk niets meer dan verzamelingen van functies die iemand anders een keer heeft ontwikkeld en beschikbaar heeft gesteld. Zo kun je ook tijdens je eigen R-sessie functies opslaan en deze later gebruiken. Net als vectoren en waarden, staan functies in RStudio in het rechtervenster onder **Functions** waar ook **Data** en **Values** staan. Ik kan je alleen maar aanmoedigen om zo veel mogelijk gebruik te maken van functies. Niet alleen om programmeren en data analyseren met R in de vingers te krijgen, maar ook voor programmeren in het algemeen. Functies, en het opslaan en distribueren hiervan, komen namelijk in iedere programmeertaal terug.

12.6 Oefeningen

12.6.0.1 Schrijf een functie die automatisch een percentage zoals “1.25 %” (`character`) omzet naar een fractie (`numeric`) zoals 0.0125

12.6.0.2 Maak een loop of functie die de prijzen van een product omzet in dollars door gebruik te maken van de wisselkoers en er een dollarteken voor het bedrag plaatst

13 Lineair regressiemodel

Met een lineaire regressie kun je net als met de correlatie de samenhang tussen variabelen vinden. Bij een regressieanalyse krijg je echter wel een formule waarbij je toekomstige uitkomsten over een variabele kunt voorspellen. Bij een regressieanalyse zit het hem vooral in de interpretatie van de uitkomst.

In dit hoofdstuk beperken we ons tot de eenvoudige lineaire regressieanalyse. Dit betekent dat er een variabele wordt voorspeld aan de hand van een andere variabele. De voorspelde variabele wordt de afhankelijke variabele genoemd. De variabele waarmee de afhankelijke variabele voorspeld wordt, heet de verklarende variabele.



Figure 23:

13.1 De dataset

Voor dit hoofdstuk worden twee varianten van een dataset gebruikt: `Projecten` en `Projecten2`. Je kunt deze datasets eenvoudig importeren als csv-bestand. In hoofdstuk 3 paragraaf 14 (*3.14 Datasets*) van dit boek kun je zien welke datasets er gebruikt worden voor dit boek en via welke link je deze kunt downloaden (<http://r-handleiding.blogspot.nl/p/voorbeeld-databestanden-downloaden.html>).

13.2 Een lineair regressie model maken met de `lm()` functie

Met de `lm()` functie kun je in R een lineair model maken. De functie vul je als volgt: `lm(<AFHANKELIJKE VARIABLE> ~ <VERKLARENDE VARIABLE>, <NAAM VAN DE DATASET>)`. Je kunt elke numerieke variabele gebruiken voor de regressie. Bijvoorbeeld:

```
attach(Projecten)

lm(TevredenheidKlant ~ AfstandKlant, Projecten)

##
## Call:
## lm(formula = TevredenheidKlant ~ AfstandKlant, data = Projecten)
##
## Coefficients:
## (Intercept) AfstandKlant
##      2.524811      -0.003409
```

De `lm()` functie geeft een formule terug.

13.2.1 Het geschatte model

Om het lineaire model te kunnen beoordelen, kun je een object maken van het model. In het onderstaande voorbeeld wordt deze bijvoorbeeld `model1` genoemd.

```
model1 <- lm(TevredenheidKlant ~ AfstandKlant, Projecten)
```

In dit voorbeeld wordt de tevredenheid van de klant verklaard met de afstand die tussen het bedrijf en de klant is. R geeft aan de hand van de regressie de volgende formule: $TevredenheidKlant = 2.524811 + -0.003409x$, waarbij x staat voor het aantal kilometers. Deze formule is een schatting van het werkelijke model om *TevredenheidKlant* te kunnen voorspellen.

13.3 De regressieanalyse uitvoeren

Met de `summary()` functie krijg je uitgebreide informatie over het zojuist getrainde model. Hierbij kijken we eerst naar het gedeelte dat onder de tekst *Coefficients* staat.

```
summary(model1)
```

```
##
## Call:
## lm(formula = TevredenheidKlant ~ AfstandKlant, data = Projecten)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.3339 -0.4396 -0.1447  0.5578  1.5831
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.524811   0.089066  28.348  < 2e-16 ***
## AfstandKlant -0.003409   0.000964  -3.536  0.000622 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6238 on 98 degrees of freedom
## Multiple R-squared:  0.1132, Adjusted R-squared:  0.1041
## F-statistic: 12.51 on 1 and 98 DF,  p-value: 0.0006216
```

Bij `Intercept` en `Afstand.Klant` zie je dezelfde waarden als in het geschatte model staan. Dit zijn de coëfficiënten. Dit interpreteer je als: “Voor iedere stijging van x , stijgt de waarde y als aangegeven in de betreffende coëfficiënt”.

Het regressiemodel analyseren kan op verschillende manieren. Je ziet dat de output van de `summary()` functie veel informatie geeft. Ieder stuk informatie is bruikbaar. Als je dieper in de wereld van statistiek duikt, zal je steeds meer onderdelen van de output herkennen en kunnen lezen. Voor deze handleiding houden we het in ieder geval nog even simpel en kijken we alleen naar de coëfficiënten, de significantie en de Adjusted R squared.

13.3.1 Significantie

Achter de rijen staan 3 sterren (***) weergegeven. De sterren geven aan dat de betreffende variabele significant is, dit wil zeggen dat het voor deze regressie van belang is dat deze variabele wordt gebruikt. In statistische termen: “De variabele X reageert significant op variabele Y ”. Een aanduiding met één ster is al genoeg om aan te tonen dat de variabele significant is. Des te meer sterren achter de rij staan, des te meer significantie de variabele heeft. Als er achter de betreffende variabele geen sterren staan, is de variabele niet significant en voegt het dus niet veel toe aan het voorspellingsmodel. De variabele is dan niet verklarend genoeg in de regressie.

13.3.2 Voorspellingskracht van de regressie / Adjusted R-squared

In het resultaat vind je ook de Adjusted R-squared. Deze geeft de voorspellingskracht van het geschatte model aan. Het percentage dat het geschatte model juist voorspelt wordt achter de Adjusted R-squared weergegeven in decimalen. Een Adjusted R-squared van 1 betekent dus dat 100% van de voorspellingen die met het geschatte model gedaan worden juist zijn. Een Adjusted R-squared van 0,5 betekent dat 50% van de voorspellingen die met het geschatte model gedaan worden juist zijn.

Het is aan de gebruiker zelf om te beoordelen, aan de hand van *Adjusted R-squared*, of het geschatte model goed genoeg is. Er zijn verschillende interpretaties over de hoogte van de *Adjusted R-squared*. Bij het huidige voorbeeld is de *Adjusted R-squared* 0.1041. Dit betekent dat in **10,41%** van de gevallen het geschatte model de tevredenheid van de klant juist voorspelt.

We trainen een volgende lineair voorspellingsmodel. Hier wordt er naar een voorspellingsmodel gezocht om de opbrengst van het project te voorspellen met het aantal uren dat er in is gestoken. Hiervoor wordt de volgende code gebruikt:

```
model2 <- lm(OpbrengstProject~UrenProject, Projecten)
```

Door code `model2()` in te voeren, komt het volgende geschatte model tevoorschijn:

```
model2
```

```
##
## Call:
## lm(formula = OpbrengstProject ~ UrenProject, data = Projecten)
##
## Coefficients:
## (Intercept)  UrenProject
##           0           40
```

Hierin is x het aantal uren dat er in het project wordt gestoken. Met de `summary(model2)` komen de extra gegevens van het model tevoorschijn.

```
summary(model2)
```

```
## Warning in summary.lm(model2): essentially perfect fit: summary may be
## unreliable

##
## Call:
## lm(formula = OpbrengstProject ~ UrenProject, data = Projecten)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -9.831e-12  4.410e-14  1.541e-13  2.764e-13  1.693e-12
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept)  0.000e+00  4.127e-13  0.000e+00      1
## UrenProject  4.000e+01  1.930e-15  2.072e+16 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.321e-12 on 98 degrees of freedom
## Multiple R-squared:  1, Adjusted R-squared:  1
## F-statistic: 4.294e+32 on 1 and 98 DF, p-value: < 2.2e-16
```

Daarin is te zien dat alleen de variabele `UrenProject` significant is (Intercept kan net zo goed worden weggehaald omdat deze 0 is). Verder is te zien dat de *Adjusted R-squared* gelijk aan 1 is. Dat wil zeggen dat met het geschatte model 100% betrouwbaar is. Dit is logisch want blijkbaar rekent het bedrijf 40 per uur en stijgt de opbrengst met 40 voor elk uur dat er in het project gestoken wordt. > **Perfecte lineaire regressie** Aan de waarschuwing die R geeft kun je zien dat het model de waarde van Y perfect voorspelt. Dit klopt omdat in dit geval de Y variabele gebouwd is op de X variabele. Deze situatie kom je zelden tegen maar geeft de bevestiging dat als een variabele een andere variabele perfect voorspelt, het model dit gewoon laat zien.

13.4 Meervoudige Regressieanalyse

In het vorige hoofdstuk is besproken hoe je een eenvoudige lineaire regressie met R kunt maken. Hierbij wordt er een afhankelijke variabele voorspeld met één verklarende variabele. Echter kun je ook een afhankelijke variabele voorspellen met meerdere verklarende variabelen.

Bij R is er weinig verschil in code tussen een enkelvoudige regressie en een meervoudige regressie. Je kunt zo veel verklarende variabelen gebruiken als u wilt, zolang u ze maar met een + in de code blijft toevoegen. Op dezelfde manier als bij een enkelvoudige regressie kunt u het geschatte model en de gegevens van de regressie tevoorschijn halen.

13.4.1 Interpretatie van de regressieanalyse

In tegenstelling tot de enkelvoudige regressie, staan er in het resultaat van de meervoudige regressie meerdere rijen van variabelen. Het principe is hetzelfde, aan de hand van de sterren achter de rij worden de significanties van de variabelen getoond. In het volgende voorbeeld wordt het volgende voorspellingsmodel gemaakt: De winst voorspellen aan de hand van personeelskosten, materiaalkosten en de tevredenheid van de klant. Hiervoor wordt het volgende command gebruikt:

```
model3 <- lm(Winst~Personeelskosten+Materiaalkosten+TevredenheidKlant, Projecten)
```

Door de naam van de regressie Regressie in te toetsen, verschijnt het geschatte model:

$$Winst = 8845.8805 + (-0.2964)X1 + (-1.9292)X2 + (-1338.5787)X3.$$

Waarbij X1 staat voor de *personeelskosten*, X2 voor de *materiaalkosten* en X3 voor de *tevredenheid van de klant*.

```
summary(model3)
```

```
##
## Call:
## lm(formula = Winst ~ Personeelskosten + Materiaalkosten + TevredenheidKlant,
##     data = Projecten)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4869.0 -1123.5    15.4  1281.8  6511.4
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8845.8805   943.3177   9.377 3.24e-15 ***
## Personeelskosten    -0.2964    0.1038  -2.855 0.005278 **
## Materiaalkosten    -1.9292    0.9469  -2.037 0.044364 *
## TevredenheidKlant -1338.5787   360.4634  -3.713 0.000343 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2046 on 96 degrees of freedom
## Multiple R-squared:  0.2756, Adjusted R-squared:  0.2529
## F-statistic: 12.17 on 3 and 96 DF,  p-value: 8.094e-07
```

Valt het je op dat alle rijen van variabelen significant zijn? De een is echter minder significant dan de ander, maar toch zijn alle variabelen significant genoeg om in het model toe te passen. In het voorbeeld is de

Adjusted R-squared slechts 0.2529. De drie verklarende variabelen zijn dus slechte variabelen om de winst mee te voorspellen. U hebt kunnen zien dat een meervoudige regressie op dezelfde manier werkt als een enkelvoudige regressie.

Let op! Het begrip voorspellen moet u interpreteren als voorspellen in uw eigen dataset. Uw dataset is een afspiegeling van uw eigen situatie, niet hoe het er in alle situaties op elk moment in de wereld aan toe gaat. Het blijven namelijk maar statistische functies die verbanden in uw data analyseren. U bent met de regressieanalyse dus beperkt uitspraken te doen over voorspellingen in uw eigen situatie.

13.5 Categorische variabelen

In de vorige twee paragrafen is er een regressieanalyse gedaan met numerieke variabelen, variabelen met een numerieke waarde. Het kan wel eens voor komen dat je in een databestand ook wel eens niet-numerieke variabelen of waarden tegenkomt. Deze worden categorische variabelen genoemd. Dit zijn variabelen die namen of categorieën bevatten.

13.5.1 Het voorbeeldbestand

Bij de laatste twee hoofdstukken wordt er in de voorbeelden gebruik gemaakt van het databestand `Projecten2.csv`. Je kunt zien dat het dezelfde data bevat als het bestand `Projecten.csv`. Echter zijn eraan dit bestand 3 nieuwe, categorische, variabelen toegevoegd:

- *Werkgroep*(A,B of C);
- *Maand* (januari t/m december);
- *TypeProject*(Alfa, Beta, Gamma, Delta).

De code die gebruikt wordt voor een regressieanalyse met categorische variabelen verschilt niet veel met de regressie met alleen numerieke variabelen. Het verschil is dat de categorische variabele in de code omgezet moet worden in een numerieke variabele.

In het volgende voorbeeld wordt de afhankelijke variabele `Winst` voorspelt met de verklarende variabele `Werkgroep`. Omdat `werkgroep` een categorische variabele is en er voor een regressieanalyse met numerieke waarden gerekend moet worden, moet de categorische variabele `Werkgroep` omgezet worden in een numerieke variabele.

```
detach(Projecten)
```

```
attach(Projecten2)
```

```
## The following object is masked from CDNOW.df:
```

```
##
```

```
##      Maand
```

```
# voorproefje van de Projecten2 dataset
```

```
str(Projecten2)
```

```
## 'data.frame':   100 obs. of  11 variables:
```

```
##  $ Nummer           : int  20131 20132 20133 20134 20135 20136 20137 20138 20139 20140 ...
```

```
##  $ TevredenheidKlant: int   3 2 3 2 3 3 2 3 2 1 ...
```

```
## $ AfstandKlant      : int  50 125 36 25 12 23 56 23 21 86 ...
## $ UrenProject       : int  100 200 200 300 200 100 150 100 150 300 ...
## $ Materiaalkosten   : int  987 645 789 546 788 987 546 878 879 132 ...
## $ PersoneelsKosten  : int  2312 4654 5654 6786 8456 2515 3571 5641 1325 6511 ...
## $ OpbrengstProject  : int  4000 8000 8000 12000 8000 4000 6000 4000 6000 12000 ...
## $ Winst             : int   701 2701 1557 4668 -1244 498 1883 -2519 3796 5357 ...
## $ Type.Project      : Factor w/ 3 levels "A","B","C": 1 3 2 3 1 1 2 1 2 3 ...
## $ Maand             : Factor w/ 12 levels "april","augustus",...: 5 4 8 1 9 7 6 2 12 11 ...
## $ Werkgroep         : Factor w/ 4 levels "Alfa","Beta",...: 1 2 4 3 3 2 1 3 2 1 ...
```

```
model4 <- lm(Winst~factor(Werkgroep), Projecten2)
```

```
model4
```

```
##
## Call:
## lm(formula = Winst ~ factor(Werkgroep), data = Projecten2)
##
## Coefficients:
##             (Intercept)  factor(Werkgroep)Beta  factor(Werkgroep)Delta
##                4178.0                -1589.6                -3522.9
## factor(Werkgroep)Gamma
##                250.5
```

Door `model4` in te toetsen wordt het geschatte model gepresenteerd. Het geschatte model bij deze regressieanalyse is: $\text{Winst} = 1479.1 + (0 \text{ A}) + (1987 \text{ B}) + (4097.1 \text{ C})$.

Hierbij wordt alleen de waarde opgeteld voor de betreffende werkgroep. Als de winst dus geschat moet worden als *Werkgroep C* het project doet, is de berekening: $1479.1 + 1987 = 3.466.1$. Door `summary(model4)` in te voeren, krijg je de formule van het model te zien:

```
summary(model4)
```

```
##
## Call:
## lm(formula = Winst ~ factor(Werkgroep), data = Projecten2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4595.5 -1305.6   -78.6  1236.9  4925.6
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      4178.0      360.4  11.593 < 2e-16 ***
## factor(Werkgroep)Beta -1589.6      554.0  -2.869  0.00506 **
## factor(Werkgroep)Delta -3522.9      624.2  -5.644 1.68e-07 ***
## factor(Werkgroep)Gamma  250.5      497.9   0.503  0.61608
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1974 on 96 degrees of freedom
## Multiple R-squared:  0.3256, Adjusted R-squared:  0.3045
## F-statistic: 15.45 on 3 and 96 DF,  p-value: 2.826e-08
```


Je kunt zien dat `factor()` in het command van de regressie wordt geplaatst. Het kan gezien worden als een kleine toevoeging aan de categorische variabele.

13.5.2 Meer categorische variabelen

Een regressieanalyse uitvoeren met meerdere categorische variabelen gaat volgens hetzelfde principe als bij een enkelvoudige regressieanalyse met categorische variabelen.

Het command gaat volgens het principe: `<NAAM VAN DE REGRESSIE> <-lm(<AFHANKELIJKE VARIABELE> ~ factor(<VERKLARENDE VARIABELE 1>)+ as.numeric(<VERKLARENDE VARIABELE 2>)+ factor(<VERKLARENDE VARIABELE X...>), VARIABELE VAN HET BESTAND)`.

```
model5 <- lm(Winst ~ factor(Werkgroep) + as.numeric(Type.Project), Projecten2)
```

Door `model5` in te voeren verschijnt het geschatte model: $Winst = 2706 + (0 A) + (1123 B) + (3274 C) + (-1121 Beta) + (-2562 Delta) + (-140 Gamma)$. Ook in dit model met categorische variabelen wordt alleen de waarde opgeteld als de categorie van toepassing is. Als bijvoorbeeld de winst voorspeld moet worden als werkgroep C type project Gamma gaat doen, is de voorspelde winst $2706 + 3274 - 140 = 5840$.

Je vraagt je misschien af waarom de waarde bij A nul is. Deze zit al in de Intercept (1479) verwerkt.

13.5.3 Categorische variabelen bij de regressieanalyse gebruiken.

Door de `summary()` functie te gebruiken, wordt de regressieanalyse gepresenteerd.

```
summary(model5)
```

```
##
## Call:
## lm(formula = Winst ~ factor(Werkgroep) + as.numeric(Type.Project),
##     data = Projecten2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2625.6 -1172.1   -65.4   1011.9   6122.3
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)       777.29     521.67   1.490   0.1395
## factor(Werkgroep)Beta -1031.12     441.53  -2.335   0.0216 *
## factor(Werkgroep)Delta -2316.16     514.81  -4.499 1.93e-05 ***
## factor(Werkgroep)Gamma  -38.72     393.28  -0.098   0.9218
## as.numeric(Type.Project) 1645.50     211.94   7.764 9.43e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1552 on 95 degrees of freedom
## Multiple R-squared:  0.5874, Adjusted R-squared:  0.57
## F-statistic: 33.81 on 4 and 95 DF, p-value: < 2.2e-16
```


Aan de rijen van variabelen kunt u zien dat niet alle variabelen even significant zijn. In het voorbeeld is de rij met de factor **Gamma** bijvoorbeeld niet significant. Dit betekent dat de voorspellingen die worden gedaan als Gamma erbij wordt betrokken, niet erg betrouwbaar zijn. Het is niet erg om de factor **Gamma** in de formule te laten staan, maar de voorspellingen die worden gedaan met betrekking tot **Gamma** zijn niet betrouwbaar. De *Adjusted R-squared* van 0.5759 laat zien dat de kwaliteit van het geschatte model redelijk hoog is.

13.6 Samenvatting

13.6.1 Een lineair model trainen

Met de `lm()` functie kun je een lineair model trainen in R. Wat je hiermee doet is onderzoeken in hoeverre een X -variabele in staat is om een Y variabele te voorspellen. Hiermee kun je bijvoorbeeld prijzen voorspellen of de invloed van een variabele op een andere variabele goed onderzoeken.

13.6.2 Een lineair model beoordelen

Er zijn verschillende manieren waarop je een lineair model kunt beoordelen. Je kunt individueel de voorspellende X -variabelen bekijken door naar de significantie en de coëfficiënten te kijken. Het model in het algemeen kun je beoordelen door naar de *R-Squared* of naar de *Adjusted R-squared* te kijken. Voor een model met meerdere variabelen is het advies om naar de *Adjusted R squared* te kijken. De *R-squared* is de correlatie tussen de X en Y en geeft antwoord op de vraag: *‘‘In hoeverre is de variantie van X in staat om de variantie van Y te verklaren?’’.

13.6.3 Een lineair model interpreteren

De uitkomst van een lineair model is een formule en een beoordeling. De formule kun je invullen door de coëfficiënten te vermenigvuldigen met de X -en. Dit resulteert in een kwantitatieve uitkomst, namelijk een numerieke waarde. Dit kan bijvoorbeeld een prijs of iets dergelijks zijn. Door naar de coëfficiënten te kijken, krijg je inzicht in de invloed van de variabelen en kun je de invloeden van de variabelen vergelijken.

13.7 Oefeningen

Maak bij deze oefeningen gebruik van de *Projecten* en *Projecten2* datasets.

13.7.0.0.1 Maak een voorspellingsmodel die Winst voorspelt op basis van UrenProject

13.7.0.0.2 Onderzoek in welke maand over het algemeen de minste Winst wordt gemaakt (los van het feit hoe betrouwbaar het model is)

14 Uitgebreide visualisaties maken in R met ggplot

R staat o.a. bekend om zijn uitgebreide mogelijkheden om data te visualiseren. Een voorbeeld hiervan is visualiseren met grafieken. In hoofdstuk 9 hebben we kunnen zien hoe we met R standaard grafieken kunnen maken. Echter zijn er meer mogelijkheden in R om te visualiseren, vooral door gebruik te maken van aanvullende packages.

Het veruit meest bekende package van R om data op een zeer effectieve manier te visualiseren om er waardevolle inzichten uit te halen is het ggplot2 package van Hadley Wickam. De twee “G’s” in ggplot2 staan voor “Grammar of graphics”, de grammatica (of het gesprek) van afbeeldingen. Dit is de goede beschrijving voor dit package omdat ggplot2 visualisaties namelijk echt uitnodigen voor een gesprek. Ook kun je, mits je de data goed voorbereid, veel inzichten halen uit visualisaties met ggplot2.

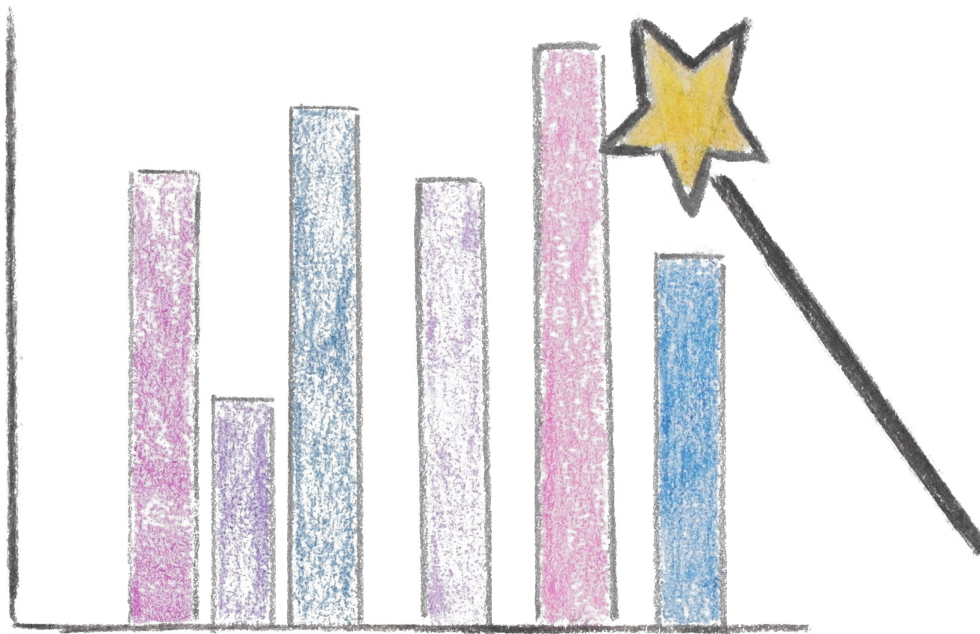


Figure 24:

14.1 Het ggplot2 package installeren

Het `ggplot2` package is gewoon op dezelfde manier te installeren als andere R packages, `ggplot2` staat namelijk ook gewoon in de CRAN repository. Daarom is het mogelijk om de `install.packages()` functie te gebruiken om het package te installeren.

```
# het ggplot2 package installeren vanaf CRAN
install.packages('ggplot2')
```

14.2 Visualiseren in R met ggplot

Het sterkste punt van `ggplot` is dat je meerdere dimensies in een enkele visualisatie kunt plaatsen. Waarbij je bij een standaard grafiek meestal slechts twee dimensies hebt, namelijk de x-as en de y-as, zijn er in `ggplot` veel meer mogelijkheden. Naast de x- en y-as als dimensies kun je namelijk ook door middel van kleur, grootte en vorm andere dimensies toevoegen.

Zo'n uitgebreide visualisatie kun je stap voor stap opbouwen.

```
# installeer het package met het onderstaande command als je dat nog niet gedaan hebt:
install.packages("ggplot2")
```

Open het package:

```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'mtcars':
##
##      mpg
```

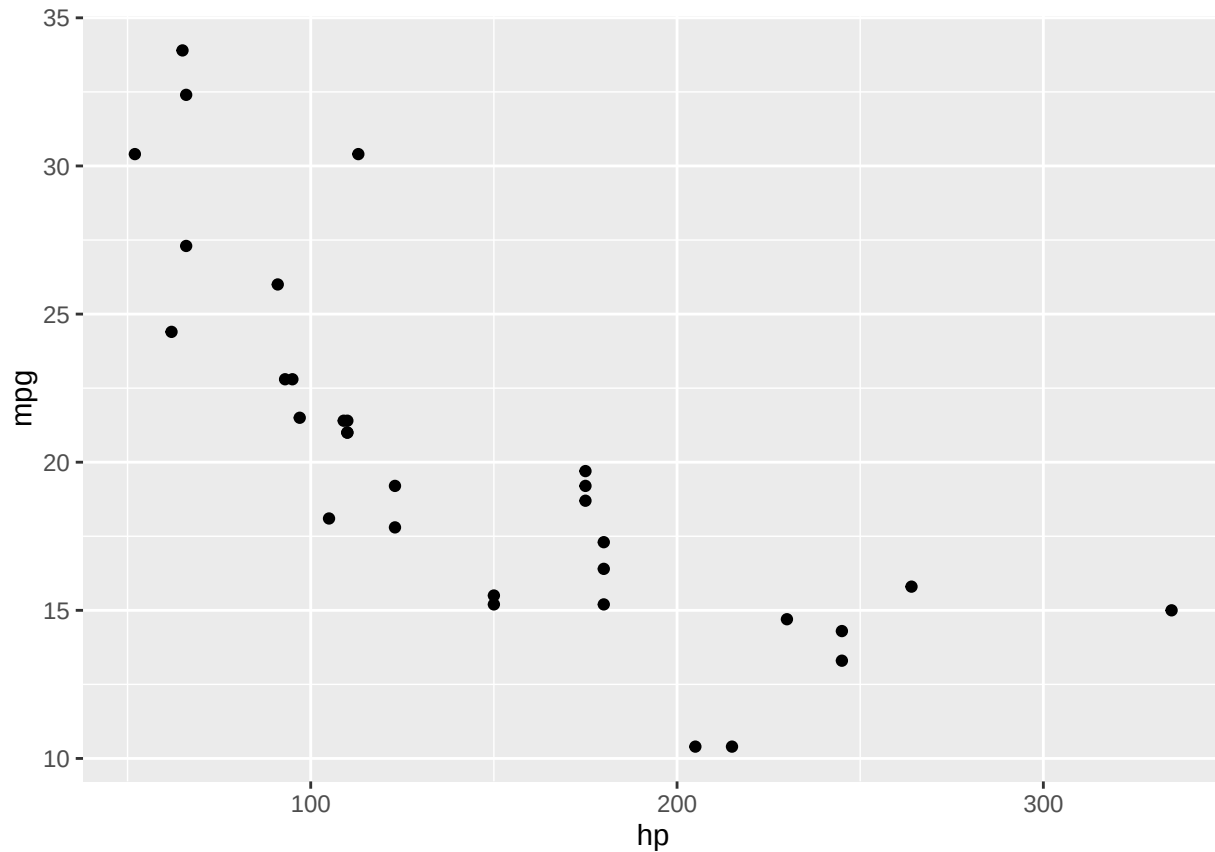
14.2.1 De basis-assen van ggplot

Voor het gemak gebruiken we de oude vertrouwde `mtcars` dataset die we eerder in dit boek zijn tegengekomen.

```
# maak de grafiek, alleen nog zonder data
mtcarsPlot <- ggplot(mtcars, aes(hp, mpg))
```

14.2.2 Visualiseer de datapunten met `geom_point()`

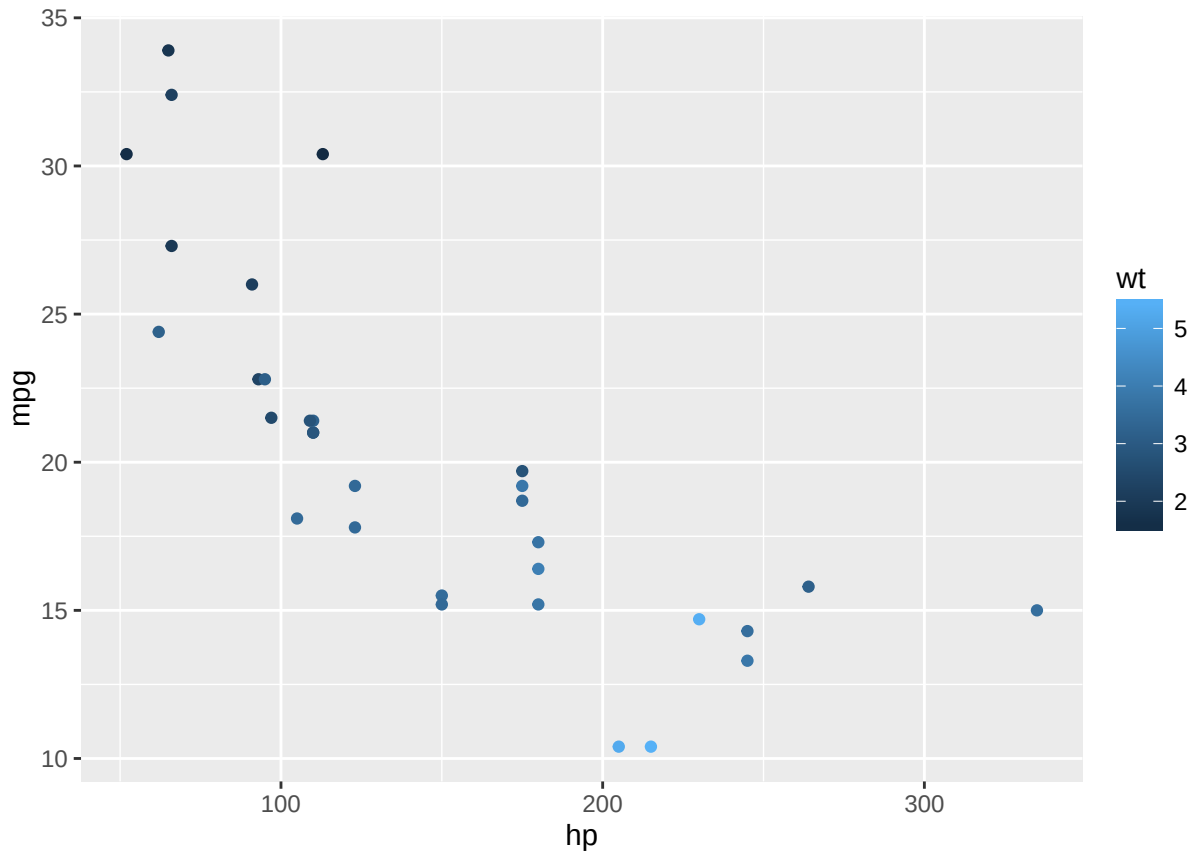
```
#plaats punten op de grafiek
mtcarsPlot + geom_point()
```



14.3 De *kleur* dimensie

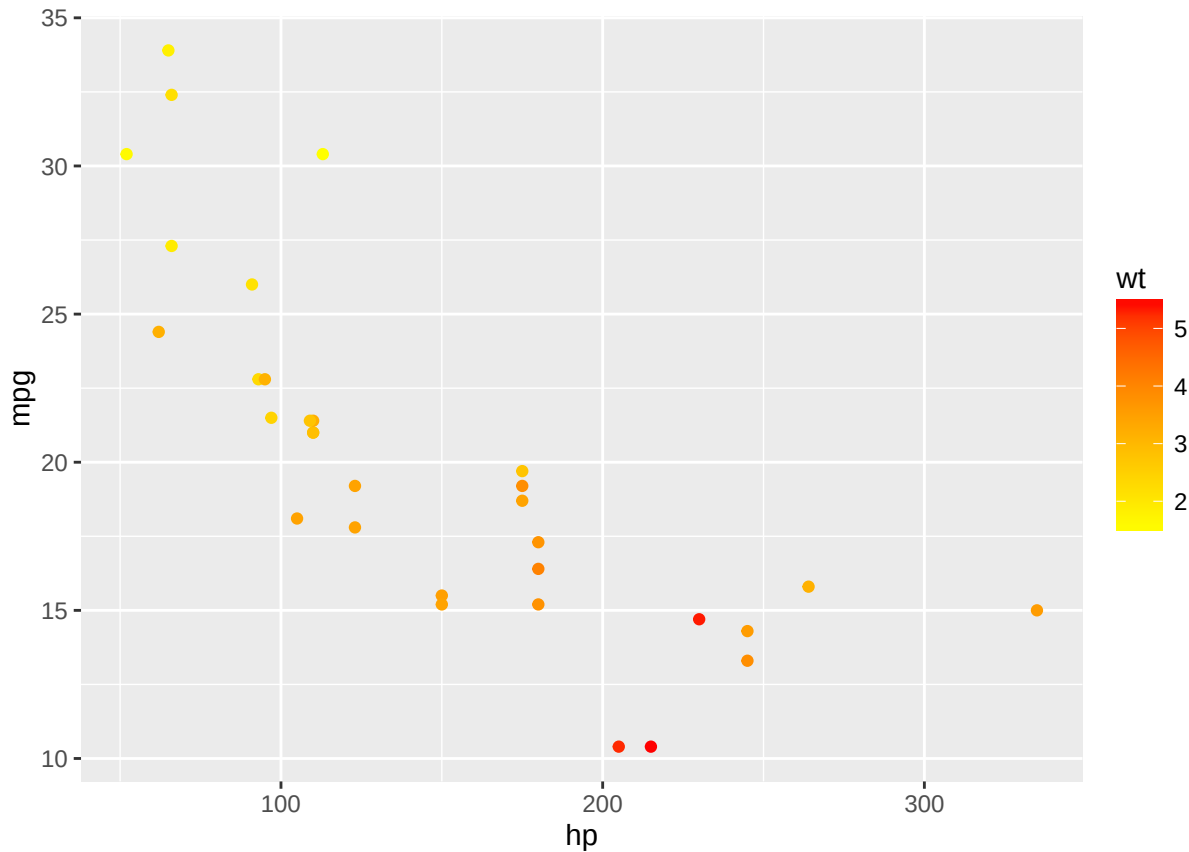
De punten zijn nu geplot. Echter willen we nu een derde dimensie toevoegen. Naast het weergeven van hp en mpg willen we nu ook het gewicht van de auto laten zien. Deze derde dimensie kunnen we creëren als we op basis van kleur het gewicht van de auto laten zien. Kleur toevoegen doen we door in de `geom_point()`-functie vullen met onze voorkeuren.

```
mtcarsPlot <- ggplot(mtcars,aes(hp,mpg))  
mtcarsPlot + geom_point(aes(colour=wt))
```



Op basis van de kleur zie het gewicht van de auto. Hoe zwaarder de auto is, des te lichter blauw het puntje is. Misschien is dit geen handige manier qua kleuren om dit te visualiseren. Daar kunnen we ook voor andere kleuren kiezen. Met de `scale_colour_gradient` functie kunnen we aangeven hoe we onze kleuren willen indelen. In het volgende script geven we aan dat we een overvloeiing van geel naar rood willen hebben op basis van het gewicht van de auto.

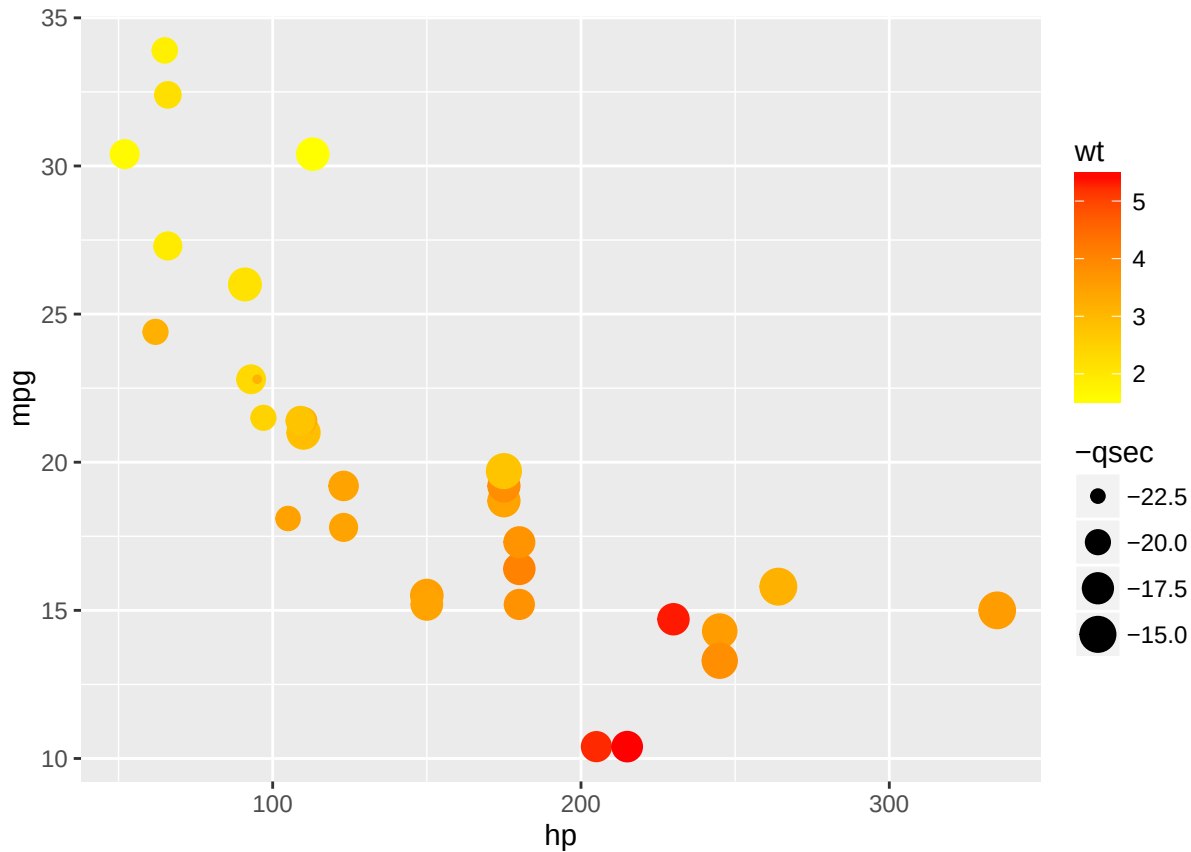
```
mtcarsPlot + geom_point(aes(colour = wt)) + scale_colour_gradient(low = "yellow",  
  high = "red")
```



14.4 De *grootte* dimensie

Naast kleur kunnen we nog een volgende dimensie toevoegen, namelijk de grootte van de punten. Hiervoor voegen we een nieuw argument toe in `geom_point()`, namelijk `size`. We voegen de kolom `qsec` toe (acceleratie/snelheid) die op basis van grootte deze waarde aangeeft. Aangezien `qsec` aangeeft binnen hoeveel tijd een kwart mijl (1/4 mile) wordt afgelegd (minder is beter), zetten we een `-` voor `qsec` zodat de punten groter worden als de auto sneller is. Het script ziet er dan als volgt uit:

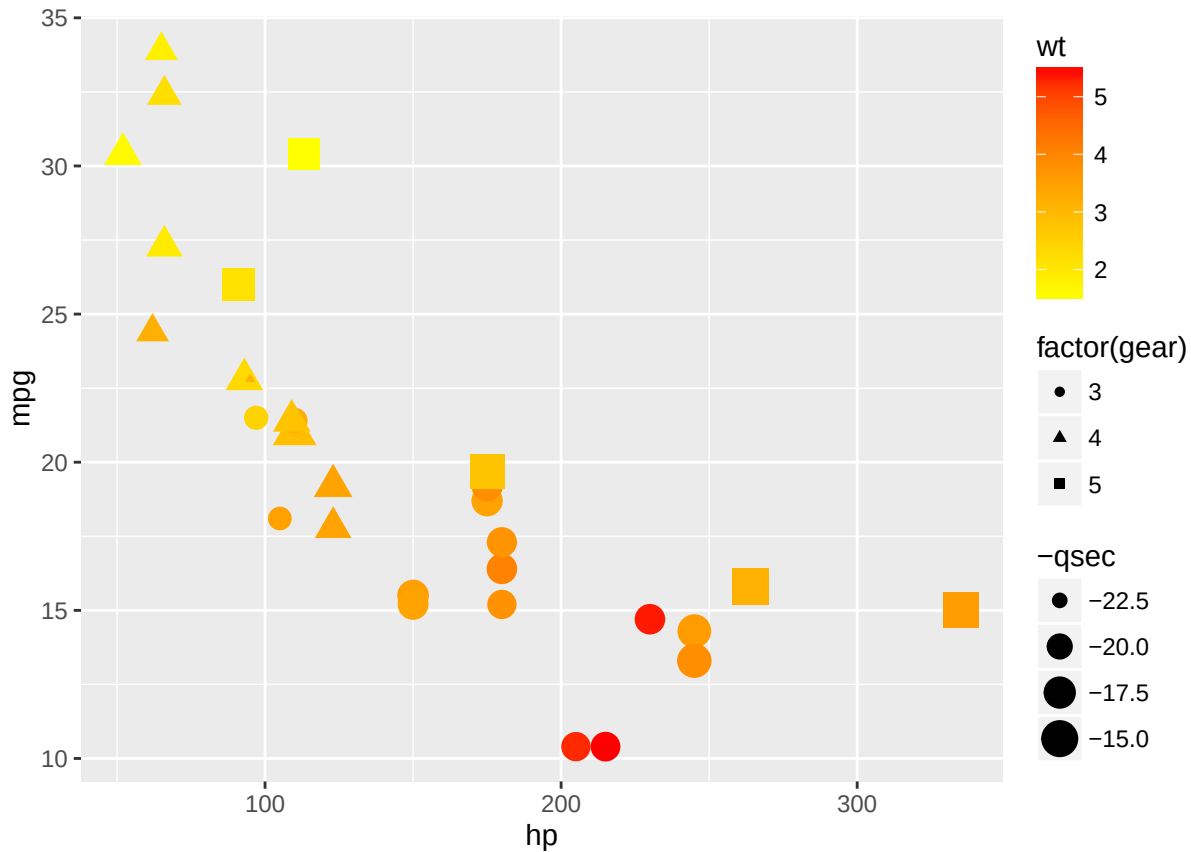
```
mtcarsPlot + geom_point(aes(colour = wt, size = -qsec)) + scale_colour_gradient(low = "yellow",
  high = "red")
```



14.5 Voeg een extra dimensie toe op basis van vorm in *ggplot*

Ten slotte kunnen we nog een dimensie toevoegen, namelijk de vorm van de datapunten. De datapunten worden nu aangegeven in ronde punten, echter kunnen we daar variatie aan geven. Een mooie variabele om als dimensie te gebruiken in vormen is de **gear** variabele. De **gear** variabele geeft aan hoeveel versnellingen de auto heeft. Dit doen we door het commando als volgt aan te passen: **shape = factor(gear)** wordt toegevoegd. Het commando ziet er dan als volgt uit. Na het script wordt onderbouwd waarom er **factor** wordt gebruikt.

```
mtcarsPlot + geom_point(aes(colour = wt, size = -qsec, shape = factor(gear))) +
  scale_colour_gradient(low = "yellow", high = "red")
```



Je kunt zien dat de meeste auto's 3 versnellingen hebben, dit wordt aangegeven met de balletjes. De allersnelste auto met de meeste paardenkracht heeft 5 versnellingen.

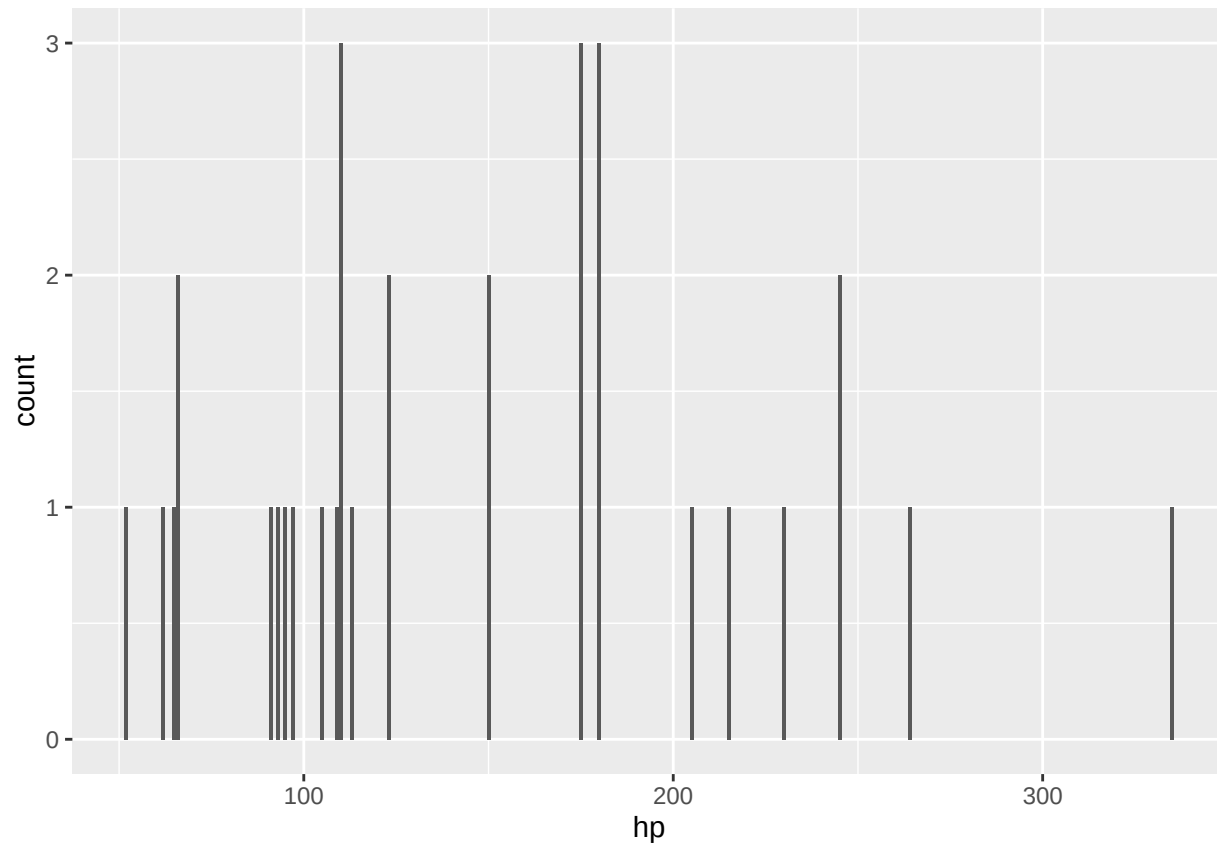
Er wordt gebruikt gemaakt van **factor**. Factor is een datatype die tussen de **numeric** en **character** datatypes inzit. Het kan het best uitgelegd worden als een numerieke waarde die als **character** datatype behandeld wordt, of andersom: een character waarde die als numeriek (qua volgorde) wordt behandeld.

14.6 Andere visualisaties in ggplot

Naast een grafiek met datapunten, kan er in *ggplot2* ook gebruik gemaakt worden van andere visualisaties zoals staafdiagrammen, lijndiagrammen en boxplots. Hieronder vindt je een aantal voorbeelden:

14.6.1 Staafdiagram

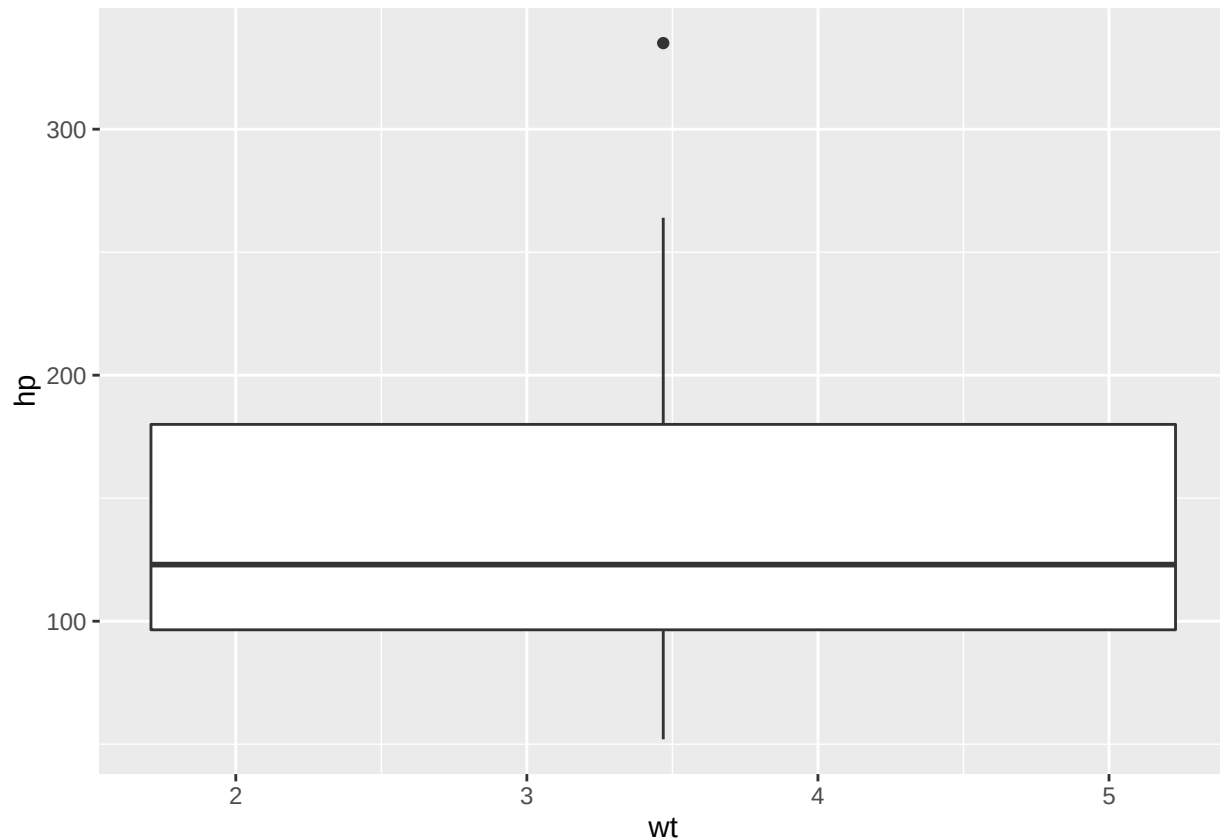
```
#Staafdiagram
mtcarsPlot2 <- ggplot(mtcars, aes(hp))
mtcarsPlot2 + geom_bar()
```

14.6.2 Boxplot

```
#Boxplot
mtcarsPlot3 <- ggplot(mtcars,aes(wt,hp))
mtcarsPlot3 + geom_boxplot()
```

```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



14.7 Samenvatting

14.7.1 Het ggplot package

Naast de ingebouwde functies in R om data te visualiseren, is het zeker de moeite waard om jezelf te verdiepen in het *ggplot2* package. Niet alleen ziet *ggplot* er iets mooier uit, kun je fantastische visualisaties maken als de functies goed in de vingers krijgt. Daarbij kun je in *ggplot* zo goed als alle grafiek-typen creëren als bij de standaard R functies.

14.7.2 Meerdere dimensies in ggplot

Een van de sterkste punten in *ggplot* is dat je extra dimensies kunt weergeven in een grafiek. In dit hoofdstuk zijn demonstraties gegeven met eenvoudige scatter plots. Echter heb je kunnen zien dat je ook andere visualisaties kunt toepassen, waarbij het ook mogelijk is om extra dimensies toe te passen. Extra dimensies toepassen in *ggplot* is een zeer krachtige manier om de data te laten spreken. Afhankelijk van de data, zijn waardevolle inzichten goed te vinden in een *ggplot* visualisatie.

14.7.3 Grafiek-typen in ggplot

Naast de eenvoudige scatter plots in de demonstraties in dit hoofdstuk, zijn er veel meer grafiek-typen mogelijk in *ggplot* waarmee je helemaal los kunt. Ieder grafiektype komt voor een bepaalde situatie het beste tot zijn recht. Voor een handig overzicht van de mogelijke visualisaties, kun je terecht op de documentatie-pagina van *ggplot*: [11] “Help topics”. Naast de documentatie is er een erg handige *cheat sheet* beschikbaar: [12] Cheat sheet ggplot.

Data Camp heeft een uitstekende cursus waarmee je ggplot goed in de vingers kunt krijgen. [13]
“Data Visualization with ggplot2”.

14.8 Oefeningen

14.8.0.0.1 Creëer een scatterplot (zoals in de demonstraties) waarmee een goed inzicht in de volgende dimensies kan worden gegeven:

*Het gewicht van de auto; De paardenkracht van de auto; De zuinigheid van de auto; De acceleratie van de auto; *Het aantal cylinders van de auto.*

Aan de hand van de visualisatie kun je een auto vinden die zeer snel optrekt (hoge `qsec`) maar slechts weinig paardenkracht (`hp`) heeft. Natuurlijk wil je weten welke auto dit is, dat gaan we uitzoeken in de volgende oefening.

14.8.0.0.2 Visualiseer de namen van de auto's in de zojuist gemaakte grafiek. Hiermee kun je achterhalen welke eigenschappen bij welke auto's horen.

*Maak gebruik van het [14] **Textual annotations**- hoofdstuk uit de documentatie van ggplot om er achter te komen hoe je gebruik kan maken van labels.*

15 Case 2: Open data ontsluiten

In deze case leer je o.a. hoe je een externe dataset kunt ontsluiten. Dit wordt ook in Hoofdstuk 5 - Data importeren en exporteren behandeld. Echter wordt er in deze case voor een aanpak gekozen die iets verschilt van de methode in het hoofdstuk. In deze case wordt namelijk data door middel van een API in json formaat naar de R-sessie geïmporteerd. Op deze data worden de basics van dataverkenning toegepast, gevolgd door een iets diepere statistische analyse en afgesloten met een set aan waardevolle visualisaties met ggplot. Veel succes!

15.1 Importeren van de dataset via de API van *Open data overheid*

15.1.1 De API van het Dataportaal van de Nederlandse overheid

De data wordt ontsloten door middel van de API die het [15] **Dataportaal van de Nederlandse overheid** aanbiedt. Een API heeft een zogenaamde *API-key*, dit is een link waarmee je identificeert welke dataset je wilt ontsluiten.

15.1.2 JSON-formaat

Data bestaat in verschillende formaten. Zo hebben we eerder in dit boek gezien dat data in *csv* formaat wordt geïmporteerd. Als we kijken naar API's, zijn er over het algemeen twee formaten waarmee je data uit een API kunt halen. Dit zijn *XML* en *JSON*.

Er zijn tegenwoordig veel meer formaten die ontsloten kunnen worden via een API. Geo-locatie data is ook steeds meer aanwezig in de API's. Met dit formaat kun je data op kaarten plotten.

Voor deze case maken we gebruik van het *JSON* formaat. *JSON* staat voor *Javascript Object Notation*, *JSON* data is namelijk een verzameling van *Javascript* objecten. *Javascript* objecten worden genoteerd in *key-value* pairs. Dit is een ander soort datastructuur dan die wij over het algemeen kennen uit bijvoorbeeld een *csv*-bestand of tabel in een database. Om deze reden behoort *JSON* tot een datastructuur die ook wel **NoSQL** wordt genoemd. *NoSQL* data kun je het best beschrijven als data die niet in het formaat van een normale tabel is gestructureerd en je dus niet met een *SQL*-query op zou kunnen halen.

NoSQL moet je niet verwarren met de term **ongestructureerde data**. *NoSQL* data is over het algemeen wel degelijk gestructureerd.

In normale tabellen wordt elk record gewoon weergegeven in een rij. In *JSON* wordt het echter weergegeven als een object. Een voorbeeld van een record in *JSON* formaat is:

```
"auto": {
  "Merk" : "Mercedes",
  "Brandstof" : "Diesel"
}
```

Een korte introductie over **JSON** [16] kun je vinden op json-schema.org, hier wordt uitgelegd hoe de *JSON* variant van een database schema er uit ziet.

15.1.3 JSON data en *lists* in R

R kan prima met *JSON* data omgaan, echter blijft het geen *JSON* formaat in de R-sessie. R ziet *JSON* data namelijk als *lists*! Eerder in dit boek hebben we kunnen zien dat *lists* in R ook als *key-value* pairs kunnen worden genoteerd, net als bij *JSON* data. Dit is een aspect dat ervoor zorgt dat *Javascript* en *R* uitstekend kunnen samenwerken!

Voor nu genoeg theorie over API's en *JSON* data. We vervolgen de case met de dataset over **Pompprijzen brandstoffen; per dag** door middel van de API van het Dataportaal van de Nederlandse overheid is geïmporteerd.

15.2 De API met JSON-data ontsluiten met het *jsonlite* package

We importeren de dataset in R. Hiervoor gebruiken we het *jsonlite* package:

```
library(jsonlite)

brandstofData <- fromJSON(
  "http://opendata.cbs.nl/ODDataApi/ODData/80416ned/UntypedDataSet"
)
```

De datastructuur in R van de geïmporteerde *JSON* dataset is een *list* in R:

```
# de structuur van het object 'brandstofData' weergeven

class(brandstofData)
```

```
## [1] "list"
```

```
# een voorproefje weergeven van de dataset
head(brandstofData)
```

15.3 De dataset bewerken

We maken een data frame van *brandstofData*.

```
brandstofDF <- as.data.frame(brandstofData)
```

De eerste kolom hebben we niet nodig, daarom maken we een nieuw object dat deze kolom niet heeft.

```
brandstof <- brandstofDF[,c(2:6)]
```

We hebben de tabel die we willen, echter zijn de namen nog een beetje ongelukkig. Laten we dat veranderen:

```
names(brandstof) = c("Value", "Periode", "Benzine", "Diesel", "LPG")
```

We evalueren de klassen van de *brandstof* data frame:

```
str(brandstof)
```

```
## 'data.frame':  4020 obs. of  5 variables:
## $ Value   : int  0 1 2 3 4 5 6 7 8 9 ...
## $ Periode: chr  "20060101" "20060102" "20060103" "20060104" ...
## $ Benzine: chr   " 1.325" " 1.328" " 1.332" " 1.348" ...
## $ Diesel  : chr   " 1.003" " 1.007" " 1.007" " 1.020" ...
## $ LPG     : chr   " 0.543" " 0.542" " 0.540" " 0.550" ...
```

Alle variabelen, behalve *Value*, zijn van het *character* datatype. Aangezien dit prijzen zijn, willen we dat natuurlijk niet omdat we anders niet kunnen rekenen. Daarom gaan we dit converteren.

```
# 'Periode' converteren van "character" naar "Date"
brandstof$Periode<-as.Date(brandstof$Periode, "%Y%m%d")

# de klassen van 'Benzine', 'Diesel' en 'LPG' omzetten naar numerieke waarden
class(brandstof$Benzine)<-"numeric"

class(brandstof$Diesel)<-"numeric"

class(brandstof$LPG)<-"numeric"

# het resultaat bekijken
str(brandstof)
```

```
## 'data.frame':  4020 obs. of  5 variables:
## $ Value   : int  0 1 2 3 4 5 6 7 8 9 ...
## $ Periode: Date, format: "2006-01-01" "2006-01-02" ...
## $ Benzine: num  1.32 1.33 1.33 1.35 1.35 ...
## $ Diesel  : num  1 1.01 1.01 1.02 1.02 ...
## $ LPG     : num  0.543 0.542 0.54 0.55 0.55 0.549 0.555 0.55 0.554 0.552 ...
```

Uiteindelijk hebben we de gewenste klassen.

15.4 Extra, berekende, kolommen toevoegen aan de data frame

Om meer inzichten te krijgen, kunnen we extra kolommen toevoegen aan de *brandstof* data frame die berekend zijn op basis van de waarden van andere kolommen. We voegen de volgende kolommen toe:

- Voor elke brandstof-type, het prijsverschil per dag in eurocenten vergeleken met de vorige dag;
- Namen van de dagen;
- Namen van de maanden;
- Kwartalen
- Indicatie of het een werkdag of dag in het weekend was.

15.5 Prijsverschillen per dag

Met de `diff()` functie kunnen we de verschillen tussen rijen berekenen. Per brandstof-type maak ik een kolom aan die de prijsverschillen voor die betreffende brandstof per dag weergeeft. Er wordt een 0 toegevoegd om dezelfde lengte te hebben met de dataframe en omdat dag 1 geen verschil heeft ten opzichte van een vorige dag.

```
brandstof$verschilBenzine <- c(0, diff(brandstof$Benzine))
brandstof$verschilDiesel <- c(0, diff(brandstof$Diesel))
brandstof$verschilLPG <- c(0, diff(brandstof$LPG))

head(brandstof)
```

```
##   Value   Periode Benzine Diesel   LPG verschilBenzine verschilDiesel
## 1      0 2006-01-01   1.325  1.003 0.543           0.000           0.000
## 2      1 2006-01-02   1.328  1.007 0.542           0.003           0.004
## 3      2 2006-01-03   1.332  1.007 0.540           0.004           0.000
## 4      3 2006-01-04   1.348  1.020 0.550           0.016           0.013
## 5      4 2006-01-05   1.347  1.021 0.550          -0.001           0.001
## 6      5 2006-01-06   1.353  1.023 0.549           0.006           0.002
##   verschilLPG
## 1          0.000
## 2         -0.001
## 3         -0.002
## 4          0.010
## 5          0.000
## 6         -0.001
```

```
str(brandstof)
```

```
## 'data.frame':   4020 obs. of  8 variables:
##  $ Value      : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Periode    : Date, format: "2006-01-01" "2006-01-02" ...
##  $ Benzine    : num  1.32 1.33 1.33 1.35 1.35 ...
##  $ Diesel     : num  1 1.01 1.01 1.02 1.02 ...
##  $ LPG        : num  0.543 0.542 0.54 0.55 0.55 0.549 0.555 0.55 0.554 0.552 ...
##  $ verschilBenzine: num  0 0.003 0.004 0.016 -0.001 ...
##  $ verschilDiesel : num  0 0.004 0 0.013 0.001 ...
##  $ verschilLPG   : num  0 -0.001 -0.002 0.01 0 -0.001 0.006 -0.005 0.004 -0.002 ...
```

15.6 Namen van de weekdays weergeven

Omdat ik wil weten op welke dag een bepaalde datum viel, heb ik een functie nodig die dit voor mij kan uitrekenen. Standaard heeft R deze functie niet. Gelukkig is er wel een package dat dit voor mij kan oplossen, het *lubridate* package. Dit package heeft de `weekdays()` functie waar ik naar op zoek ben.

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##      date
```

```
brandstof$dag <- weekdays(brandstof$Periode)

str(brandstof)
```

```
## 'data.frame':    4020 obs. of  9 variables:
##  $ Value          : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Periode        : Date, format: "2006-01-01" "2006-01-02" ...
##  $ Benzine        : num  1.32 1.33 1.33 1.35 1.35 ...
##  $ Diesel         : num  1 1.01 1.01 1.02 1.02 ...
##  $ LPG            : num  0.543 0.542 0.54 0.55 0.55 0.549 0.555 0.55 0.554 0.552 ...
##  $ verschilBenzine: num  0 0.003 0.004 0.016 -0.001 ...
##  $ verschilDiesel : num  0 0.004 0 0.013 0.001 ...
##  $ verschilLPG    : num  0 -0.001 -0.002 0.01 0 -0.001 0.006 -0.005 0.004 -0.002 ...
##  $ dag            : chr  "zondag" "maandag" "dinsdag" "woensdag" ...
```

15.7 Weekdagen in goede volgorde zetten

De *dag* kolom in de *brandstof* dataframe heeft het *character* datatype. Als ik straks wil visualiseren, worden deze dagen niet in de goede volgorde gesorteerd. R heeft namelijk geen idee van de betekenis van de kolommen, ook geen weekdagen. Daarom moet ik zelf de volgorde bepalen, dit doe ik door gebruik te maken van de *factor* klasse.

```
brandstof$dag <- factor(brandstof$dag, levels = c("maandag", "dinsdag", "woensdag", "donderdag", "vrijdag", "zaterdag", "zondag"))
```

Hetzelfde doe ik voor de maanden, kwartalen en jaren.

```
brandstof$maanden <- months(brandstof$Periode)
brandstof$maanden <- factor(brandstof$maanden,
                             levels = c("januari", "februari", "maart",
                                           "april", "mei", "juni", "juli",
                                           "augustus", "september", "oktober",
                                           "november", "december"))

brandstof$kwartalen <- quarters(brandstof$Periode)
brandstof$kwartalen <- factor(brandstof$kwartalen,
                              levels = c("Q1", "Q2", "Q3", "Q4"))

brandstof$jaar <- year(brandstof$Periode)
```

15.8 Weergeven of het een weekday of een dag in het weekend was

Nu ik per rij de weekdagen heb, kan ik door middel van de *ifelse()* functie een kolom maken die op basis van de weekdagen aangeeft of het een *weekday* was of een *weekend* -dag was.

```
brandstof$weekend <- ifelse(brandstof$dag == "zaterdag" |
                           brandstof$dag == "zondag", "weekend", "weekday")
```


15.9 Visualisaties in ggplot

15.9.1 Prijsverschillen van benzine per dag

```
benzinePlot <- ggplot(brandstof,aes(Periode,Benzine)) +
  geom_point(aes(colour = verschilBenzine, size=verschilBenzine)) +
  scale_colour_gradient(low="yellow", high="red") + theme(legend.position="none")

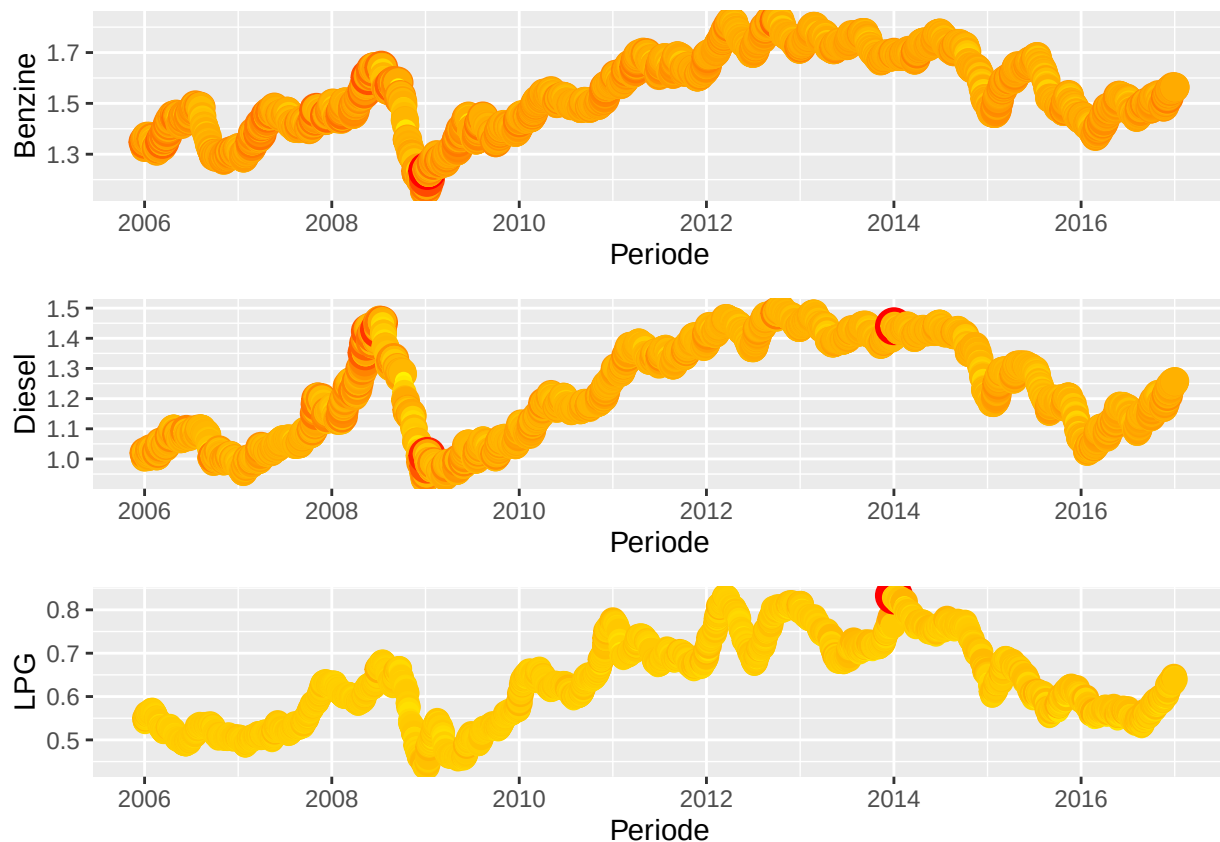
dieselPlot <- ggplot(brandstof,aes(Periode,Diesel)) +
  geom_point(aes(colour = verschilDiesel, size=verschilDiesel)) +
  scale_colour_gradient(low="yellow", high="red")+ theme(legend.position="none")

LPGPlot <- ggplot(brandstof,aes(Periode,LPG)) +
  geom_point(aes(colour = verschilLPG, size=verschilLPG)) +
  scale_colour_gradient(low="yellow", high="red")+ theme(legend.position="none")

require(easyGgplot2)
```

```
## Loading required package: easyGgplot2
```

```
ggplot2.multiplot(benzinePlot,dieselPlot,LPGPlot,cols=1)
```



Met het **easyGgplot2** package kan ik meerdere ggplots in dezelfde visualisatie plaatsen.

15.9.2 Prijzen per dag - boxplots

```
benzineDagBoxplot <- ggplot(brandstof,aes(dag,Benzine)) +
  geom_boxplot(aes(fill = dag)) +
  theme(axis.text.x = element_blank(),legend.position="none") +
  ylim(0.5, 1.8)

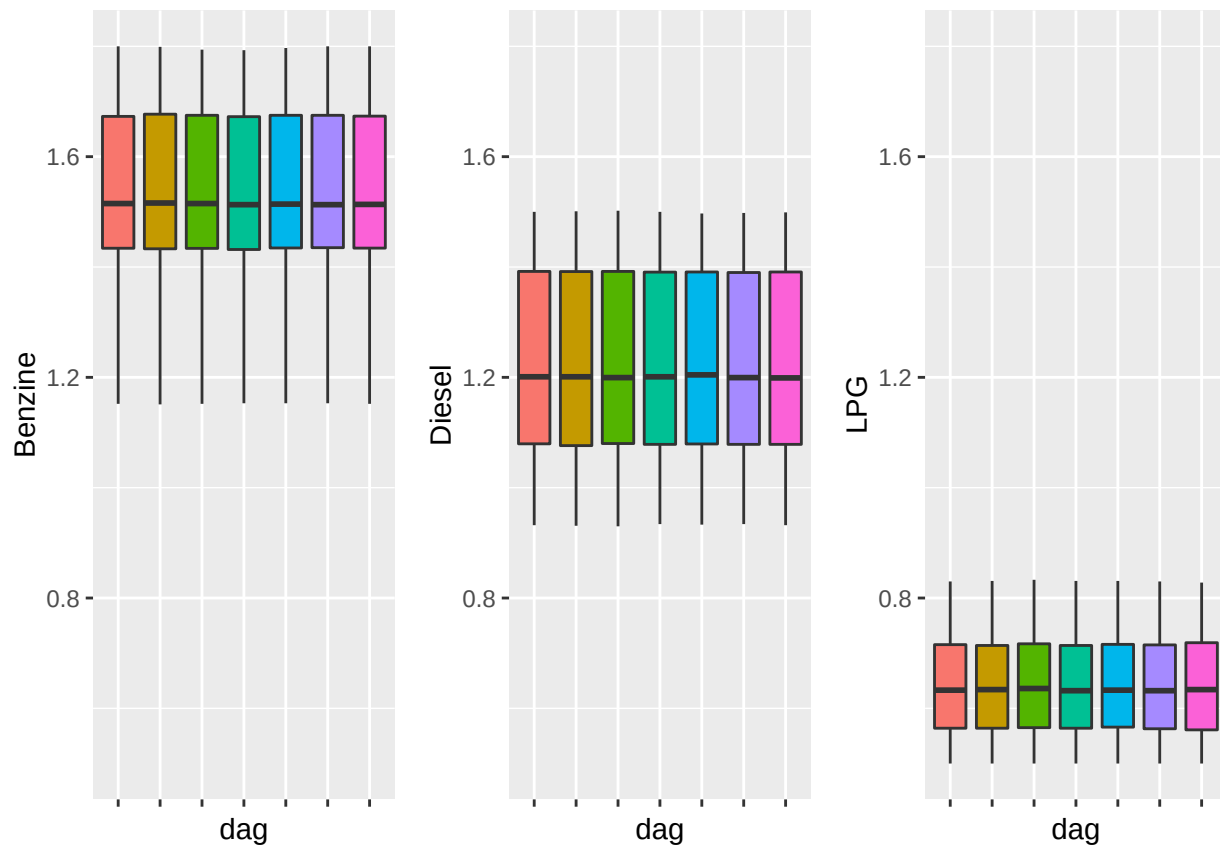
dieselDagBoxplot <- ggplot(brandstof,aes(dag,Diesel)) +
  geom_boxplot(aes(fill = dag)) +
  theme(axis.text.x = element_blank(),legend.position="none") +
  ylim(0.5, 1.8)

LPGDagBoxplot <- ggplot(brandstof,aes(dag,LPG)) +
  geom_boxplot(aes(fill = dag)) +
  theme(axis.text.x = element_blank(),legend.position="none") +
  ylim(0.5, 1.8)

ggplot2::multiplot(benzineDagBoxplot,dieselDagBoxplot,LPGDagBoxplot,cols=3)
```

```
## Warning: Removed 96 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 251 rows containing non-finite values (stat_boxplot).
```



15.9.3 Prijzen per dag - Jitter plot

```

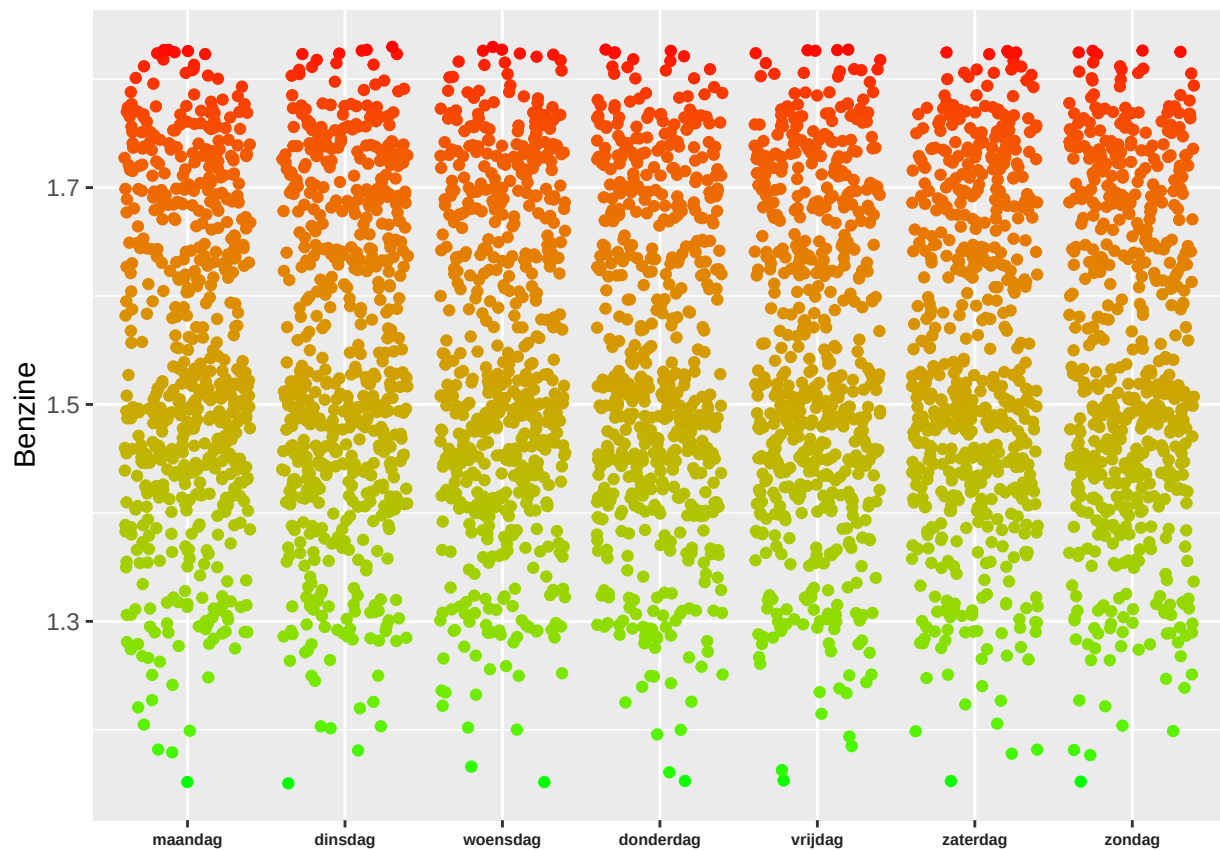
benzineDagJitter <- ggplot(brandstof,aes(dag,Benzine)) +
  geom_jitter(aes(color = Benzine)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",
        axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

dieselDagJitter <- ggplot(brandstof,aes(dag,Diesel)) +
  geom_jitter(aes(color = Diesel)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

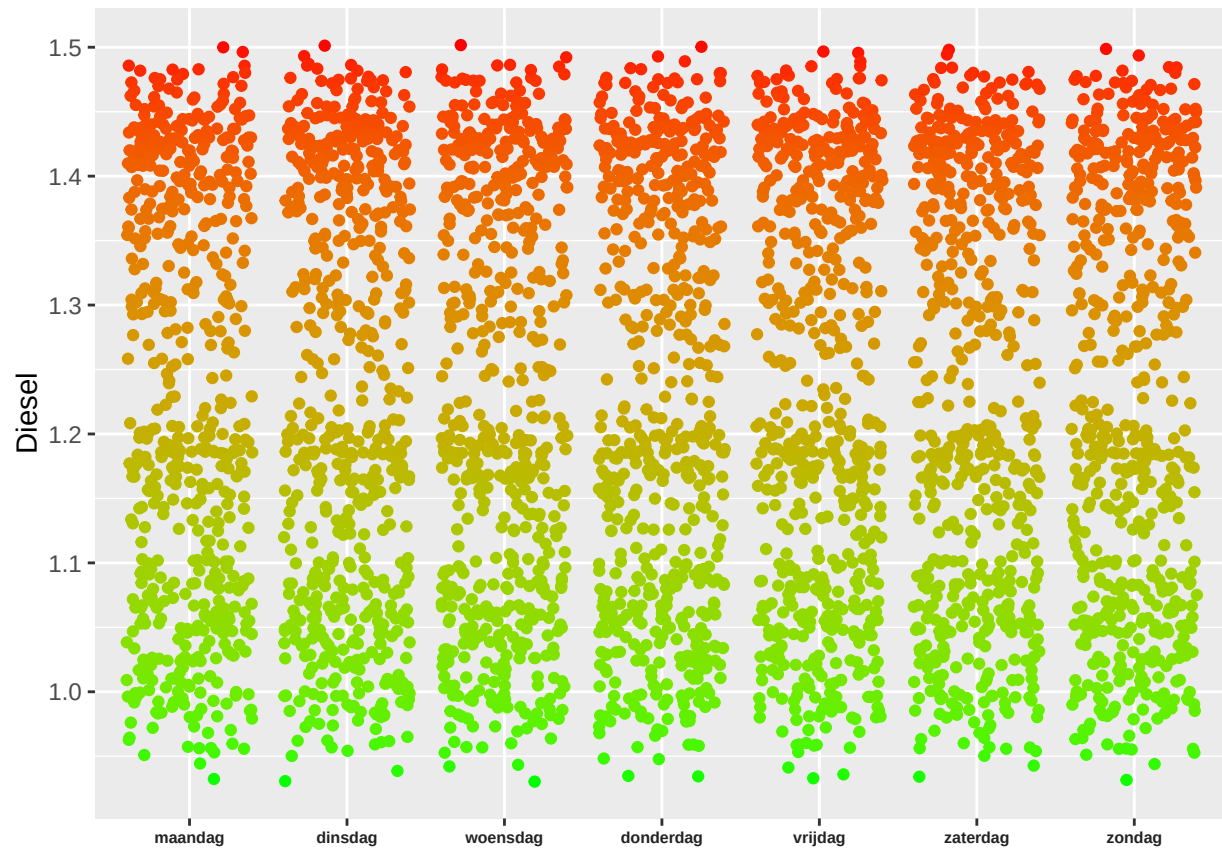
LPGDagJitter <- ggplot(brandstof,aes(dag,LPG)) +
  geom_jitter(aes(color = LPG)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

benzineDagJitter

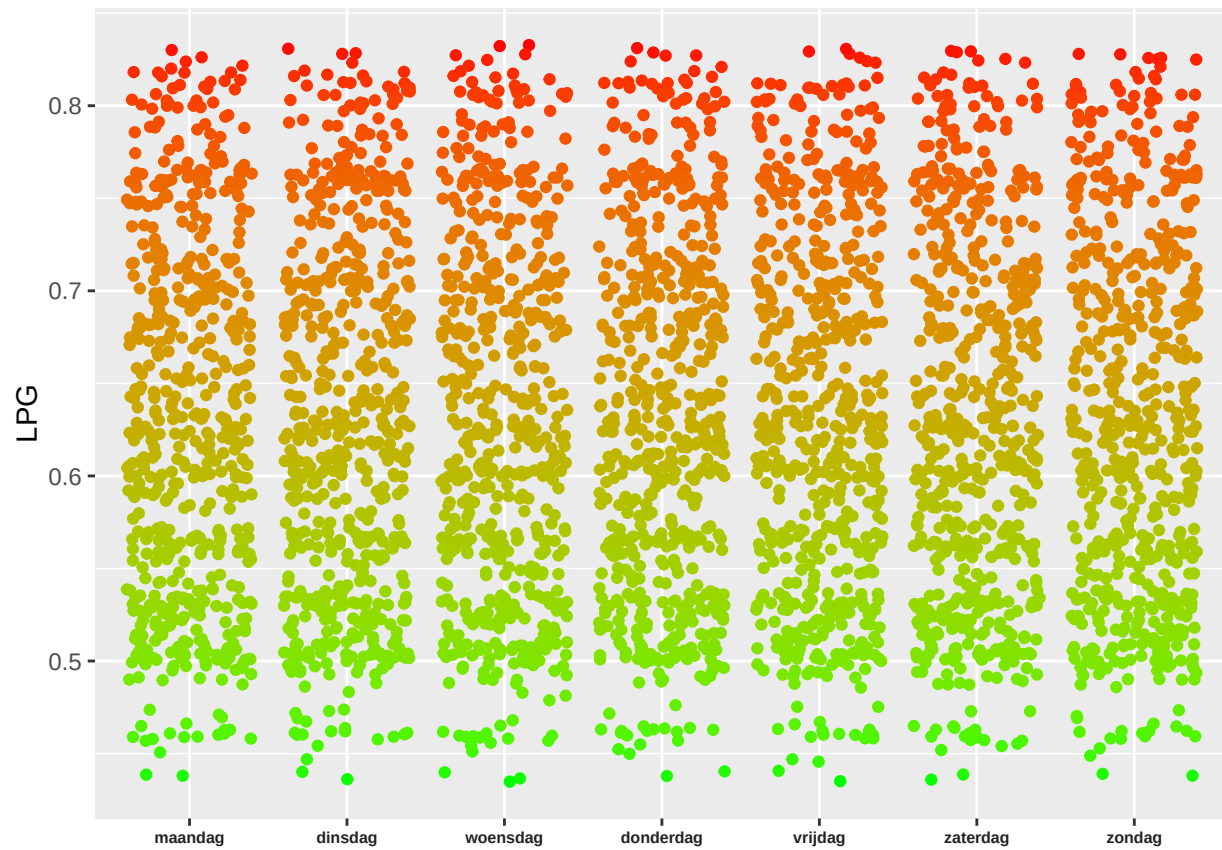
```



```
dieselDagJitter
```



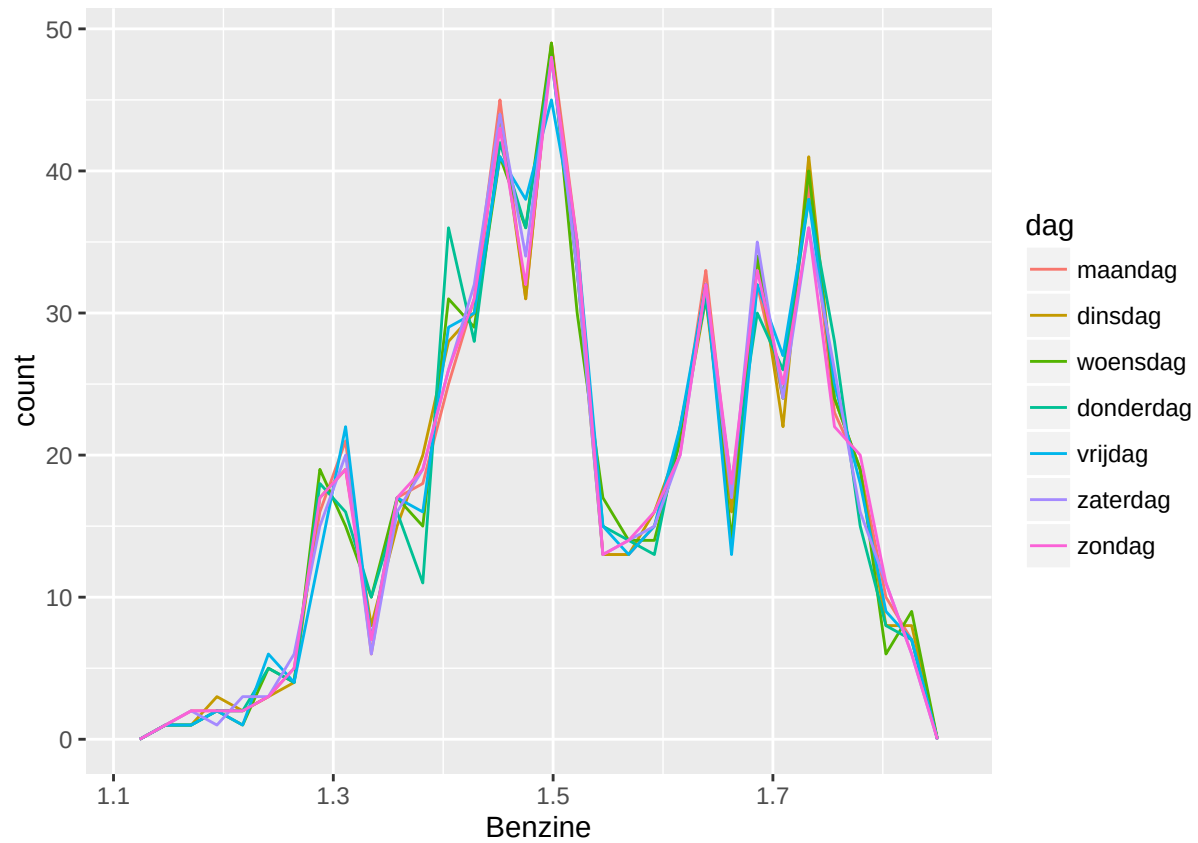
```
LPGDagJitter
```



15.9.4 Prijzen per dag - frequenties

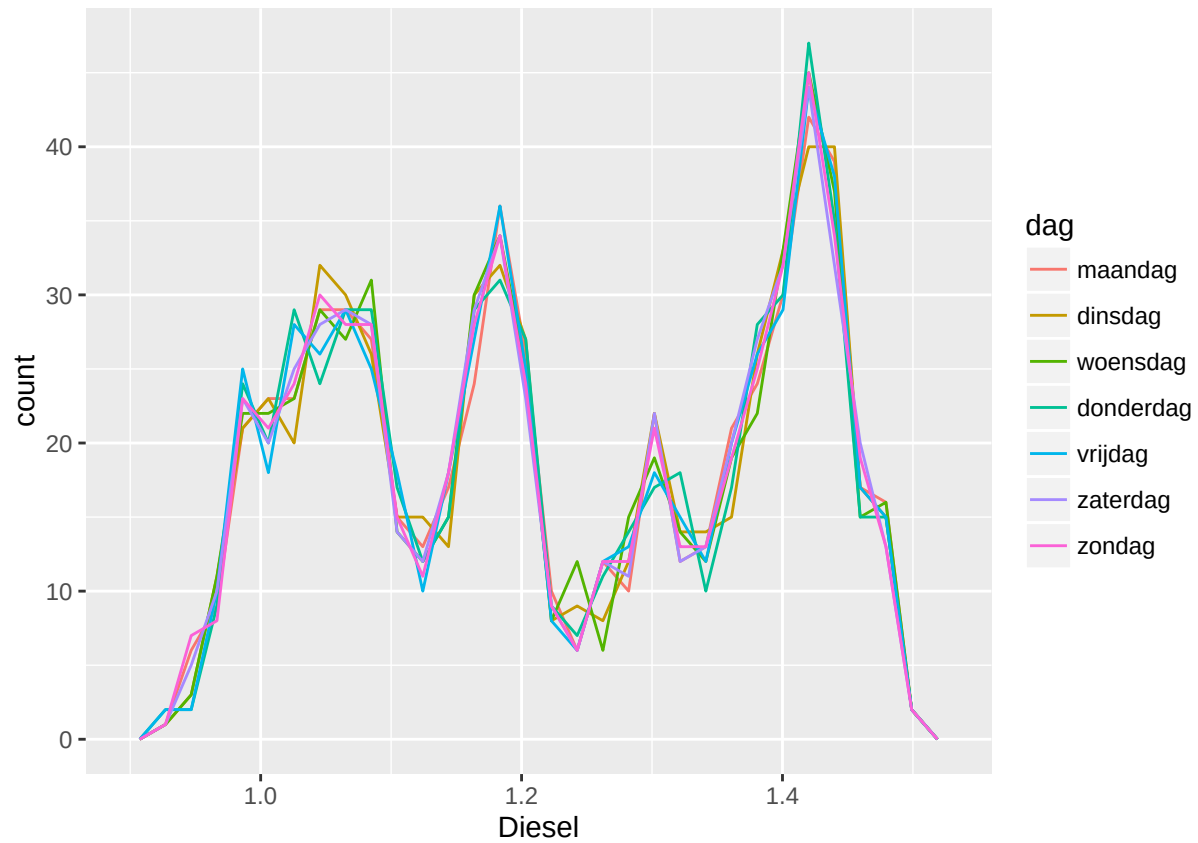
```
ggplot(brandstof, aes(Benzine, colour=dag))+geom_freqpoly(aes(group = dag))
```

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



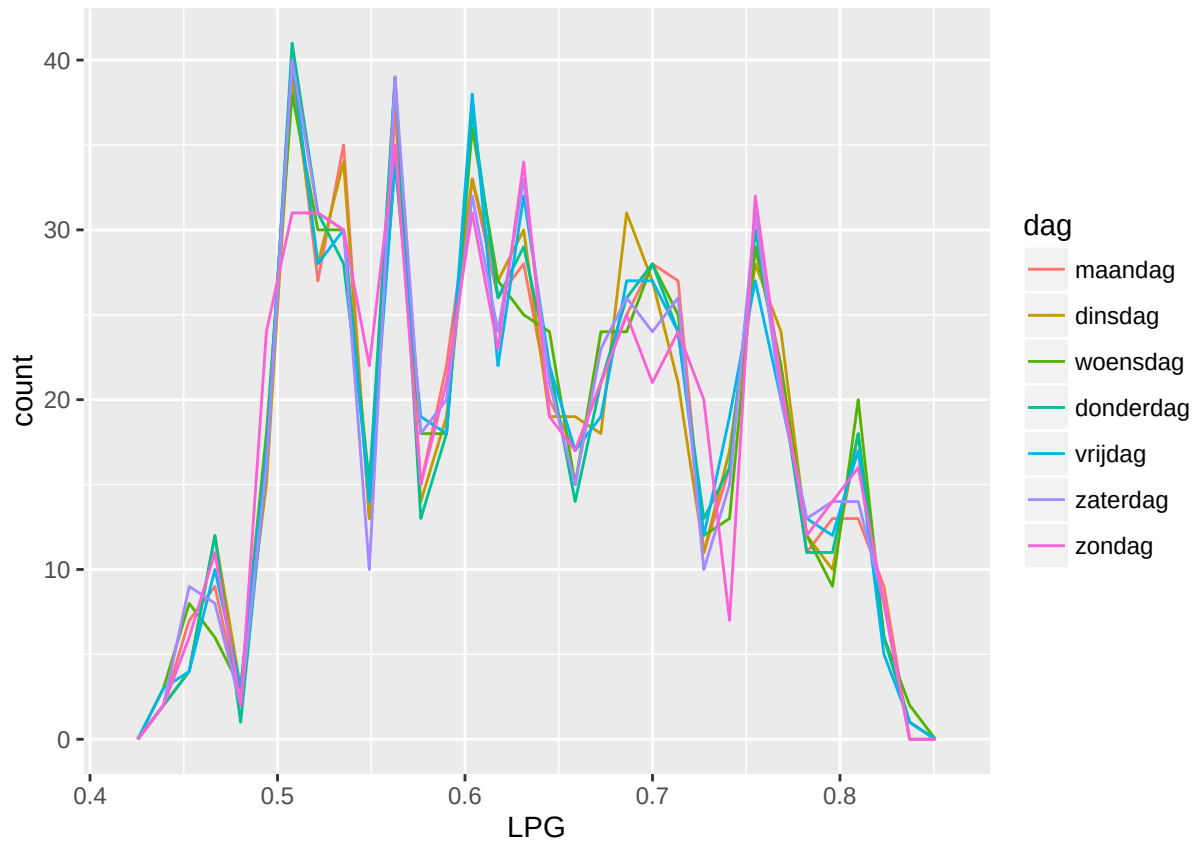
```
ggplot(brandstof, aes(Diesel, colour=dag))+geom_freqpoly(aes(group = dag))
```

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



```
ggplot(brandstof, aes(LPG, colour=dag))+geom_freqpoly(aes(group = dag))
```

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



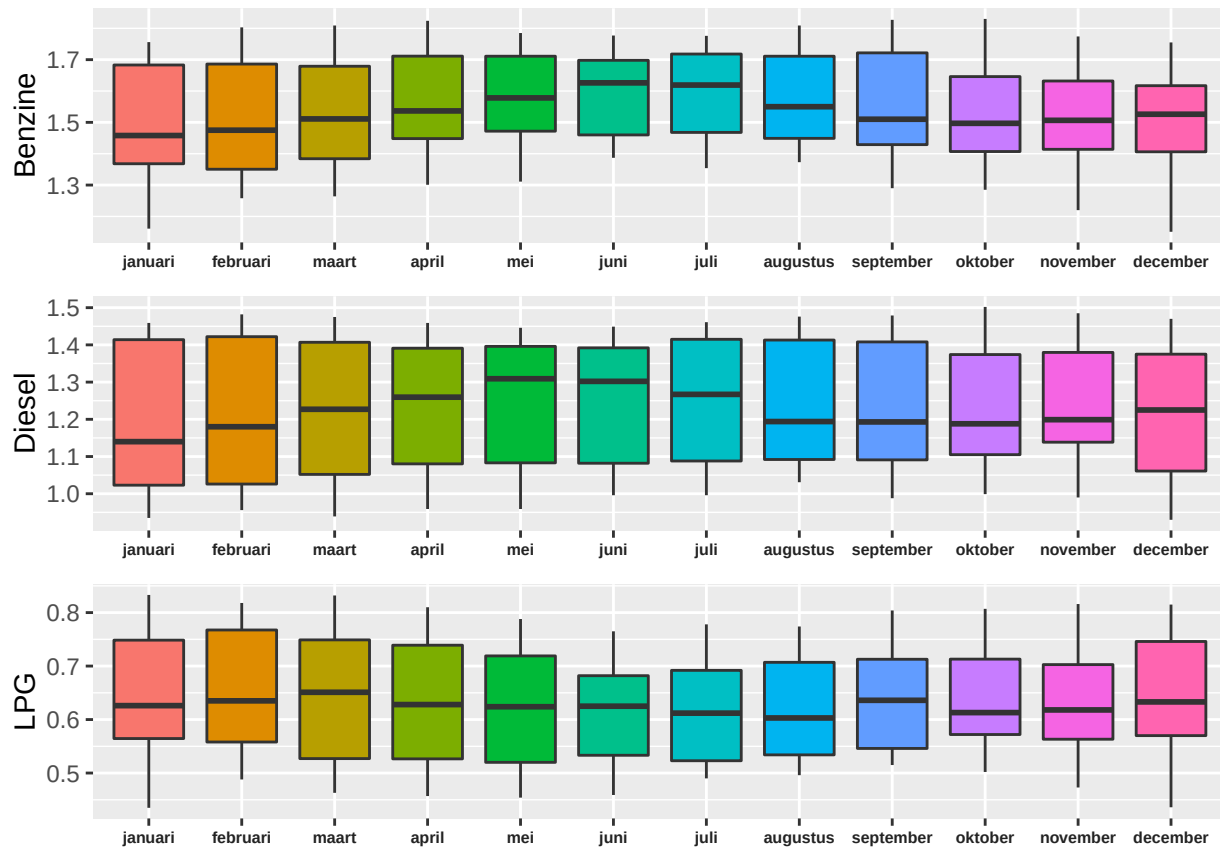
15.9.5 Prijzen per maand

```
benzineMaandBoxplot <- ggplot(brandstof,aes(maanden,Benzine)) +
  geom_boxplot(aes(fill = maanden)) +
  theme(legend.position="none",axis.text.x = element_text(face="bold", color="#222222",size=6),axis.tit.

dieselMaandBoxplot <- ggplot(brandstof,aes(maanden,Diesel)) +
  geom_boxplot(aes(fill = maanden)) +
  theme(legend.position="none",axis.text.x = element_text(face="bold", color="#222222",size=6),axis.tit.

LPGMaandBoxplot <- ggplot(brandstof,aes(maanden,LPG)) +
  geom_boxplot(aes(fill = maanden)) +
  theme(legend.position="none",axis.text.x = element_text(face="bold", color="#222222",size=6),axis.tit.

ggplot2.multiplot(benzineMaandBoxplot,dieselMaandBoxplot,LPGMaandBoxplot,cols=1)
```

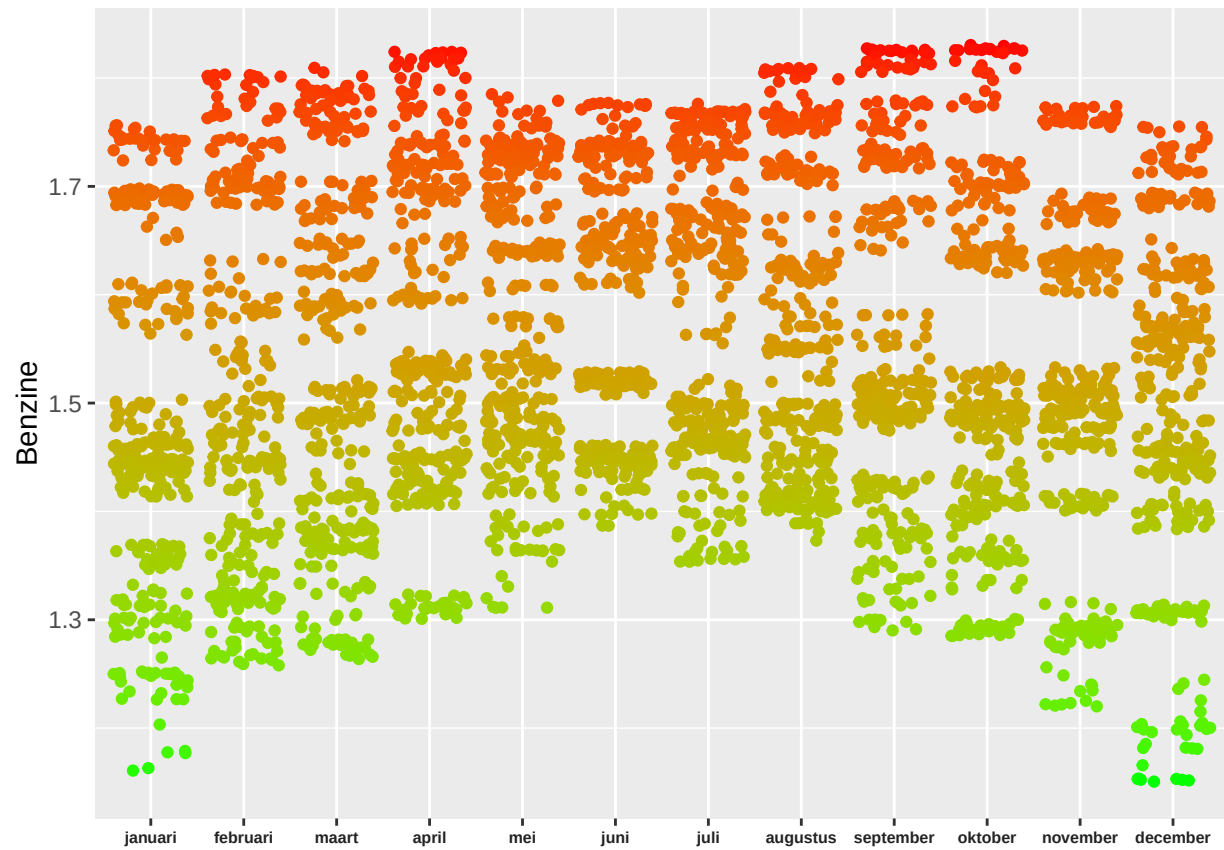
15.9.6 Prijzen per maand

```
benzineMaandJitter <- ggplot(brandstof,aes(maanden,Benzine)) +
  geom_jitter(aes(color = Benzine)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

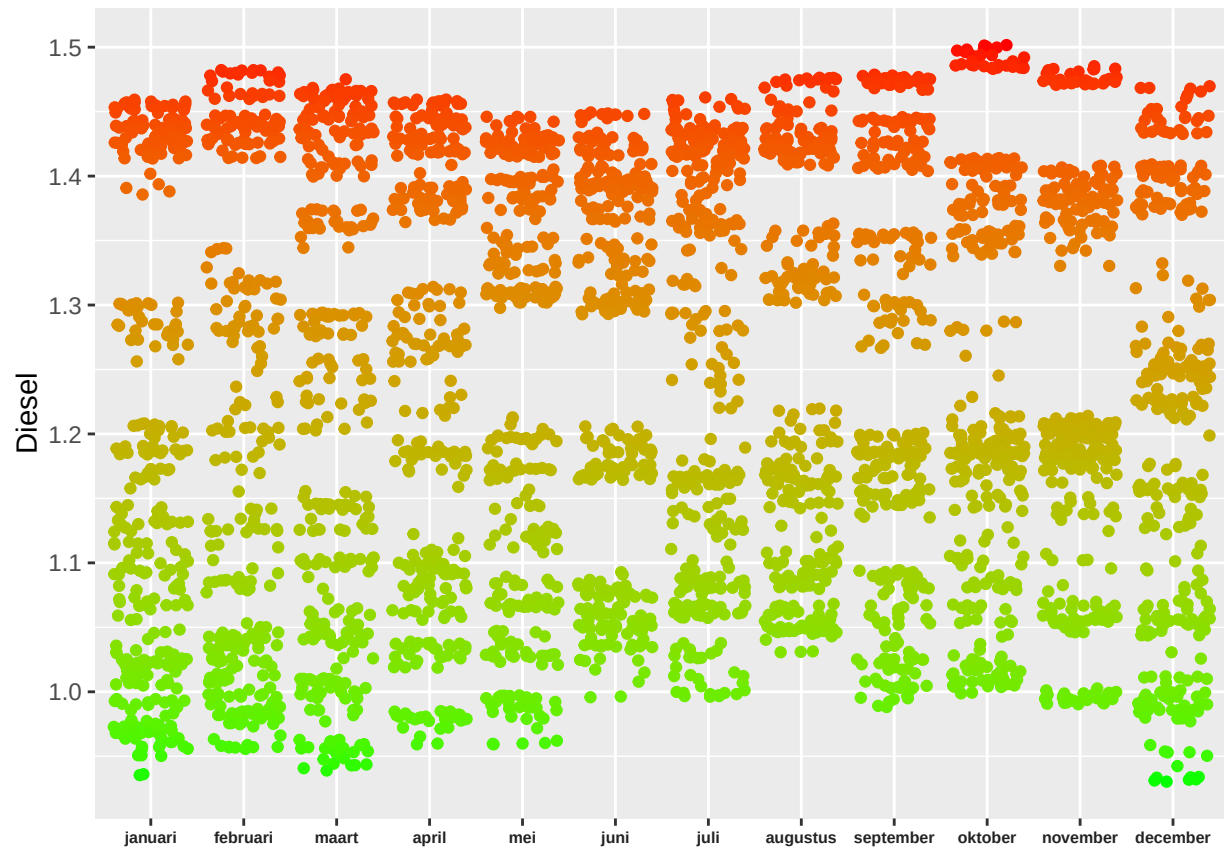
dieselMaandJitter <- ggplot(brandstof,aes(maanden,Diesel)) +
  geom_jitter(aes(color = Diesel)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

LPGMaandJitter <- ggplot(brandstof,aes(maanden,LPG)) +
  geom_jitter(aes(color = LPG)) +
  scale_color_gradient(low='green',high='red') +
  theme(legend.position="none",axis.title.x=element_blank(),
        axis.text.x = element_text(face="bold", color="#222222",size=6))

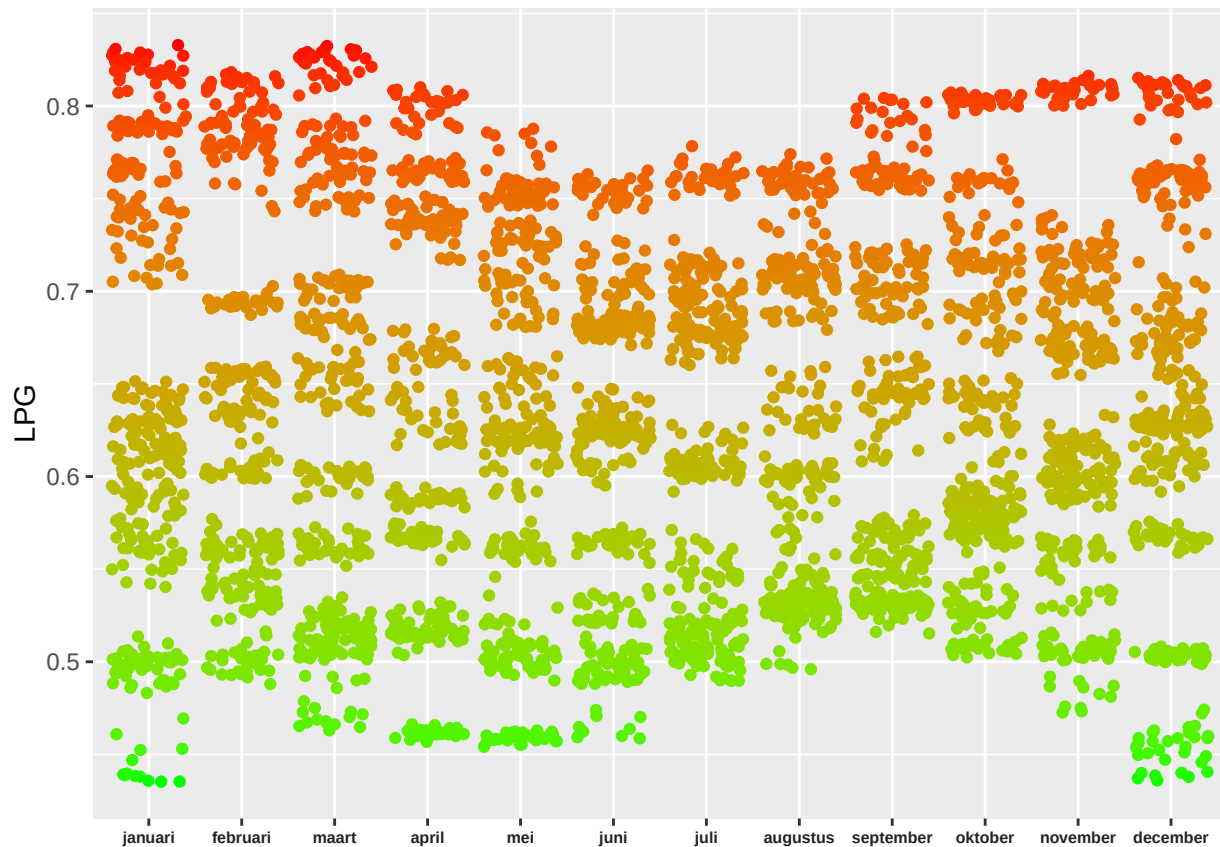
benzineMaandJitter
```



```
dieselMaandJitter
```



```
LPGMaandJitter
```



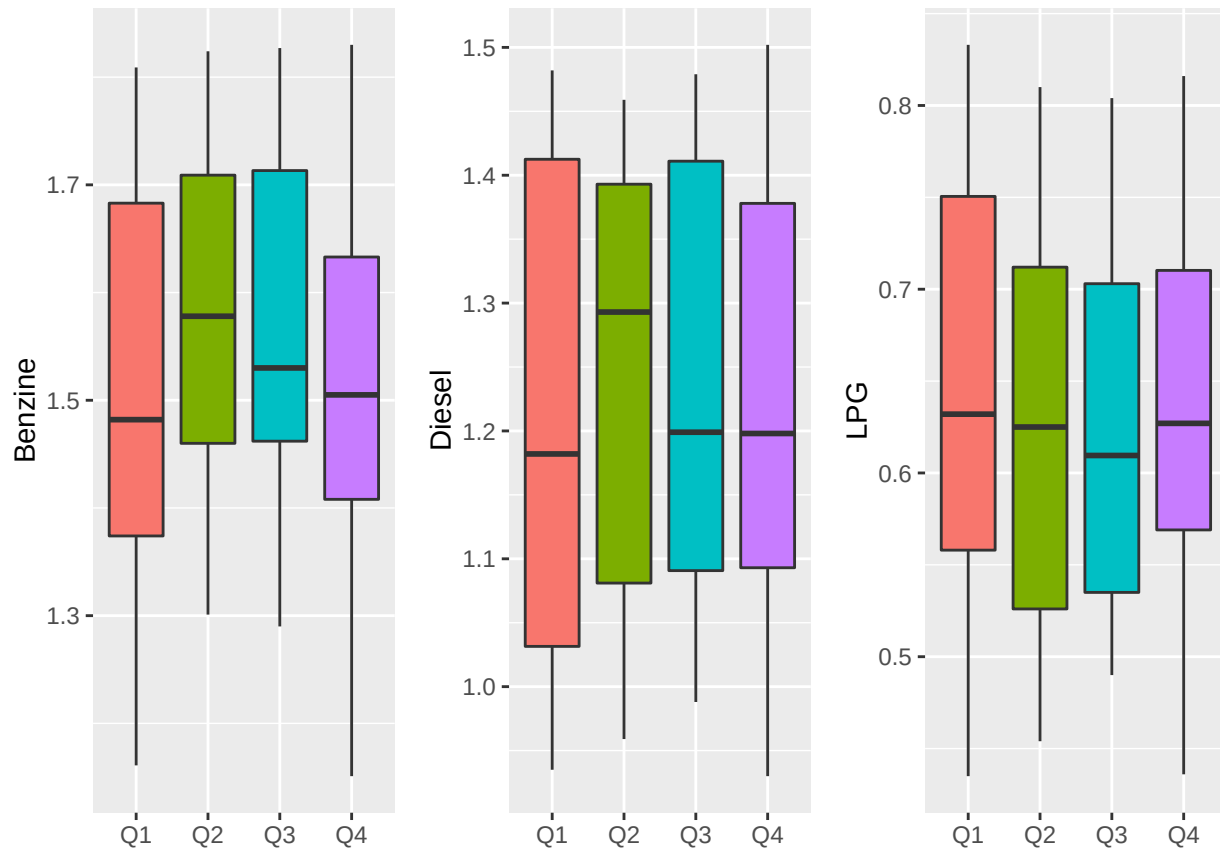
15.9.7 Prijzen per kwartaal

```
benzineKwartaalBoxplot <- ggplot(brandstof,aes(kwartalen,Benzine)) +
  geom_boxplot(aes(fill = kwartalen)) +
  theme(legend.position="none",axis.title.x=element_blank())

dieselKwartaalBoxplot <- ggplot(brandstof,aes(kwartalen,Diesel)) +
  geom_boxplot(aes(fill = kwartalen)) +
  theme(legend.position="none",axis.title.x=element_blank())

LPGKwartaalBoxplot <- ggplot(brandstof,aes(kwartalen,LPG)) +
  geom_boxplot(aes(fill = kwartalen)) +
  theme(legend.position="none",axis.title.x=element_blank())

ggplot2.multiplot(benzineKwartaalBoxplot,dieselKwartaalBoxplot,LPGKwartaalBoxplot,cols=3)
```



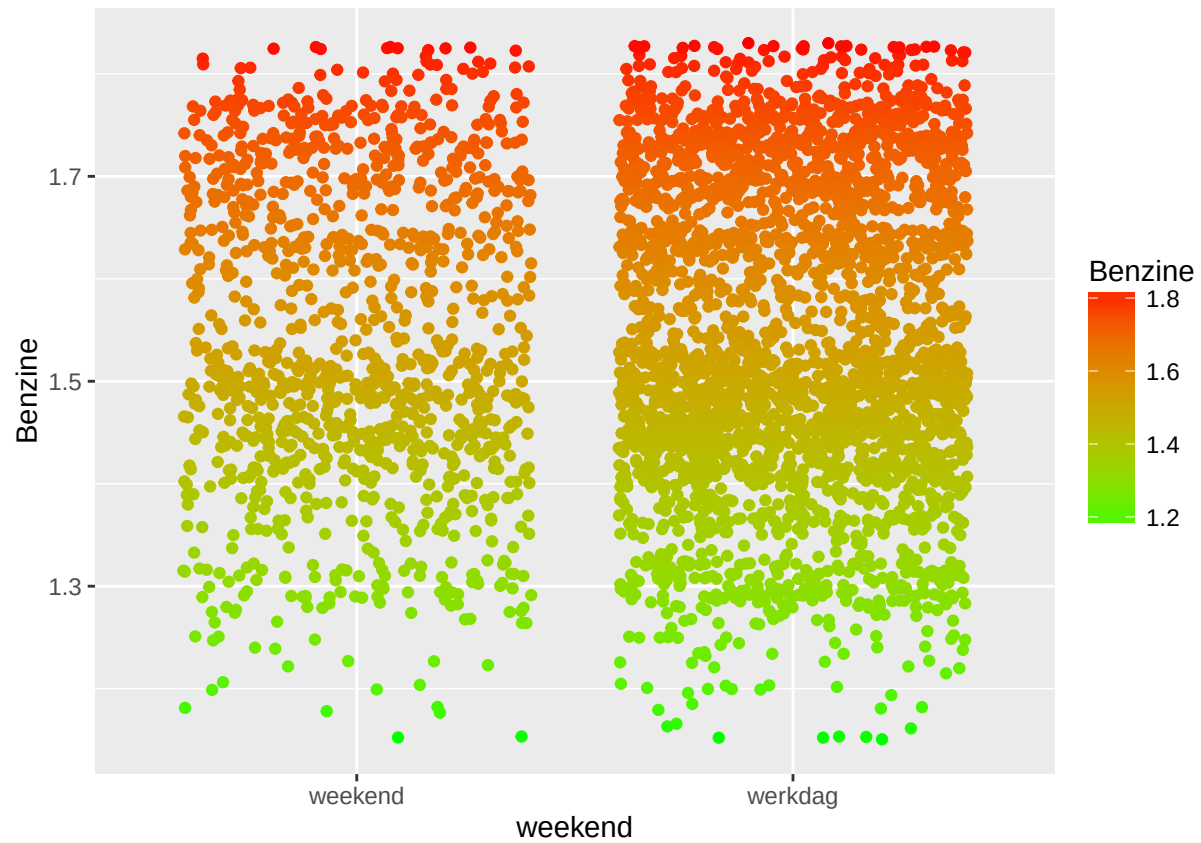
15.9.8 Vergelijking prijzen werkdagen en in het weekend

```
benzineWeekendPlot <- ggplot(brandstof,aes(weekend,Benzine)) +
  geom_jitter(aes(color = Benzine)) +
  scale_color_gradient(low='green',high='red')

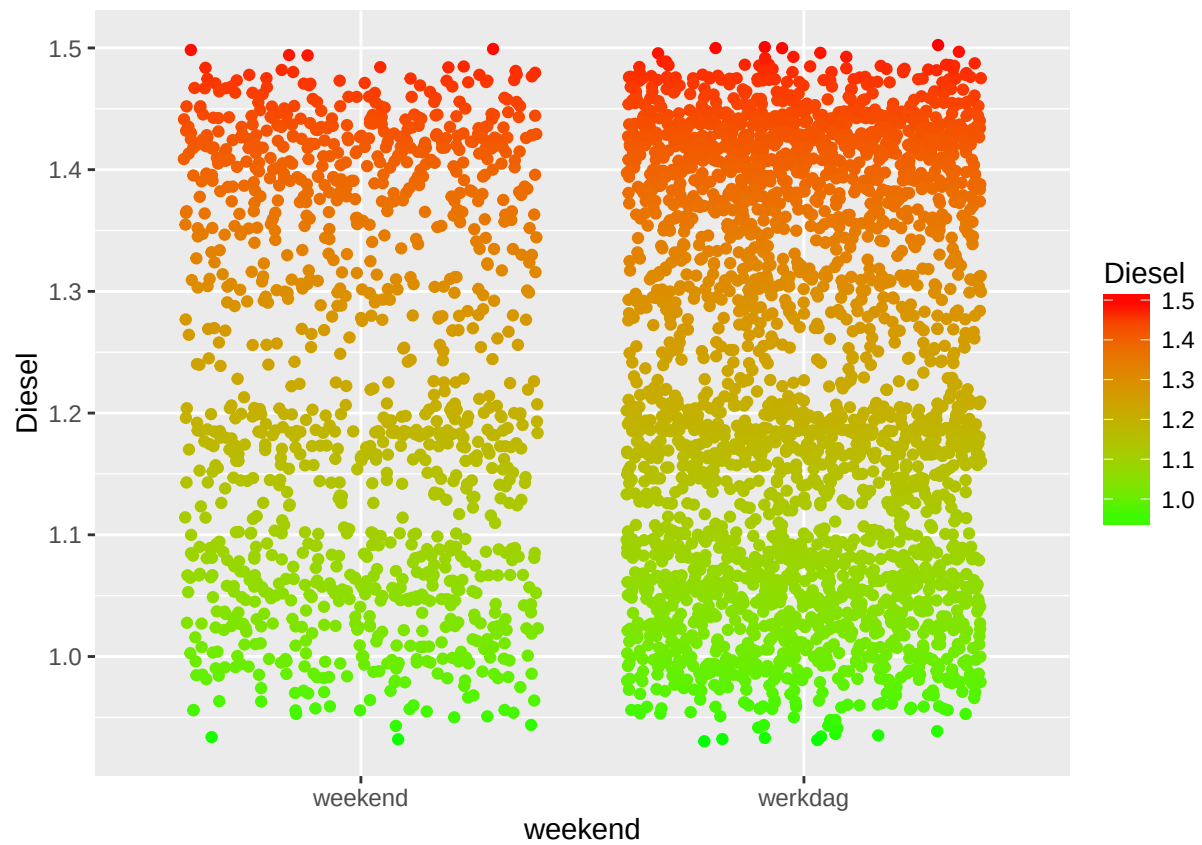
dieselWeekendPlot <- ggplot(brandstof,aes(weekend,Diesel)) +
  geom_jitter(aes(color = Diesel)) +
  scale_color_gradient(low='green',high='red')

LPGWeekendPlot <- ggplot(brandstof,aes(weekend,LPG)) +
  geom_jitter(aes(color = LPG)) +
  scale_color_gradient(low='green',high='red')

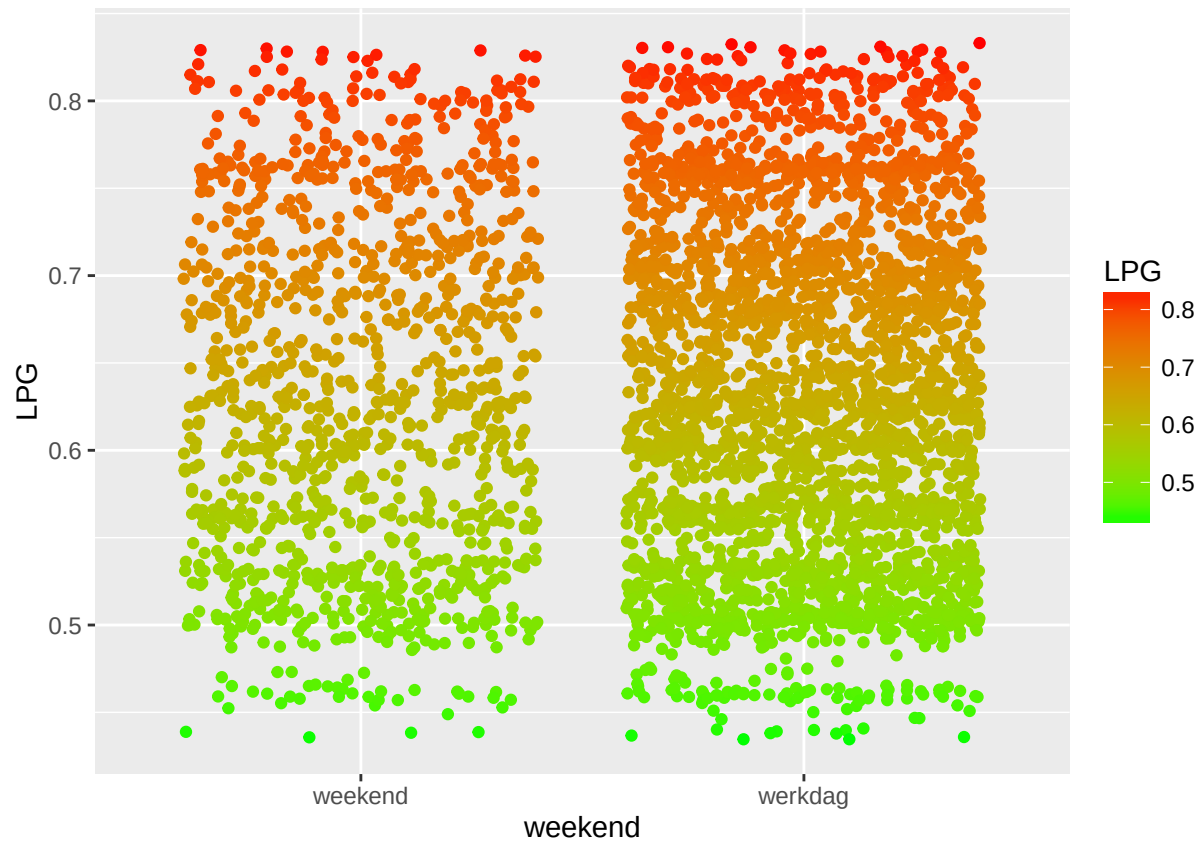
benzineWeekendPlot
```



```
dieselWeekendPlot
```



```
LPGWeekendPlot
```



16 Antwoorden voor oefeningen

16.1 De basics van de R-syntax

16.1.0.1 Maak vectoren met jouw persoonlijke informatie zoals *naam*, *leeftijd*, *woonplaats* en *opleiding*

```
naam <- "Peter"

leeftijd <- 32

woonplaats <- "Leeuwarden"

opleiding <- "WO - Bedrijfskunde"
```

16.1.0.2 Voeg de vectoren samen in een vector die deze informatie bevat

```
persoon <- c(naam, leeftijd, woonplaats, opleiding)

persoon
```

16.1.0.3 Selecteer de woonplaats uit de zojuist aangemaakte vector

```
persoon[3]
```

```
## [1] "Amsterdam"
```

16.2 Reken- en statistische functies in R

16.2.0.1 Bereken de oppervlakte van een cirkel die een doorsnede heeft van 11 meter

```
(5.5 ^ 2) * pi
```

```
## [1] 95.03318
```

16.2.0.2 Gebruik de stelling van Pythagoras om de C van een driehoek te berekenen waarvan A 11cm is en B 4cm

```
A <- 11
B <- 4

C <- A ^ 2 + B ^ 2

C
```

```
## [1] 137
```

```
sqrt(C)
```

```
## [1] 11.7047
```

16.2.0.3 Reken uit hoeveel geld je op je spaarrekening hebt na 25 jaar als je 1000 vastzet voor 2%, MAAR de spaarrekening 2,70 euro per jaar kost

```
(1000 * 1.02 ^ 25) - (25 * 2.7)
```

```
## [1] 1573.106
```

16.3 Data analyseren en bewerken in R

16.3.0.1 Geef een statistische samenvatting van de mpg variable uit de mtcars dataset

```
summary(mtcars$mpg)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.40   15.42   19.20   20.09   22.80   33.90
```

```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

```
sd(mtcars$mpg)
```

```
## [1] 6.026948
```

16.3.0.2 Maak een lineair model die de zuinigheid (mpg) voorspelt op basis van het gewicht (wt) van de auto

```
attach(mtcars)
```

```
## The following object is masked from package:ggplot2:
```

```
##
```

```
##      mpg
```

```
## The following objects are masked from mtcars (pos = 9):
```

```
##
```

```
##      am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt
```

```
autoModel <- lm(mpg ~ wt, mtcars)
```

```
autoModel
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Coefficients:
## (Intercept)          wt
##      37.285      -5.344
```

```
summary(autoModel)
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776   19.858 < 2e-16 ***
## wt          -5.3445     0.5591   -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

16.3.0.3 Welke variabelen uit de mtcars dataset hebben de hoogste correlatie?

```
str(mtcars)
```

```
## 'data.frame':  32 obs. of  11 variables:
## $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num  160 160 108 258 360 ...
## $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num  16.5 17 18.6 19.4 17 ...
## $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
## $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
cor(mtcars)
```

```
##           mpg           cyl           disp           hp           drat           wt
## mpg      1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594
## cyl     -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958
```

```
## disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799
## hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479
## drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406
## wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000
## qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159
## vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157
## am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953
## gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870
## carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059
##      qsec      vs      am      gear      carb
## mpg  0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
## cyl -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
## disp -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
## hp   -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
## drat  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
## wt   -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
## qsec  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
## vs    0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
## am   -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
## gear -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
## carb -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

16.4 Data importeren en exporteren in R

16.4.0.1 Exporteer de ingebouwde `mtcars` dataset zodat deze perfect kan worden geopend in Excel (mag `.csv` of `.xls` zijn)

```
autos <- mtcars
autos$merk <- row.names(autos)
write.table(autos, "autos.csv", sep = ";", row.names = FALSE)
```

16.4.0.2 Importeer een externe dataset. Dit kan bijvoorbeeld een tabel zijn op de website van de Centraal Bureau voor de statistiek

```
# Dataset opslaan en importeren met "Import Dataset" in RStudio
# Maak gebruik van de "url()" functie in combinatie met read.csv()
```

16.4.0.3 Exporteer deze externe dataset als `.csv` bestand

```
# maak gebruik van de "write.csv()" functie
```

16.5 Data frame bewerkingen

16.5.0.1 Maak een selectie van `autos`'s in de `mtcars` data frame die meer dan 110 pk (hp) hebben

```
mtcars[mtcars$hp >110 , ]
```

```
##          mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0   3   2
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0   3   4
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0   4   4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0   4   4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0   3   3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0   3   3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0   3   3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0   3   4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0   3   4
## Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0   3   4
## Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0   3   2
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0   3   2
## Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0   3   4
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0   3   2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1   5   2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1   5   4
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1   5   6
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1   5   8
```

16.5.0.2 Voeg een nieuwe kolom toe die aangeeft met “Ja” of “Nee” of de auto wel of niet meer dan 110 hp heeft

```
autos <- mtcars
autos$plus110pk <- ifelse(autos$hp > 110, "Ja", "Nee")

# bekijk het resultaat
# View(autos)
```

16.5.0.3 Voeg zelf een auto toe aan de data frame met dezelfde eigenschappen als in de mtcars dataset (Tip, maak gebruik van de rbind() functie)

```
autos <- mtcars

# bekijk de namen van de dataset
names(autos)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

```
# maak een data frame die uit een enkele rij bestaat
# vul deze rij met data in precies dezelfde volgorde als in de
# autos/mtcars dataset
```

```
bmw <- data.frame(row.names = "BMW 10-er", mpg = 30, cyl = 6, disp = 250, hp = 300, drat = 3.78, wt = 2.9, qsec = 11.2, vs = 1, am = 0, gear = 5, carb = 6)

bmw
```

```
##          mpg cyl disp  hp drat   wt  qsec vs am gear carb plus110pk
## BMW 10-er  30   6  250 300 3.78 2.9 11.2  1  0   5   6         Ja
```

16.6 Werken met lijsten en matrixen in R

16.6.0.1 Maak een matrix die bestaat uit 3 kolommen en 4 rijen en uit willekeurige nummers bevat.

```
getallen <- c(1:12)
matrix <- matrix(getallen, nrow = 4, ncol = 3)
```

16.6.0.2 Maak lists over de gegevens van jouw familie, en stel deze samen toe een matrix

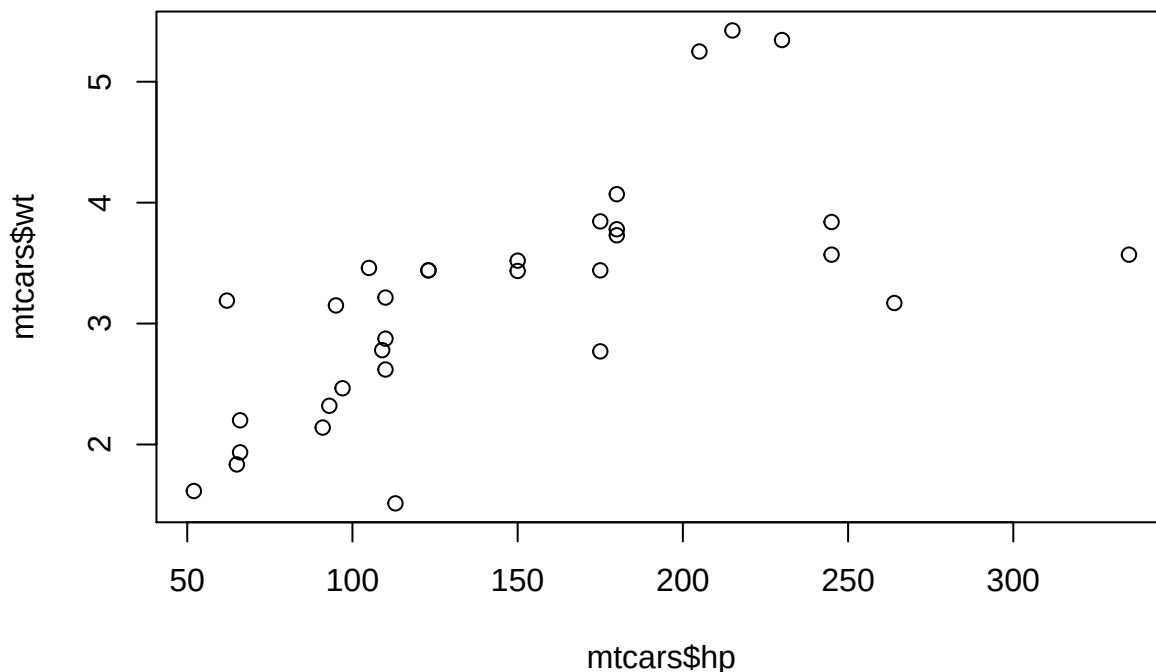
```
# zoals in voorbeeld

Japie <- list("Japie","Krekel", 1.47, 65.3, "Utrecht", TRUE,"Rechten","M",23)
Roos <- list("Roos","Jansen", 1.67, 60, "Utrecht", FALSE,"Geneeskunde","V",25)
Hans <- list("Hans","Klock", 1.78, 78.2, "Utrecht", FALSE,"Bedrijfskunde","M",20)
Marco <- list("Marco","Polo", 1.89, 81.3, "Leiden", FALSE,"Rechten","M",19)
Erica <- list("Erica","Jansen", 1.53, 65.3, "Leiden", TRUE,"Geneeskunde","V",27)

personenMatrix <- matrix(c(Harry, Japie, Roos, Hans, Marco, Erica),nrow = 6, ncol = 9, byrow = TRUE, dimnames = list(c("Naam", "Voorname", "Leeftijd", "Gewicht", "Plaats", "Rechts", "Opleiding", "Geslacht", "Jaar"), c("Harry", "Japie", "Roos", "Hans", "Marco", "Erica")))
```

16.6.0.3 Gebruik de mtcars dataset om de verhouding tussen paardenkracht (hp) en gewicht (wt) te visualiseren in een grafiek (je mag er zelf voor kiezen of je gebruikt maakt van de attach() functie of niet)

```
plot(mtcars$hp, mtcars$wt)
```



16.7 Handig gebruik maken van het “factor” datatype

16.7.0.1 Maak een gesorteerde *factor* van de maanden in het jaar en sorteer deze op de juiste manier

```
maanden <- c("januari", "februari", "maart", "april",
             "mei", "juni", "juli", "augustus", "september",
             "oktober", "november", "december")

maandenFactor <- factor(maanden,
                        levels = c("januari", "februari", "maart", "april",
                                   "mei", "juni", "juli", "augustus", "september",
                                   "oktober", "november", "december"))
```

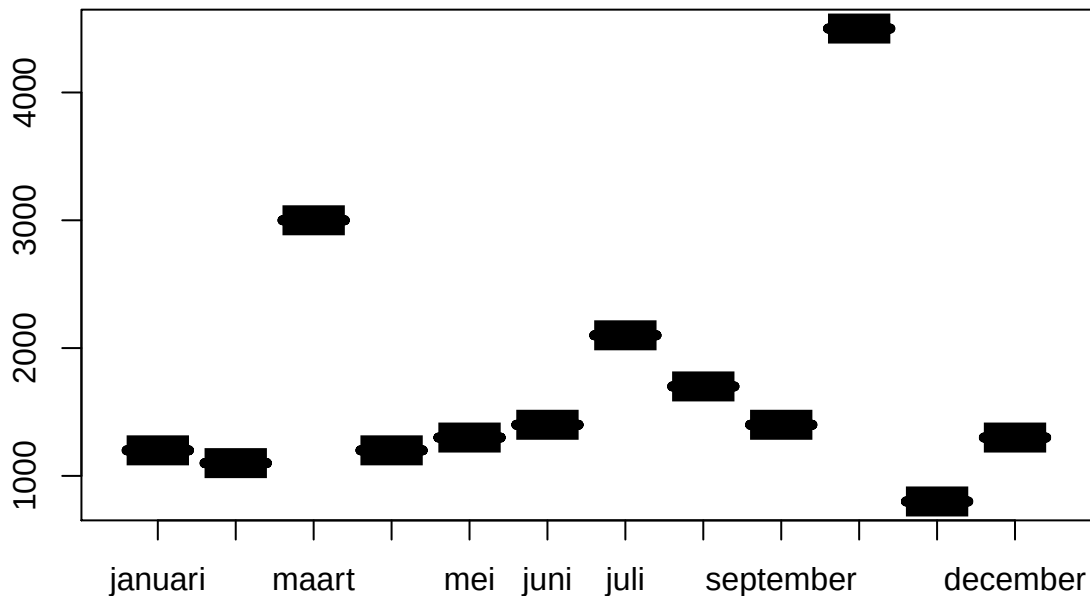
16.7.0.2 Maak een dataframe die uit twee kolommen bevat, **omzet**: bevat numerieke waarden over een denkbeeldige omzet per maand, en **maand**: een *factor* vector die de maanden van het jaar bevat (die in de vorige oefening is gemaakt).

```
omzet <- c(1200, 1100, 3000, 1200, 1300, 1400, 2100, 1700, 1400, 4500, 800, 1300)

omzetTabel <- data.frame(omzet, maandenFactor)
```

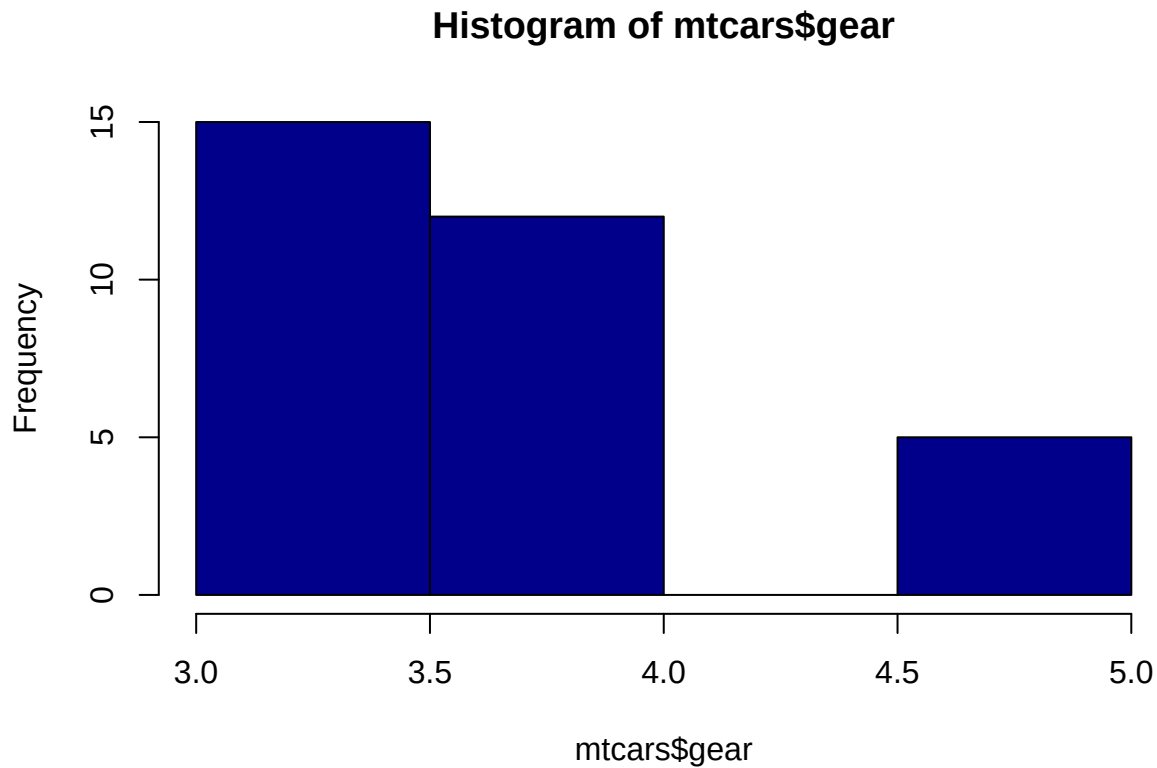
16.7.0.3 Sorteer de data frame op de maanden oplopend. Maak daarbij ook een lijndiagram die de omzet per maand weergeeft

```
plot(omzetTabel$maandenFactor, omzetTabel$omzet, type = "l", lwd = 5)
```



16.7.0.4 Maak een donkerblauwe histogram die horizontaal is weergegeven en informatie bevat over het aantal auto's die per aantal versnellingen

```
hist(mtcars$gear, col = "darkblue")
```



16.7.0.5 Maak dezelfde visualisatie als in de eerste opgave. Echter deze keer helemaal uitgewerkt met de volgende onderdelen: Een titel op de x-as, een titel op de y-as, een titel van de grafiek, een leuke vorm en een mooie kleur. Gebruik je creativiteit.

16.8 Correlaties, functies en loops in R

16.8.0.1 Schrijf een functie die automatisch een percentage zoals “1.25 %” (character) omzet naar een fractie (numeric) zoals 0.0125

```
Percentage <- "1,25%"

percentageToFraction <- function(Percentage){
  x <- Percentage
  x <- gsub("%", "", x)
  x <- gsub(",", ".", x)
  x <- as.numeric(x)
  x <- x / 100
  x
}

# testen

percentageToFraction("56,78%")
```

```
## [1] 0.5678
```


16.8.0.2 Maak een loop of functie die de prijzen van een product omzet in dollars door gebruik te maken van de wisselkoers en er een dollarteken voor het bedrag plaatst

```
Prijs <- 2.34

priceToDollar <- function(Prijs){
  x <- Prijs
  x <- round(x * 1.12, 2)
  x <- as.character(x)
  x <- paste("$", x)
  x
}

# de functie testen

priceToDollar(600)
```

```
## [1] "$ 672"
```

16.9 Lineaire regressiemodel

16.9.0.1 Maak een voorspellingsmodel die Winst voorspelt op basis van UrenProject

```
attach(Projecten2)
```

```
## The following objects are masked from Projecten2 (pos = 8):
##
##   AfstandKlant, Maand, MateriaalKosten, Nummer,
##   OpbrengstProject, PersoneelsKosten, TevredenheidKlant,
##   Type.Project, UrenProject, Werkgroep, Winst
```

```
## The following object is masked from CDNOW.df:
##
##   Maand
```

```
model1 <- lm(Winst ~ UrenProject, Projecten2)
```

16.9.0.2 Onderzoek in welke maand over het algemeen de minste Winst wordt gemaakt (los van het feit hoe betrouwbaar het model is)

```
model2 <- lm(Winst ~ Maand, Projecten2)

summary(model2)
```

```
##
## Call:
## lm(formula = Winst ~ Maand, data = Projecten2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -5647.1 -1361.6    14.9  1703.1  4881.9
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4227.44     804.31   5.256 1.02e-06 ***
## Maandaugustus -1099.32     1172.48  -0.938  0.3510
## Maanddecember  -956.32     1172.48  -0.816  0.4169
## Maandfebruari   -44.56     1137.47  -0.039  0.9688
## Maandjanuari  -1239.56     1137.47  -1.090  0.2788
## Maandjuli      -857.57     1172.48  -0.731  0.4665
## Maandjuni      -1167.94     1172.48  -0.996  0.3219
## Maandmaart     -1617.89     1137.47  -1.422  0.1585
## Maandmei        19.18     1172.48   0.016  0.9870
## Maandnovember  -876.57     1172.48  -0.748  0.4567
## Maandoktober   -243.19     1172.48  -0.207  0.8362
## Maandseptember -2115.07     1172.48  -1.804  0.0747 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2413 on 88 degrees of freedom
## Multiple R-squared:  0.07611,    Adjusted R-squared:  -0.03938
## F-statistic: 0.659 on 11 and 88 DF,  p-value: 0.773

# september heeft de grootste negatieve coëfficiënt
```

16.10 Uitgebreide visualisaties maken in R met ggplot

16.10.0.1 Creer een scatterplot (zoals in de demonstraties) waarmee een goed inzicht in de volgende dimensies kan worden gegeven:

*Het gewicht van de auto; De paardenkracht van de auto; De zuinigheid van de auto; De acceleratie van de auto; *Het aantal cylinders van de auto.*

*Aan de hand van de visualisatie kun je een auto vinden die zeer snel optrekt (hoge *qsec*) maar slechts weinig paardenkracht (*hp*) heeft. Natuurlijk wil je weten welke auto dit is, dat gaan we uitzoeken in de volgende oefening.*

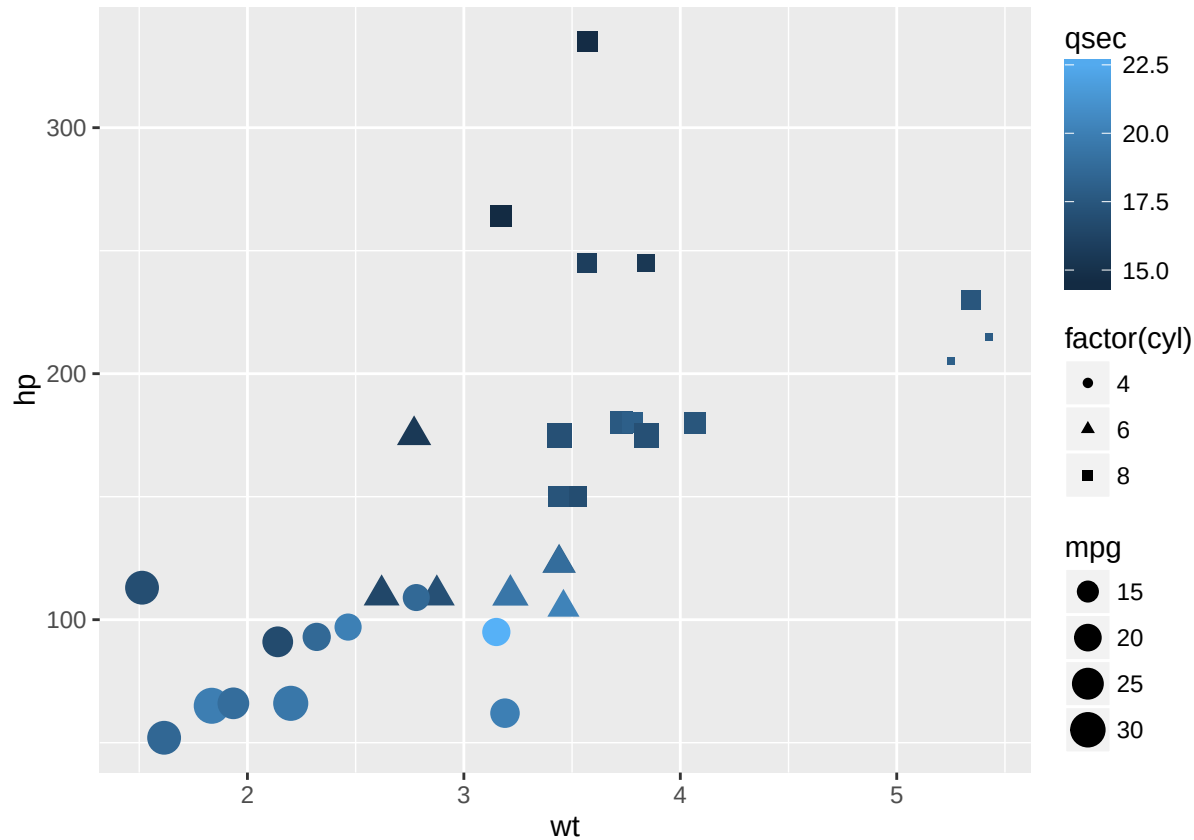
```
require(ggplot2)
attach(mtcars)

## The following objects are masked from mtcars (pos = 4):
##
##      am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

## The following object is masked from package:ggplot2:
##
##      mpg

## The following objects are masked from mtcars (pos = 11):
##
##      am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt
```

```
autos <- ggplot(mtcars,aes(wt, hp))
autos + geom_point(aes(size = mpg, color = qsec, shape = factor(cyl)))
```



```
detach(mtcars)
```

16.10.0.2 Visualiseer de namen van de auto's in de zojuist gemaakte grafiek. Hiermee kun je achterhalen welke eigenschappen bij welke auto's horen.

Maak gebruik van het [14] Textual annotations- hoofdstuk uit de documentatie van ggplot om er achter te komen hoe je gebruik kan maken van labels.

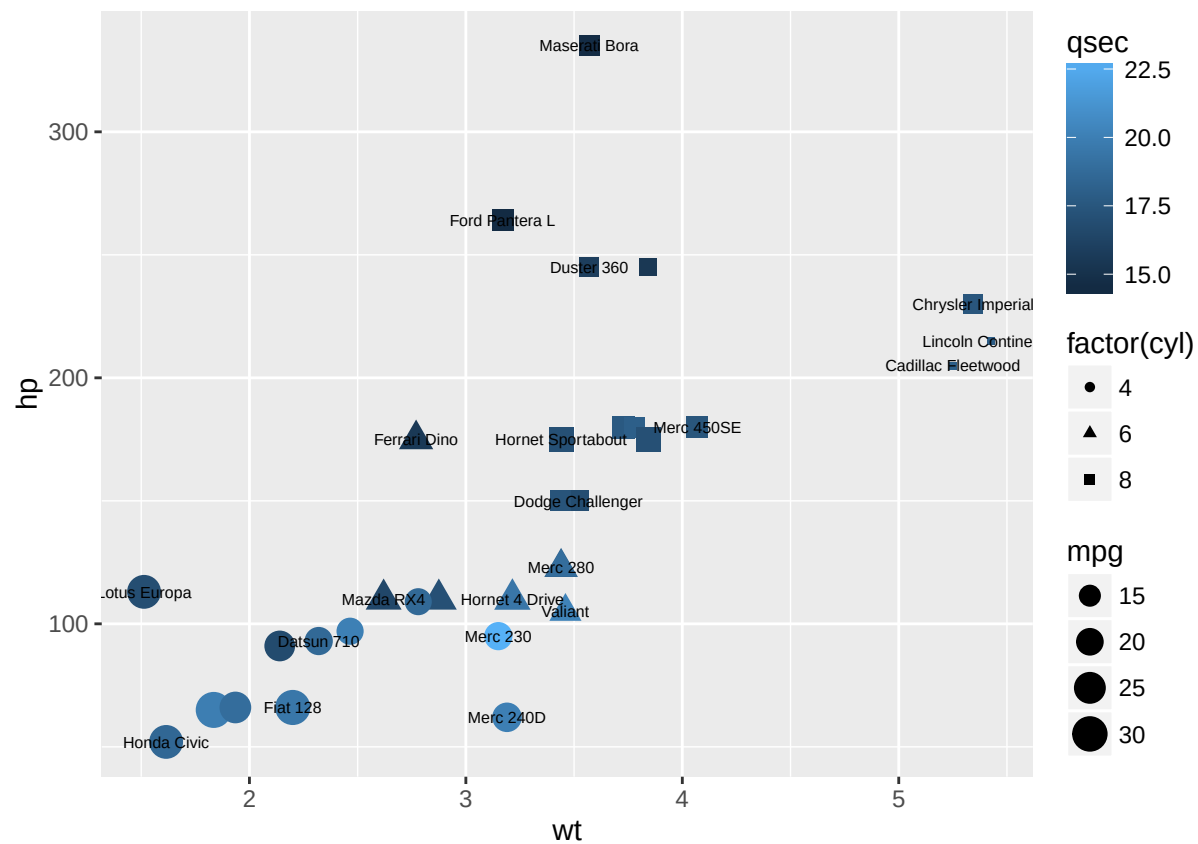
```
require(ggplot2)
attach(mtcars)
```

```
## The following objects are masked from mtcars (pos = 4):
##
##   am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

## The following object is masked from package:ggplot2:
##
##   mpg

## The following objects are masked from mtcars (pos = 11):
##
##   am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt
```

```
autos <- ggplot(mtcars, aes(wt, hp, label = rownames(mtcars)))
autos + geom_point(aes(size = mpg, color = qsec, shape = factor(cyl))) + geom_text(check_overlap = TRUE)
```



```
detach(mtcars)
```

17 Jouw data avontuur is nog maar net begonnen...

17.1 Experimenteer!

Ik hoop dat je een leuke ervaring hebt gehad met het leren van R aan de hand van dit boek. De voorbeelden in dit boek zijn slechts een heel klein deel van alle mogelijkheden in R. Het is daarom leuk om verder te experimenteren met R door zelf een dataset uit te zoeken, bijvoorbeeld uit op het dataportaal van de overheid, en daarmee jezelf steeds uit te dagen om waardevolle informatie te halen uit de dataset. Ik kan je nu al verklappen dat dit gepaard gaat met veel Google-werk en dat je veel op de website **StackOverflow** zal komen voor antwoorden op jouw vragen. Dit is het leerproces, alles wat je niet weet en opzoekt is een onderdeel dat je er weer bij hebt geleerd. Je zal door kennis van de structuur in R nu ook in staat zijn de goede vragen te stellen. Je weet namelijk nu beter wat je nodig hebt en kan daardoor specifiek zijn bij de zoekopdrachten in Google. Op deze manier zal jouw kennis in R groeien. Daarbij zal ik in mijn volgende boek onderwerpen behandelen die meer uitdagend zijn en geschikt zijn voor personen met voldoende kennis van R, waar jij nu dus ook bij hoort.

17.2 Andere platformen om R te leren

Op dit moment zijn er fantastische platformen om R te leren. [**DataCamp**] heeft namelijk fantastische interactieve R-tutorials. Hierbij moet je opdrachten uitvoeren door de juiste commando's in R te toetsen, dit is een goede manier om R goed in de vingers te krijgen. De tutorials zijn per onderwerp verdeeld en variëren van de basics tot uitgebreide statistische analyse in R. Naast **DataCamp** heb je ook online cursussen op **Coursera**. Coursera benadert het iets meer vanuit de theoretische hoek. Je leert hierbij vooral de toepassing van R: *“hoe moet ik bepaalde functies van R gebruiken om een bepaald probleem op te lossen”*. Hiermee wordt vooral het denkvermogen om data gerelateerde problemen in R op te lossen vergroot. **DataCamp** en **Coursera** zijn slechts twee voorbeelden van de beschikbare platformen waarmee je jouw kennis in R kunt vergroten. Er zijn talloze boeken, vooral nog in het Engels, geschreven over R. Daarbij zijn er gigantisch veel artikelen en blogs waarbij experimenten in R worden uitgevoerd. Men is altijd heel vriendelijk om de code te delen waardoor het fantastische hulpmiddelen zijn om jouw R-kennis te vergroten.

17.3 Laat je inspireren en join the club

Laat je ook vooral inspireren over de data gerelateerde cases zoals in dit boek of die je in artikelen tegenkomt. Alles is mogelijk in R zolang je maar de juiste bewerkingen toepast en de juist basis-skills in huis hebt. Daarbij is er een wildgroei aan beschikbare data, hierdoor groeien de mogelijkheden aan onderwerpen om data gerelateerde experimenten uit te voeren. Daarnaast vinden mensen het leuk om een mooi experiment of visualisatie met waardevolle inzichten te lezen. Je kunt er dus zelf ook leuke artikelen over schrijven. Je hebt nu de skillset om analyses met data te doen, R! Zie het einde van dit boek dus als een begin voor jouw avontuur in de wereld van data-analyse.

18 We houden contact

18.0.1 Mail

Ik kom graag met je in contact voor vragen, voorstellen of andere redenen. De beste manier is om mij te mailen op `twigt.arie@gmail.com`, ik zal mijn best doen om zo snel mogelijk te reageren.

18.0.2 Twitter

Je kunt natuurlijk altijd met mij in contact komen via Twitter op mijn account **@ArieTwigt**. Hier kan ik bijvoorbeeld nog sneller je vragen beantwoorden dan via de mail. Ook tweet ik artikelen op het gebied van data analyse en over R, volg mijn account als je het leuk lijkt om deze te lezen.

18.0.3 LinkedIn, website en blog

Op LinkedIn (<https://nl.linkedin.com/in/arietwigt>) , mijn website (arietwigt.com) en blog [*\(arietwigt.wordpress.com\)](http://arietwigt.wordpress.com)⁸ post ik mijn experimenten. Daarbij kom ik graag met je in contact op LinkedIn.

19 Referenties en links

[1] Use = or <- for assignment?

Revolution Analytics, 2008

<http://blog.revolutionanalytics.com/2008/12/use-equals-or-arrow-for-assignment.html>

[2] Logical operators in R

R-bloggers

<http://www.r-bloggers.com/logical-operators-in-r/>

[3] Data Import

Chi Yau, 2016

<http://www.r-tutor.com/r-introduction/data-frame/data-import>

[4] R Library: Matrices and matrix computations in R

IDRE

http://www.ats.ucla.edu/stat/r/library/matrix_alg.htm

[5] Factor variables

Bruin, 2006

http://www.ats.ucla.edu/stat/r/modules/factor_variables.htm

[6] Sorting data

Kabacoff,

<http://www.statmethods.net/management/sorting.html>

[7] Exploratory Data Analysis: Variations of Box Plots in R for Ozone Concentrations in New York City and Ozonopolis

Eric Cai - RBloggers, 2013

<http://www.r-bloggers.com/exploratory-data-analysis-variations-of-box-plots-in-r-for-ozone-concentrations-in-new-york-city-a>

[8] Correlatiecoëfficiënt

M Zandee, 2006

<http://www.mzandee.nl/statistiek/syllabus/correlatiecoëfficiënt.shtml>

[9] Operators

Kabacoff, 2014

<http://www.statmethods.net/management/operators.html>

[10] A Tutorial on Loops in R - Usage and Alternatives

Fanara, 2015

<https://www.datacamp.com/community/tutorials/tutorial-on-loops-in-r>

[11] Help topics

ggplot2

<http://docs.ggplot2.org/current/>

[12] Data visualization with ggplot2

ggplot2

<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

[13] Data Visualization with ggplot2

Datacamp

<https://www.datacamp.com/courses/data-visualization-with-ggplot2-1>

[14] Textual annotations

ggplot2

http://docs.ggplot2.org/current/geom__text.html

[15] Dataportaal van de Nederlandse overheid

Dataportaal van de Nederlandse overheid

<https://data.overheid.nl/>

[16] json-schema.org

json-schema.org

<http://json-schema.org/example1.html>

20 Over de Auteur

Arie Twigt (Leiden, 1990) studeerde Business Administration - Information & Knowledge Management op de Vrije Universiteit Amsterdam. Tijdens zijn studie heeft hij zijn interesse ontwikkeld voor het vakgebied Business Intelligence & Analytics. Hierdoor heeft hij kennis gemaakt met de programmeertaal R. Dit heeft geleid tot veel experimenten en verschillende data gedreven webapplicaties.

De passie voor R heeft Arie meegenomen naar zijn huidige werkplek bij Sogeti Nederland B.V., waar het onderwerp Data Analytics hoog in het vaandel staat. Hierdoor heeft hij R bij verschillende projecten bij organisaties kunnen toepassen en daar complete oplossingen mee gebouwd. Hiermee worden klanten geholpen waardevolle informatie te halen uit data. Daarbij geeft hij verschillende workshops over R aan collega's en klanten.

Naast R heeft hij ook een passie voor sport, vooral voetbal. Het aanschaffen van een Fitbit® is daarom een gebaar van zijn gecombineerde passie voor data en sport.