

GERT JANSSENSWILLEN, BENOÎT DEPAIRE

EXPLORATIEVE EN DESCRIPTIEVE DATA ANALYSE

Contents

<i>Voorwoord</i>	7
<i>Hoe de lecture notes te gebruiken</i>	7
<i>Hoe de tutorials te gebruiken</i>	8
<i>Over de auteurs</i>	8
<i>Disclaimer</i>	8
1 <i>[Lecture notes] Introductiecollege</i>	9
1.1 <i>Data science?</i>	9
1.2 <i>Data & Data types</i>	20
1.3 <i>Referenties</i>	24
2 <i>[Lecture notes] Data visualisatie</i>	25
2.1 <i>Univariate visualisaties (1 variabele)</i>	26
2.2 <i>Bivariate visualisatie (2 variabelen)</i>	36
2.3 <i>Multivariate visualisaties (meer dan 2 variabelen)</i>	49
2.4 <i>Visualisaties voor communicatie</i>	57
2.5 <i>How charts lie</i>	58
2.6 <i>Referenties</i>	58
3 <i>[Tutorial] Data visualisatie</i>	59
3.1 <i>Voor je begint</i>	59
3.2 <i>Introductie</i>	59

4	gert janssenswillen, benoît depaire	
3.3	<i>Verschillende geometries</i>	61
3.4	<i>Layout van onze grafieken verbeteren</i>	76
3.5	<i>Geavanceerde plots</i>	85
3.6	<i>Background material</i>	90
4	<i>[Lecture notes] Descriptieve statistieken</i>	93
4.1	<i>Beschrijvende statistieken versus exploratieve plots</i>	93
4.2	<i>Notatie</i>	93
4.3	<i>Data</i>	93
4.4	<i>Univariate statistieken</i>	93
4.5	<i>Bivariate statistieken</i>	98
	<i>Referenties</i>	102
5	<i>[Tutorial] Descriptieve statistieken</i>	105
5.1	<i>Voor je begint</i>	105
5.2	<i>Introductie</i>	106
5.3	<i>Nuttige dplyr functies</i>	106
5.4	<i>Univariate analyse van een continue variabele</i>	110
5.5	<i>Bivariate analysis of a continuous variable with respect to a categorical variable</i>	114
5.6	<i>Univariate analyse van een categorische variabele</i>	118
5.7	<i>Bivariate analyse van een continue variabele ten opzichte van een andere continue variabele</i>	131
5.8	<i>Bivariate analyse van een categorische variabele ten opzichte van een andere categorische variabele</i>	136
5.9	<i>Background material</i>	146
6	<i>[Lecture notes] Structuur van een data analyse project.</i>	147
7	<i>[Lecture notes] Data voorbereiden</i>	149
7.1	<i>Beginnen bij het begin</i>	149
7.2	<i>Data inlezen</i>	150

7.3	<i>Dataproblemen identificeren en corrigeren</i>	161
7.4	<i>Data opwaarderen</i>	173
	<i>Referenties</i>	180
8	<i>[Tutorial] Data voorbereiden</i>	181
8.1	<i>Voor je begint</i>	181
8.2	<i>Inleiding</i>	182
8.3	<i>Gegevens lezen</i>	182
8.4	<i>Cleaning Data</i>	189
8.5	<i>Data transformatie</i>	216
8.6	<i>Data transformaties tijdens data visualisatie</i>	224
8.7	<i>Achtergrondmateriaal</i>	231
9	<i>[Lecture notes] Tidy data</i>	233
9.1	<i>Inleiding</i>	233
9.2	<i>Case: NYC Vluchten 2013</i>	233
9.3	<i>Data in een lang formaat plaatsen (voor visuele analyses)</i>	236
9.4	<i>Data in een breed formaat plaatsen (voor overzichtelijke tabellen)</i>	241
9.5	<i>Referenties</i>	243
10	<i>[Tutorial] Tidy data</i>	245
10.1	<i>Voor je begint</i>	245
	<i>Disclaimer</i>	245
10.2	<i>Data samenvoegen</i>	245
10.3	<i>Transforming data</i>	258
10.4	<i>[Case study]: WHO</i>	266
11	<i>[Lecture notes] Tijdsdata</i>	279
11.1	<i>Inleiding</i>	279
11.2	<i>Periode-data</i>	280
11.3	<i>Analyseren van tijdgerelateerde data</i>	281
11.4	<i>Referenties</i>	285

<i>12 [Tutorial] Tijdsdata</i>	287
<i>12.1 Voor we beginnen</i>	287
<i>12.2 Introduction</i>	287
<i>12.3 What's in a date?</i>	288
<i>12.4 There's always time</i>	297
<i>12.5 My time is not your time</i>	298
<i>12.6 Extract information from dates</i>	301
<i>12.7 Time differences</i>	302
<i>12.8 Arithmetics</i>	307
<i>12.9 Interval</i>	308

Voorwoord

Dit boek bevat de lecture notes en tutorials voor het opleidingsonderdeel “Exploratieve en Descriptieve Data Analyse” (1ste Ba Handelsingenieur/Handelsingenieur in de Beleidsinformatica) aan de Universiteit Hasselt. De lectures notes dienen ter begeleiding van de hoorcolleges, terwijl de tutorials telkens een vervolg zijn hierop ter voorbereiding van de werkzittingen.

Hoe de lecture notes te gebruiken

Het idee van de lecture notes is om een begeleidende tekst aan te reiken ter ondersteuning van de slide-decks die gebruikt worden tijdens de hoorcolleges. Deze tekst is “bullet-point”-gewijs opgebouwd en helpt het verhaal dat tijdens het hoorcollege wordt verteld terug op te roepen. Daarnaast zal er per hoofdstuk ook een *referentielijst* aangereikt worden met werken die de diverse topics in detail uitleggen.

- Neem de lecture notes mee naar het hoorcollege (digitaal of geprint), en gebruik deze om belangrijke aspecten tijdens het hoorcollege te markeren en korte nota's toe te voegen. Ga zeker niet de volledige uitleg van het hoorcollege noteren. Dit is vaak niet mogelijk en indien je er toch in slaagt zal je tijdens het hoorcollege niet in staat zijn geweest om een eerste keer te reflecteren over de leerstof.
- Bestudeer na de les de lecture notes samen met de notities. Controleer of je alles begrijpt en waar nodig noteer je aanvullingen. Probeer een overzicht te verkrijgen van de diverse concepten die je tijdens het hoorcollege bestudeerd hebt en tracht na te gaan hoe je deze inzichten kunt gebruiken voor exploratieve en descriptieve data analyse.
- (optioneel) Lees de bronnen in de referentielijst. Indien er elementen niet duidelijk zijn in je eigen notities of de lecture notes, dan ga je best gericht op zoek naar de antwoorden op je vragen in de referentiewerken.

Hoe de tutorials te gebruiken

De tutorials zijn een logisch gevolg op de leerstof in het hoorcollege en bereiden je voor op de oefening in de werkzittingen. In de tutorials worden de concepten uit het hoorcollege geïllustreerd in R code. Het is niet enkel de bedoeling de tutorials te lezen, maar ook zelf de voorbeelden uit te proberen in Rstudio. Je vindt de nodige datasets hiervoor telkens terug op Blackboard. Zonder de tutorials grondig te bekijken heeft het geen zin om naar de werkzittingen te komen.

Over de auteurs

dr. Gert Janssenswillen is academisch medewerker aan de faculteit Bedrijfseconomische Wetenschappen (BINF Business Informatics) van de Universiteit Hasselt. Na het verwerven van zijn diploma Handelsingenieur in de Beleidsinformatica in 2014, behaalde hij in 2019 een PhD in de Bedrijfseconomie aan de Universiteit Hasselt. Tijdens zijn doctoraat ontwikkelde hij de open-source R packages-suite bupaR, welke wereldwijd gebruikt wordt door bedrijven en organisaties voor de analyse van bedrijfsprocessen. Hij spreekt regelmatig op business process management - conferenties, zoals BPM, ICPM, en SIMPDA, alsook R conferenties, zoals useR. Sinds 2019 is hij lid van het organisatie comité van de Europese R User Meeting (eRum).

Prof. dr. Benoît Depaire is hoofddocent Beleidsinformatica aan de Universiteit Hasselt en lid van de onderzoeksgroep Beleidsinformatica. Zijn onderzoeksinteresse situeert zich rond de topics data mining, data-gedreven procesanalyse en statistiek met een focus op de extractie van bedrijfskundige inzichten uit data. Als voorzitter van het onderwijsmanagementteam voor de opleiding Beleidsinformatica, alsook op basis van zijn jarenlange ervaring als docent, heeft hij een onderwijsexpertise uitgebouwd rond diverse topics zoals projectmanagement, business process management, data analyse en de rol van IT in de moderne bedrijfswereld. Daarnaast houdt hij zich ook bezig met dienstverlening naar de bedrijfswereld toe door middel van gastlezingen, adviesverstrekking en toegepaste onderzoeksprojecten.

Disclaimer

Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geademtiseerd gegevensbestand en/of openbaar gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnemen of op enige andere manier zonder voorafgaande schriftelijke toestemming van de uitgever.

1

[Lecture notes] Introductiecollege

1.1 Data science?

1.1.1 Netflix

- Netflix Prize (2006)
 - Wereldwijde open competitie voor de constructie van een nieuw algoritme dat moest voorspellen hoe goed een klant een film zou beoordelen op basis van zijn of haar filmvoorkeuren.
 - Winnaar was het team dat als eerste een verbetering van 10% kon realiseren ten opzichte van het algoritme van Netflix zelf.
 - Eerste prijs was 1 miljoen USD.
 - Hiervoor stelde Netflix een dataset ter beschikking met 100 miljoen filmbeoordelingen van 500 000 klanten met betrekking tot 18 000 films.
- Het kunnen voorspellen hoe hun klanten gaan reageren op specifieke films/series laat Netflix toe hun aanbod aan films en series te optimaliseren om het huidige klantenbestand te behouden en nieuwe klanten aan te trekken.
- De hoeveelheid data die door Netflix wordt verzameld is enorm.
 - In 2016 had Netflix 93.8 miljoen leden.
 - Netflix weet wanneer je pauzeert.
 - Netflix weet op welke dagen en welke uren je kijkt.
 - Netflix weet wat je kijkt.
 - Netflix weet van waar je kijkt.
 - Netflix weet op welk soort toestellen je kijkt.
 - Netflix weet wanneer je definitief stopt met het bekijken van een serie.
 - Netflix weet hoe snel je verschillende afleveringen van een serie achter elkaar kijkt.
 - Netflix weet welke titels je zoekt.

- Netflix komt op deze manier zeer veel te weten over het kijkgedrag van zijn klanten en kan op basis van deze inzichten betere beslissingen nemen. Bijvoorbeeld:
 - Netflix ontdekt uit haar data dat 40% van haar klanten een serie zijn beginnen te kijken die door het oorspronkelijke productiehuis is stopgezet.
 - Stel dat Netflix uit de data ook ontdekt dat 85% van deze klanten de serie volledig uitkijken zonder dat het tempo waartegen men afleveringen kijkt significant afneemt.
 - Op basis van deze inzichten kan Netflix eventueel beslissen om de rechten van de serie te kopen (die goedkoop zullen zijn aangezien de serie was stopgezet) en zelf een nieuw seizoen voor de serie te maken.
- House of Cards
 - Netflix deed het beste bod voor de serie House of Cards waardoor het won van kanalen zoals HBO.
 - Ze kochten initieel 2 seizoenen van de serie waar een prijskaartje aan vast hing van meer dan 100 miljoen dollar.
 - Deze beslissing was voor een groot stuk gebaseerd op data:
 - * Netflix leerde uit haar data dat haar klanten geïnteresseerd waren in producties van regisseur David Fincher.
 - * Netflix leerde uit haar data dat haar klanten geïnteresseerd waren in de oorspronkelijke Britse versie van House of Cards.
 - * Netflix leerde uit haar data dat haar klanten geïnteresseerd waren in producties met Kevin Spacey.
 - Maar ook na de beslissing om deze serie te maken, bleef Netflix haar data gebruiken om slimme beslissingen te nemen.
 - * Er werden verschillende trailers gemaakt en afhankelijk van je voorkeuren kreeg je een trailer op maat te zien.
 - * Klanten die vooral graag Kevin Spacey zagen, kregen een trailer waar vooral Kevin Spacey in voorkwam.
 - * Klanten die vooral geïnteresseerd waren in films van David Fincher, kregen een trailer te zien die de typische “look&feel” had van David Fincher.
 - * Klanten die ook de Britse versie hadden gezien, kregen een trailer te zien die vooral op het verhaal focuste.

1.1.2 Waar komt data vandaan?

- Data over een maatschappij
 - Het verzamelen van data is iets dat teruggaat tot in de oudheid.
 - Denk hierbij aan de volkstellingen die reeds plaatsvonden ten tijden van de Romeinen.

- Een volkstelling gaat alle inwoners van een bevolking registreren, samen met diverse kenmerken zoals burgerlijke status, leeftijd, geslacht, enzovoort.
 - Volkstellingen waren en zijn nog steeds belangrijk voor een overheid om de impact van haar openbaar beleid te kunnen inschatten.
- Scientific Management
 - Frederick Taylor
 - Eind 19de eeuw
 - Benaderde het organiseren van werk op een wetenschappelijke manier.
 - Ging data verzamelen om vervolgens te analyseren hoe men werk efficiënter kon organiseren.
 - Een van de eerste vormen van dataverzameling en -analyse om bedrijfswaarde (productiviteit) te creëren.
 - Beperkt in hoeveelheid data omdat registratie en analyse nog manueel gebeurde.
- Het ontstaan van het digitale tijdperk
 - Met de uitvinding van de computer tijdens en na de tweede wereldoorlog, is de mensheid het digitale tijdperk ingegaan.
 - De computer zorgt ervoor dat we data in een digitale vorm (als een reeks van één en nullen) opslaan. Dit biedt het voordeel dat exacte kopieën van de data gemaakt kunnen worden met één muisklik.
- Digitalisatie van de werkvloer
 - Computers op de werkvloer dateert terug tot midden vorige eeuw, maar de grote doorbraak komt er met de opkomst van de personal computer
 - * 1977: Apple Home Computer II
 - * 1981: IBM Personal Computer
 - * Eind jaren 80, begin jaren 90 was de PC wijdverspreid op de werkvloer.
 - * Dit liet toe meer data te registreren, maar deze was nog moeilijk te delen met andere computers.
 - Opkomst Internet / WWW in de bedrijfswereld
 - * 1990: De technologie voor WWW werd publiek gedeeld door Tim Berners-Lee.
 - * Dankzij WWW en internettechnologie werd het steeds een- voudiger om digitaal werk te delen.
 - Opkomst van e-commerce
 - * 1995: Begin van dot-com bubble/hype.

- * Opkomst van digitale ondernemingen (vb. Amazon, Netflix, Google, ...).
- * Digitale handel maakt het eenvoudiger om gegevens hierover te registreren.

- Digitalisatie van mensen

- Opkomst Web 2.0 (begin 2000)
 - * Inhoud van het web wordt nu gecreëerd door de bezoekers/gebruikers/klanten.
 - * Websites worden dynamisch (passen zich aan de context en bezoeker aan).
- Opkomst sociale media
 - * Gebruikers gaan spontaan hun leven digitaliseren.
 - * Hiervoor worden diverse media gebruikt (foto, video, tekst, ...).
 - * Facebook, Twitter, Instagram, Persoonlijke blogs,
 - * Nog nooit heeft zo'n groot deel van de wereldbevolking informatie gecreëerd en gedeeld met de rest van de wereld.

- Digitalisatie van dingen

- Opkomst goedkope sensoren
- Steeds meer “dingen” (machines, auto’s, huishoudtoestellen, huizen, steden, . . .) worden ‘intelligent’.
- Internet of Things (IoT): Al deze intelligente dingen worden via het Internet met elkaar verbonden.
- De hoeveelheid data die hiermee gegenereerd zal worden is ongezien.
- Volgens IDC studie waren in 2013 reeds 7% van de “verbindbare dingen” geconecteerd aan het Internet of Things.
- In dezelfde studie voorspellen ze dat dit zal stijgen tot 15% in 2020.
- In 2013 werd 2% van alle data in het digitaal universum geproduceerd door het IoT.
- Verwacht wordt dat dit zal stijgen tot 10% in 2020.

1.1.3 Hoeveel data is er beschikbaar?

- De hoeveelheid data die de laatste decennia gegenereerd en opgeslagen wordt is enorm toegenomen.
- Deze groei is exponentieel (de groei gaat steeds sneller). Meer specifiek verdubbelt de hoeveelheid data in het digitaal universum iedere 2 jaar.
- Volgens een studie van IDC, bestond het digitaal universum in 2013 uit 4.4 Zetabytes data

- 1 Zetabyte = 1024 Exabytes
- 1 Exabyte = 1024 Petabytes
- 1 Petabyte = 1024 Terabytes
- 1 Terabyte = 1024 Gigabytes
- Volgens dezelfde studie zal het digitaal universum in 2020 uit 44 Zetabytes bestaan
- Echter, slechts 22% van deze data (in 2013) is geschikt voor analyse.
- Er wordt geschat dat dit zal stijgen tot 35% in 2020.
- Slechts 5% van de geschikte data voor analyse wordt feitelijk geanalyseerd (2013).

1.1.4 Waar wordt data voor gebruikt in de bedrijfswereld?

- Er zijn verschillende redenen waarom bedrijven data bijhouden. Deze kunnen we onderverdelen in volgende categorieën: Geschiedenis bijhouden, beslissingen nemen en voorspellingen maken.

Geschiedenis bijhouden

- Je registreert feiten zodat je achteraf met zekerheid kunt weten wat de realiteit in het verleden was.
- Dit is belangrijk als je wilt evalueren of een bedrijf goed beheerd wordt. Hiervoor heb je inzicht in het verleden nodig.
- De gegevens die worden bijgehouden in een boekhouding en jaarrekeningen zijn hier een typisch voorbeeld van.

Dagelijkse werking

- Opdat een bedrijf zijn dagelijkse werking kan uitvoeren, is het essentieel een up to date zicht te hebben van de werkelijkheid. Als een klant belt met een klacht over een levering, dan moet je als onderneming kunnen achterhalen wat de klant precies besteld heeft, of dit reeds geleverd is, of de klant al betaald heeft, enzovoort. Zonder deze informatie kan een onderneming haar dagelijkse werking niet garanderen.
- Om de dagelijkse werking te verzekeren, hebben bedrijven altijd al data bijgehouden. Denk maar aan informatie over aankoop- en verkooporders, de financiële gegevens in de boekhouding, de afschriften van een bank, de productieplanning, enzovoort.

Beslissingen nemen

- Een bedrijf neemt dagelijks talrijke beslissingen op verschillende niveaus

- Operationeel.
 - * Vb: Moet ik een nieuwe bestelling plaatsen voor grondstof X of hebben we nog genoeg voorraad?
 - * Dit zijn typisch zeer frequente beslissingen die nodig zijn om de dagelijkse werking te garanderen.
 - * Deze beslissingen worden genomen door mensen op de werkvloer of door het (lager) management.
- Tactisch/Management.
 - * Vb: Sluit ik best een exclusief contract af met 1 leverancier voor grondstof X voor een vaste periode en tegen een vaste verkoopsprijs of koop ik wanneer nodig tegen de marktprijs?
 - * Deze beslissingen worden minder frequent genomen dan operationele beslissingen en zijn typisch nodig om de werking van de onderneming op middellange termijn te optimaliseren.
 - * Deze beslissingen worden genomen door het management van een onderneming en hebben een aanzienlijke impact.
- Strategisch.
 - * Vb: Zullen we grondstof X aankopen op de markt of beslissen we deze grondstof zelf te produceren?
 - * Deze beslissingen hebben een zeer grote impact op de onderneming en worden niet frequent genomen. Ze vergen typisch ook lange voorbereidingstijd en bepalen de richting en toekomst van de onderneming op lange termijn.
 - * Deze beslissingen worden genomen door het topmanagement van een onderneming.
- Data kan bedrijven helpen bij het nemen van beslissingen.
 - Dit betekent echter niet dat beslissingen enkel en alleen op data gebaseerd zijn.
 - Vaak wordt data gecombineerd met ervaring en expertise om een beslissing te nemen.
- Bij het nemen van beslissingen op basis van data, kunnen we zowel patronen in historische data gebruiken alsook voorspellingen op basis van data.

1.1.5 Waarover verzamelen bedrijven data

- Het ultieme doel van een onderneming is gegevens te verzamelen die hen toelaten om het gedrag van hun omgeving beter te begrijpen, alsook de werking van hun eigen onderneming.
- Onder omgeving verstaan we:
 - Klanten
 - Concurrenten

- Leveranciers
- Alternatieve markten
- Overheden
- Onder werking van eigen onderneming vertaan we o.a.:
 - Werknemers
 - Processen
 - Producten
 - Diensten

1.1.6 Van data tot ‘actionable insights’

- Data
 - Data verwijst typisch naar de gegevens die geregistreerd en opgeslagen worden.
 - Data beschrijft een heel klein aspect van een realiteit (bijvoorbeeld op welk exact tijdstip ben ik aflevering 2 van “House of Cards” beginnen te kijken).
 - Data op zich heeft echter heel weinig waarde.
- Informatie
 - Als we echter data gaan analyseren, dan kunnen we dit transformeren tot informatie.
 - Informatie beschrijft een realiteit en gaat typisch op zoek naar patronen in de data en afwijkingen op deze patronen.
 - Bijvoorbeeld: Ik kijk typisch House of Cards gedurende de week om 20u00 's avonds, maar stop meestal met kijken om 20u30, waardoor ik in de week zelden een aflevering in 1 keer uitkijk.
 - Informatie is beschrijvend en zegt ons WAT de realiteit is.
- Actionable Insights
 - Actionable Insights is informatie die ons niet enkel zegt WAT de realiteit is, maar ons ook het inzicht verschafft HOE we moeten handelen.
 - Niet alle informatie is actionable.
 - Op basis van actionable insights en in combinatie met onze eigen ervaringen en kennis die we reeds bezitten, komen we soms tot inzichten die beschrijven HOE we moeten handelen.

1.1.7 Data Scientists

- Nieuwe jobomschrijving.
- Verantwoordelijk om data te transformeren naar ‘actionable insights’ en hier iets mee te doen om bedrijfswaarde te creëren.
- Omschreven als meest ‘sexy job’ van de 21ste eeuw door HBR

- Opvolgers van de Wall Street ‘Quants’ uit de jaren 80 en 90.
- Vaardigheden
 - Bedrijfskunde
 - * Productontwikkeling
 - * Management
 - Machine Learning / Big Data
 - * Ongestructureerde data
 - * Gestruktureerde data
 - * Machine Learning
 - * Big Data
 - Wiskunde en Operationeel Onderzoek
 - * Optimalisatie
 - * Wiskunde
 - * Simulatie
 - Programmeren
 - Statistiek
 - * Visualisatie
 - * Tijdreeksanalyse
 - * Wetenschappelijk onderzoek
 - * Data Manipulatie
- 4 profielen van data scientists
 - Data Businessperson
 - * Focust voornamelijk hoe data omzet kan genereren.
 - * Vaak in een leidinggevende rol.
 - * Werken zelf ook met data en beschikken over de nodige technische vaardigheden.
 - Data Creatives
 - * Zijn in staat een volledige data analyse zelfstandig uit te voeren.
 - * Hebben een hele brede bagage aan technische vaardigheden.
 - * Beschikken in zekere mate over bedrijfskundige vaardigheden.
 - * Gaan vaak innovatief om met data.
 - Data Developer
 - * Is voornamelijk gefocust op de technische uitdagingen met betrekking tot het beheer van data.
 - * Sterke programmeervaardigheden. Zijn in staat productie-code te schrijven.
 - * Zijn sterk in het gebruik van machine learning technieken.
 - Data Researcher

- * Vaak mensen met een wetenschappelijke achtergrond (doctoraat).
- * Sterk in statistische vaardigheden en wetenschappelijk onderzoek.

1.1.8 Verschillende soorten van data analyse

- Er zijn verschillende manieren om data analyse taken te classificeren.
- De classificatie die we hier hanteren is gebaseerd op het doel van de data analyse.

Descriptieve data analyse

- Deze analyse focust zich op het beschrijven van de data.
- Deze analyse gaat over het samenvatten van de grote hoeveelheid data in enkele statistische cijfers en grafieken.
- Deze analyse wordt gebruikt als je een grote hoeveelheid data krijgt en je snel inzicht wilt krijgen in de data.
- Voorbeelden:
 - Je hebt een dataset met alle studieresultaten van de studenten van 1ste bachelor HI/BI en je wilt weten wat de gemiddelde score is per vak.
 - Je hebt de verkoopscijfers van het afgelopen jaar en je wil weten welke drie producten het beste verkochten (zowel in aantal als in omzet).
- Descriptieve data analyse zegt alleen iets over de realiteit die door de data is beschreven. Je kan **geen** conclusies trekken die verder reiken dan de geobserveerde data.
- Je kan een descriptieve data analyse vergelijken met het werk van een detective die als taak heeft een beschrijving te maken van de misdaadscene.

Exploratieve data analyse

- Exploratieve analyse focust op het verkennen van de data en het zoeken naar interessante patronen en afwijkingen van deze patronen.
- Net als bij descriptieve data analyse zal exploratieve analyse de beschikbare data beschrijven en zeggen de resultaten **niets** over ongeobserveerde feiten.
- In tegenstelling tot bij descriptieve data analyse, gaat exploratieve data analyse verder dan het louter beschrijven van de data en tracht men interessante patronen te ontdekken in de data.
- Voorbeelden:

- Zijn er specifieke kenmerken van studenten die sterk gerelateerd zijn aan hun studieresultaten.
- Zijn er opmerkelijke verschillen tussen vakken wat betreft de punten die behaald worden. Zo ja, wat zijn dan deze verschillen.
- Zijn er producten in ons gamma die gevoelig zijn voor seizoenseffecten?
- Je kan een exploratieve data analyse vergelijken met het werk van een detective die als taak heeft verbanden te ontdekken tussen verschillende bewijsstukken om zo inzicht te verschaffen wat er gebeurd is tijdens de misdaad.

Confirmatorische data analyse

- Confirmatorische analyse focust op het bevestigen of weerleggen van vermoedens die men heeft met behulp van de beschikbare data.
- In tegenstelling tot descriptieve en exploratieve data analyse zal men bij confirmatorische data analyse wel conclusies trekken die verder gaan dan de geobserveerde data.
- Omdat confirmatorische data analyses ook uitspraken doen over ongeobserveerde data, is er altijd een mate van onzekerheid over de correctheid van de resultaten.
- Voorbeelden:
 - Halen studenten met 8u Wiskunde achtergrond betere resultaten dan studenten met 6u Wiskunde achtergrond? In welke mate zijn we zeker dat dit voor alle studenten geldt en niet enkel voor de studenten waarover we data hebben?
 - Verkoopt product X beter bij mannen dan bij vrouwen? In welke mate zijn we zeker dat dit verschil niet een toevalligheid in de data is?
- Je kan een confirmatorische data analyse vergelijken met het werk van een rechter die op basis van het aangeboden bewijsmateriaal moet beslissen of er genoeg bewijs is om iemand te veroordelen van de misdaad.

Predictieve data analyse

- Het doel van predictieve analyse is om op basis van de beschikbare data voorspellingen te doen over de toekomst of over nieuwe of alternatieve situaties.
- Net als bij confirmatorische data analyse zal predictieve data analyse uitspraken doen die ook van toepassing zijn voor ongeobserveerde feiten/situaties.
- Bijgevolg is er net als bij confirmatorische data analyse dus een zekere onzekerheid over de conclusies die men trekt.

- Voorbeelden:
 - Zal een studente die met meer dan 80% haar diploma van het middelbaar onderwijs behaalt slagen in eerste zit voor het vak *Exploratieve en Descriptieve Data Analyse*?
 - Zullen de verkoopcijfers van product Y het komende jaar verder stijgen en met hoeveel procent?
- Je kan een predictieve data analyse vergelijken met het werk van een detective die op basis van het bewijsmateriaal op een misdaadscene moet voorspellen waar en wanneer de dader opnieuw zal toeslaan.

1.1.9 De kunst van data analyse

- Data analyse is een kunst. Net als bij iedere kunst, kunnen we hierbij drie componenten onderscheiden: kennis en vaardigheden, ervaring en creativiteit.
- Kennis en vaardigheden
 - Als data analist moet je de juiste hulpmiddelen kunnen identificeren voor het voorgelegde probleem.
 - Deze diverse hulpmiddelen moet je zo goed mogelijk beheersen.
 - Bij (exploratieve) data analyse gaat het hierbij zowel over analyses als over datavaardigheden.
 - Dit aspect kun je leren en laat je reeds toe om correcte analyses uit te voeren.
- Ervaring
 - Hoe meer data je analyseert, hoe beter je er in wordt.
 - Ook laat ervaring toe om sneller vaste patronen in je werk te herkennen en efficiënter te worden in wat je doet.
 - Ervaring is ook essentieel om complexere uitdagingen beheersbaar te maken.
 - Dit deel kunnen we je niet ‘leren’, maar heb je wel volledig in de hand.
- Creativiteit
 - Een kunstenaar die over kennis, vaardigheden en ervaring beschikt, maar creativiteit ontbreekt, kan perfecte replica’s maken van een kustwerk, maar kan zelf geen nieuwe kunst creëren.
 - Creativiteit is in staat zijn op een nieuwe en onverwachte manier naar data te kijken en deze te visualiseren.
 - Het is niet zeker dat dit aspect aan te leren is. Maar dit hoeft niet te verhinderen dat je een goede data scientist wordt, zolang je maar voldoende aandacht besteedt aan de andere twee componenten.

1.1.10 De kracht van descriptieve en exploratieve data analyse

<https://www.youtube.com/watch?v=RUwS1uAdUcI>

1.2 Data & Data types

- Data is het resultaat van een meting van een attribuut van een specifiek object met een specifiek meetinstrument.
 - Het object verwijst naar wat je gaat meten.
 - * vb.: Student “Karel Jespers”.
 - Een object hoort meestal tot een verzameling van objecten. Deze verzameling wordt ook wel de populatie genoemd.
 - * vb.: Populatie “Studenten 1ste Ba HI/BI”.
 - Een specifiek object uit de populatie wordt ook wel element genoemd.
 - * vb.: “Karel Jespers” is een element uit de populatie “Student 1ste Ba HI/BI”.
 - Je meet altijd een specifiek aspect van het object. Omdat de meetwaarde van dit aspect kan variëren tussen verschillende objecten (elementen) in je verzameling (populatie), worden zulke aspecten ook variabelen genoemd.
 - * vb.: Lengte is een specifiek aspect (variabele) van de student “Karel Jespers” (element).
 - De meting gebeurt met behulp van een meetinstrument. Het is belangrijk te beseffen dat een meetinstrument altijd een zekere nauwkeurigheid heeft (tot hoeveel cijfers na de komma exact kan je meten?) en mogelijk ook onderhevig kan zijn aan willekeurige en/of systematische meetfouten.
 - * vb.: Student “Karel Jespers” wordt gemeten met een meetlat bevestigd tegen de muur. De meetlat heeft een nauwkeurigheid van 1cm, dus we kunnen zijn lengte niet uitdrukken in milimeters. Verder is de meetlat 2cm te laag opgehangen. Bijgevolg is er een systematische meetfout van 2cm. Tenslotte wordt de meting geregistreerd door een arts die vluchtig kijkt waar de student uitkomt op de meetlat. Het is dus niet onmogelijk dat de werkelijke lengte (willekeurig) afwijkt van de geregistreerde lengte.
 - * Tenzij anders vermeld wordt, gaan we in dit hoofdstuk uit van meetinstrumenten met oneindige nauwkeurigheid en zonder meetfouten.
 - De uitkomst van een meting voor een specifiek element wordt de waarde genoemd.

- * vb.: 1m80 is de waarde van de variabele “lengte” voor element “student Karel Jespers”

1.2.1 Dataset

- Een dataset is een verzameling van data waarbij
 - Iedere rij één element uit de populatie voorstelt.
 - Iedere kolom een variabele is die gemeten wordt.
 - De verschillende rijen verschillende elementen uit dezelfde populatie voorstellen.
 - De waarde in een cel de meting is van de betreffende variabele voor het betreffend element.

luchthaven	maatschappij	datum	vertrek_vertraging	aankomst_vertraging	afstand	vliegtijd
EWR	United Air Lines Inc.	2013-01-01 05:15:00	2	11	1400	227
LGA	United Air Lines Inc.	2013-01-01 05:29:00	4	20	1416	227
JFK	American Airlines Inc.	2013-01-01 05:40:00	2	33	1089	160
LGA	Delta Air Lines Inc.	2013-01-01 06:00:00	-6	-25	762	116
EWR	United Air Lines Inc.	2013-01-01 05:58:00	-4	12	719	150
EWR	JetBlue Airways	2013-01-01 06:00:00	-5	19	1065	158
LGA	ExpressJet Airlines Inc.	2013-01-01 06:00:00	-3	-14	229	53
JFK	JetBlue Airways	2013-01-01 06:00:00	-3	-8	944	140
LGA	American Airlines Inc.	2013-01-01 06:00:00	-2	8	733	138
JFK	JetBlue Airways	2013-01-01 06:00:00	-2	-2	1028	149

Tabel 1.1: Uitgaande vluchten NYC

1.2.2 Klassieke datatypologie

- Klassieke onderverdeling van data
 - Nominaal, Ordinaal, Interval en Ratio
 - Gebaseerd op de publicatie “On the Theory of Scales of Measurement” (1946)
 - * Beschrijft een hiërarchie van ‘datatypes’
 - Alles wat ordinaal is, is ook nominaal, maar niet omgekeerd.
 - Alles wat interval is, is ook ordinaal, maar niet omgekeerd.
 - Alles wat ratio is, is ook interval, maar niet omgekeerd.
 - * Identificeert geschikte statistische testen voor ieder type.
- Ieder datatype voldoet aan één of meerdere van de volgende eigenschappen:
 - Identiteit: Iedere waarde heeft een unieke betekenis.
 - Grootorde: Er is een natuurlijke volgorde tussen de waarden.

- Gelijke intervallen: Eenheidsverschillen zijn overall even groot. Dus het verschil tussen 1 en 2 is even groot als het verschil tussen 19 en 20.
- Absoluut nulpunt: De waarde 0 betekent dat er ook feitelijk niets aanwezig is van de variabele en is niet een arbitrair gekozen nulpunt.

Nominaal

- Voorbeelden:
 - Geslacht: Man, Vrouw.
 - Ondernemingsvorm: vzw, bvba, nv.
- Voldoet enkel aan de eigenschap ‘identiteit’.
- Dit betekent dat we enkel concluderen of twee waardes gelijk zijn of niet. Er bestaat geen natuurlijke volgorde tussen de verschillende waardes.

Ordinaal

- Voorbeeld:
 - Opleidingsniveau: Lager onderwijs, Middelbaar onderwijs, Hoger onderwijs.
 - Klanttevredenheid: Ontevreden, Matig tevreden, Tevreden, Zeer tevreden.
- Voldoet aan de eigenschappen ‘identiteit’ en ‘grootorde’.
- Dit betekent dat we niet alleen kunnen concluderen of twee waardes gelijk zijn of niet. Het is ook mogelijk te bepalen welke waarde ‘groter’ is.
- We kunnen echter niet zeggen hoeveel groter één waarde is dan de andere.

Interval

- Voorbeeld:
 - Temperatuur (Celsius).
- Voldoet aan de eigenschappen ‘identiteit’, ‘grootorde’ en ‘gelijke intervallen’.
- We kunnen nu twee waardes vergelijken, bepalen welke groter is alsook de verschillen tussen waardes met elkaar vergelijken.
 - We kunnen dus stellen dat het verschil tussen 8 en 9 graden Celsius daadwerkelijk minder groot is dan het verschil tussen 12 en 20 graden Celsius.

Ratio

- Voorbeeld:
 - Gewicht
- Voldoet aan alle 4 de eigenschappen.
- We kunnen verschillende gewichten met elkaar vergelijken, we kunnen bepalen wat zwaarder is en we kunnen gewichtsverschillen onderling vergelijken. Hierbij komt nu ook nog dat we kunnen zeggen hoeveel keer iets zwaarder is dan iets anders.
- Dit is een gevolg van het feit dat de waarde 0 nu feitelijk betekent dat iets geen gewicht heeft.

1.2.3 De klassieke datatypologie is misleidend

- Voorbeeld:
 - Op een feestje wordt bij het binnengaan oplopende nummers toegewezen aan iedere gast, beginnend bij 1.
 - Tijdens het feestje wordt er een tombola georganiseerd en wie nummer 126 heeft, heeft gewonnen.
 - 1 gast vergelijkt dit nummer met haar kaartje en ziet dat ze gewonnen heeft. Zij beschouwde de waarde op haar ticket dus als een nominale variabele want het enige wat ze vergelijkt is of de waarde op haar ticket verschillend is van de winnende waarde.
 - Een andere gast kijkt naar zijn kaartje en ziet dat hij nummer 56 heeft. Hij concludeert dat hij te vroeg is binnengekomen en beschouwt de waarde op zijn kaartje dus als ordinaal.
 - Nog een andere gast heeft een kaartje met nummer 70 en beschikt over bijkomende data omtrent het ritme waarmee gasten zijn binnengekomen. Deze gast kan dus schatten hoeveel later hij had moeten binnengekomen om te winnen en interpreteert zijn nummer dus als een interval variabele.
- Dit voorbeeld illustreert dat het datatype niet een vaststaand kenmerk is van de data, maar afhankelijk is van de vraag die je tracht te beantwoorden en de extra informatie waarover je beschikt.

1.2.4 Alternatieve datatypologie

- Alternatieve taxonomie van data
 - Graden: vb. academische graad: “op voldoende wijze”, “onderscheiding”, “grote onderscheiding”, . . . (geordende labels)
 - Rangordes: vb. plaats in voetbalklassemement: 1, 2, 3, . . . , 16 (gehele getallen die beginnen bij 1)
 - Fracties: vb. percentage opgenomen verlof: van 0% tot 100% (ligt tussen 0 en 1, als percentage uit te drukken).

- Aantallen: vb aantal kinderen: 0, 1, 2, ... (niet-negatieve gehele waarden).
- Hoeveelheden: vb. inkomen (niet-negatieve reële waarden).
- Saldo: vb. winst (negatieve en positieve reële waarden).
- Voor deze cursus volstaat het meestal een onderscheid te maken tussen nominale en ordinale variabelen (samen “categorische” variabelen) en continue variabelen
 - Categorisch:
 - * Nominaal
 - * Ordinaal.
 - Continu: (Interval + Ratio)

1.3 Referenties

1. Data Scientist, the Sexiest Job of the 21st Century
2. Netflix Prize
3. How Netflix Uses Analytics
4. The Digital Universe of Opportunities - website
5. The Digital Universe of Opportunities - videoclip
6. How the Computer Changed the Office Forever
7. History of Computers in the Workplace
8. Web 1.0, 2.0, 3.0
9. From Data to Understanding
10. Analyzing the Analyzers
11. Scales of Measurement
12. Nominal, Ordinal, Interval, and Ratio Typologies are Misleading

2

[Lecture notes] Data visualisatie

- Vaak de eerste stap om zicht te krijgen op de data.
- Relatief eenvoudig om patronen te zien, maar minder geschikt om exacte waarden te zien.
- We moeten hierbij onderscheid maken tussen exploratieve visualisaties en informatieve visualisaties om een boodschap over te brengen.
 - Exploratieve visualisaties dienen om snel inzicht te krijgen in patronen in de data. Men besteedt hierbij veel minder aandacht aan de opmaak van de visualisatie. Vaak is deze visualisatie tijdelijk en niet bedoeld voor communicatie naar derden.
 - Communicatieve visualisaties dienen om een boodschap over te brengen aan derden. Hier dient men heel veel aandacht te besteden aan de opmaak zodat de boodschap duidelijk en helder gecommuniceerd wordt.
- We kunnen bij exploratieve visualisaties een onderscheid maken tussen univariate, bivariate en multivariate visualisaties.

De grafieken in dit hoofdstuk zijn gebaseerd op volgende dataset omrent vluchten vertrekkende uit New York.

```
## Rows: 329,174
## Columns: 7
## $ luchthaven      <fct> EWR, LGA, JFK, LGA, EWR, EWR, LGA, JFK, LGA, JF...
## $ maatschappij    <chr> "United Air Lines Inc.", "United Air Lines Inc...."
## $ datum           <dttm> 2013-01-01 05:15:00, 2013-01-01 05:29:00, 2013...
## $ vertrek_vertraging <dbl> 2, 4, 2, -6, -4, -5, -3, -3, -2, -2, -2, -2...
## $ aankomst_vertraging <dbl> 11, 20, 33, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## $ afstand          <dbl> 1400, 1416, 1089, 762, 719, 1065, 229, 944, 733...
## $ vliegtijd         <dbl> 227, 227, 160, 116, 150, 158, 53, 140, 138, 149...
```

2.1 Univariate visualisaties (1 variabele)

- Als we slechts 1 variabele bestuderen, dan zijn we voornamelijk geïnteresseerd in de spreiding van de data. Dit wordt de verdeling van de data genoemd.
- Welke vragen kunnen we beantwoorden met dit soort visualisaties?
 - Wat is de meest voorkomende waarde van de data? Dit wordt ook de modus genoemd.
 - Bezit de data 1 modus, i.e. 1 waarde die duidelijk dominant is, of meerdere modi?
 - * Indien er slechts 1 afgetekende modus is, dan wordt de verdeling unimodaal genoemd.
 - * Indien er meerdere modi zijn (dominante waarden), dan wordt de verdeling multimodaal genoemd.
 - * Een multimodale verdeling kan er op wijzen dat de objecten in je data niet allemaal van hetzelfde type zijn en dat je in feiten twee populaties in je data aanwezig hebt.
 - Is de data geconcentreerd rond de modus of eerder breed verspreid. Met andere woorden, wat is de spreiding? Dit geeft inzicht in de variabiliteit van de data.
 - Is de data gelijkmataig verdeeld aan weerszijden van de modus of zien we duidelijk meer data aan één zijde van de verdeling? Indien er meer data aan één zijde van de verdeling ligt (ten opzichte van de modus) dan zegt men dat de verdeling asymetrisch verdeeld is.
 - Zijn er waardes die opmerkelijk ver van de modus verwijderd zijn en geïsoleerd zijn van andere observaties? Dit worden extreme waarden of outliers genoemd. Deze verdienen meestal extra aandacht.

2.1.1 Categorische variabele

Staafdiagram

- Op de X-as staan de verschillende waarden van de categorische variabele. (Fig. 2.1)
- Bij iedere waarde tekenen we een verticale balk die aangeeft hoe vaak die waarde in de dataset voorkomt.
- Minder geschikt indien er veel waarden zijn. Dan wordt de X-as snel onleesbaar. (Fig. 2.2)
- Je kan natuurlijk de labels roteren. Maar dit kan nog steeds onhandig zijn om te lezen. (Fig. 2.3).

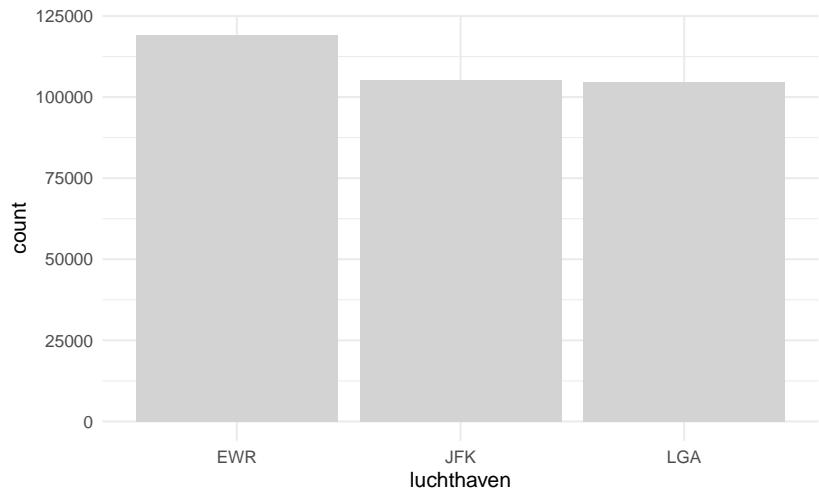


Figure 2.1: Staafdiagram luchthavens

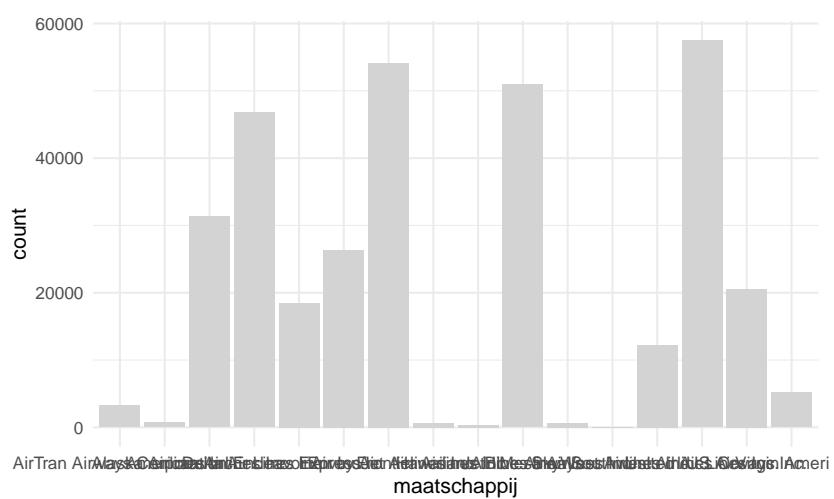


Figure 2.2: Staafdiagram maatschappijen

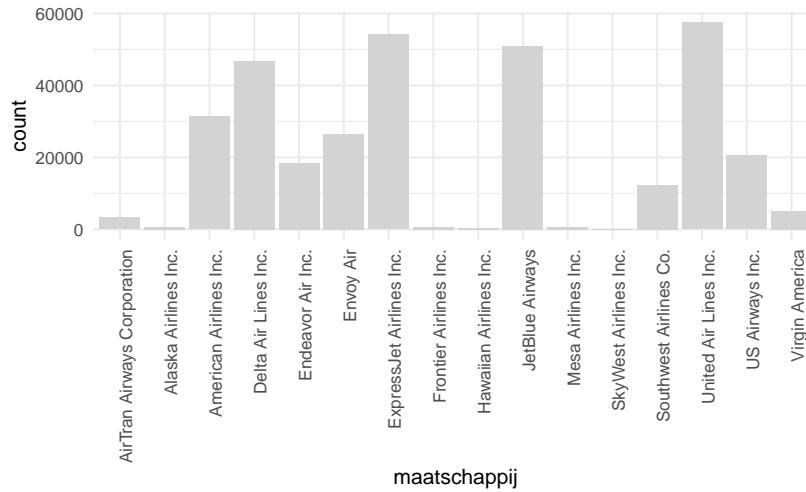


Figure 2.3: Staafdiagram met geroteerde labels

- In geval van een **nominale** variabele zijn er twee mogelijkheden om de waarden te rangschikken:
 - Alfabetisch. (standaard) Dit is handig om snel waarden terug te vinden.
 - Volgens frequentie. Dit is handig om snel te zien welke waarden vaak/weinig voorkomen en geeft ook een beter beeld van de verdeling van de waarden. (Fig. 2.4)

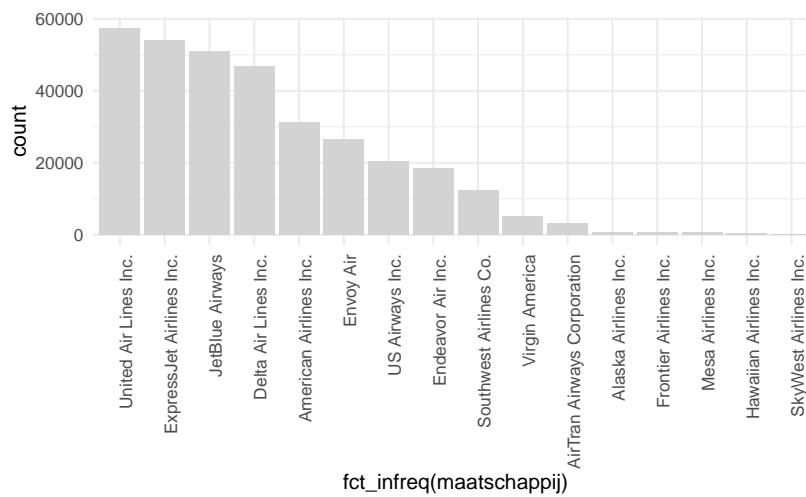


Figure 2.4: Staafdiagram gesorteerd op frequentie

- In het geval van een **ordinale** variabele houd je best de intrinsieke volgorde van de waarden aan.
- Je kan ook een horizontaal staafdiagram maken. (Fig. 2.5)

- Zelfde principe, maar dan met horizontale balken.
- Is handiger om de verschillende waarden te lezen, vooral indien dit er veel zijn.

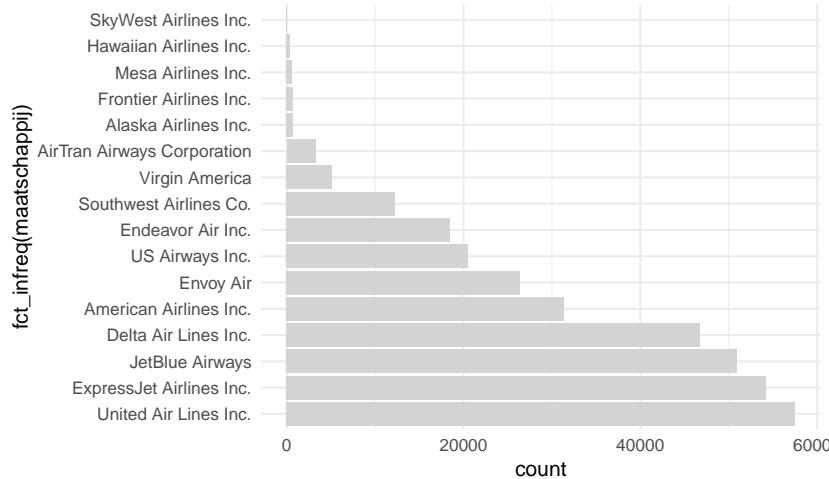


Figure 2.5: Verticaal staafdiagram gesorteerd op frequentie

Dotplot

- In plaats van balken te gebruiken om de frequentie van een waarde aan te geven, kan je dit ook met punten doen. (Fig. 2.6)
- Een dotplot laat duidelijker zien waar de sprongen in de verdeling zit. Daarom is de dotplot vooral relevant als je de waarden ordent volgens frequentie.

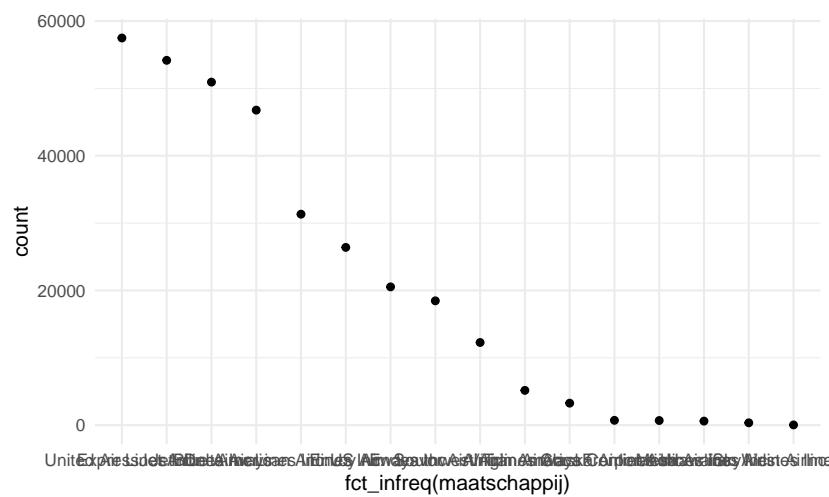


Figure 2.6: Dotplot maatschappij

- Net als de barplot kan je zowel een verticale als horizontale dotplot maken. (Fig. 2.7)

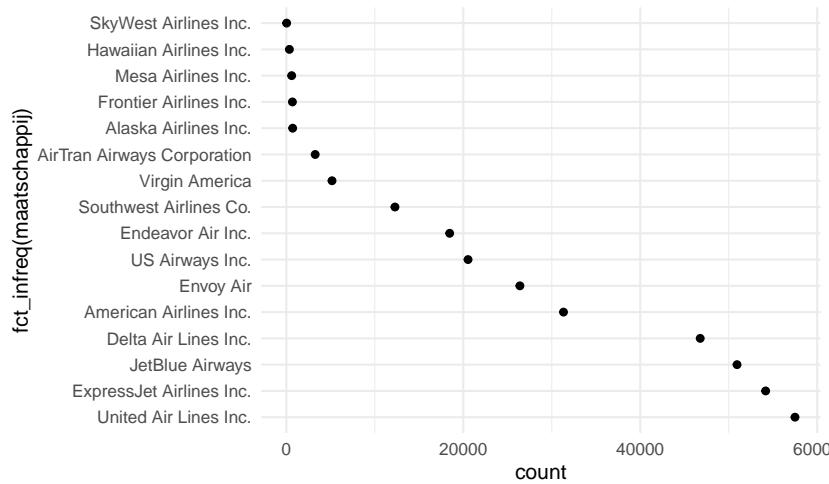


Figure 2.7: Verticale dotplot

'Stacked' staafdiagram

- We maken nu slechts 1 kolom. Iedere waarde is een andere kleur en neemt een deel van de balk in beslag. De volledige balk stelt 100% van de data voor. (Fig. 2.8)
- Kan nuttig zijn om data cumulatief te bestuderen.
- Hiermee kunnen we vragen beantwoorden zoals: "Welke waarden moeten we nemen om met zo weinig mogelijk waarden x% van de objecten te hebben?"
- We kunnen ook horizontale versies maken. (Fig. 2.9)
- Univariate stacked barcharts kunnen soms wat *raar* overkomen.
Vaak komt een gewone barchart beter over.

Andere soorten

- treemap: indelen van rechthoekige oppervlakte volgens categorische variabelen
- pie chart
 - Moeilijk te interpreteren.
 - Verschillen tussen waarden zijn enkel duidelijk bij grote verschillen, terwijl barplots en dotplots deze ook bij kleine verschillen kunnen tonen.
 - Voor cumulatieve analyses van de data zijn barplots beter omdat het hier eenvoudiger is om af te leiden waar x% zicht bevindt.

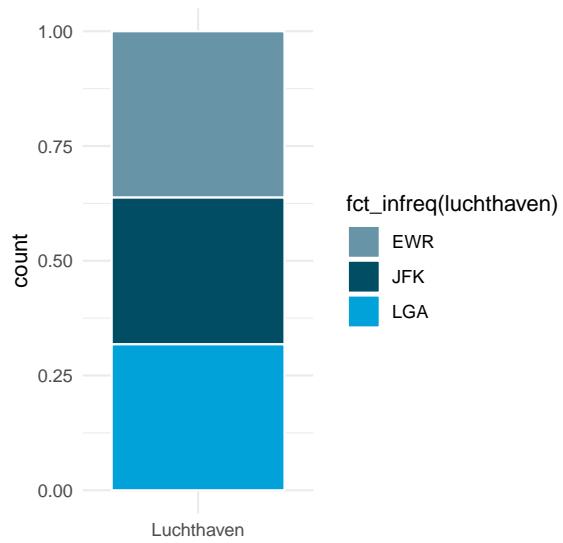


Figure 2.8: Stacked barplot

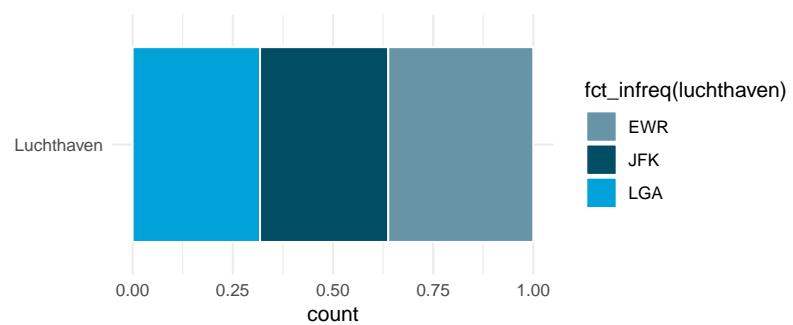


Figure 2.9: Horizontale stacked barplot

2.1.2 Continue variabele

Histogram

- Analoog met barplot, alleen gaan we hier eerst onze “categorieën” definiëren. (Fig. 2.10)
- Dit wordt ‘binning’ genoemd en wordt bepaald door een bin-breedte te kiezen.
- Je kan de binbreedte rechtstreeks kiezen of bepalen door vast te leggen hoeveel categorieën/bins je wenst.

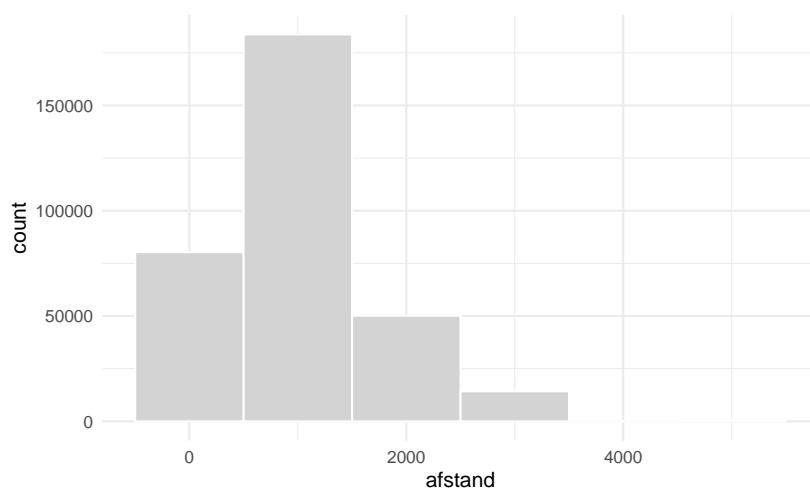


Figure 2.10: Histogram with binwidth 1000

- Voor de visualisatie, worden alle waarden gegroepeerd per ‘bin’.
- De binbreedte kan een enorme impact hebben op het uitzicht van de verdeling. (Fig. 2.11 - 2.12)
 - Hoe breder de bins, hoe minder modi je kan detecteren.
 - Hoe smaller de bins, hoe meer modi je gaat zien, hoewel dit niet altijd even betekenisvol is.
 - Hoe smaller de bins, hoe minder data er in iedere bin gaan zitten en dan kunnen patronen wel in jouw dataset bestaan maar louter ten gevolge van toeval.

Density

- Variant van histogram.
- In plaats van staven wordt er een curve getekend. (Fig. 2.13)
- De oppervlakte onder de curve is steeds gelijk aan 1
- Hoe hoger de curve, hoe meer observaties ter hoogte van deze waarde (hoe hoger de densiteit)

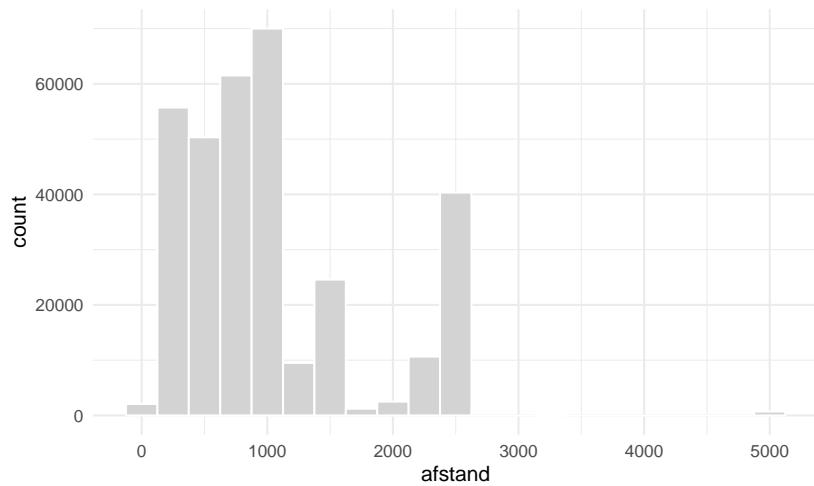


Figure 2.11: Histogram with binwidth 250

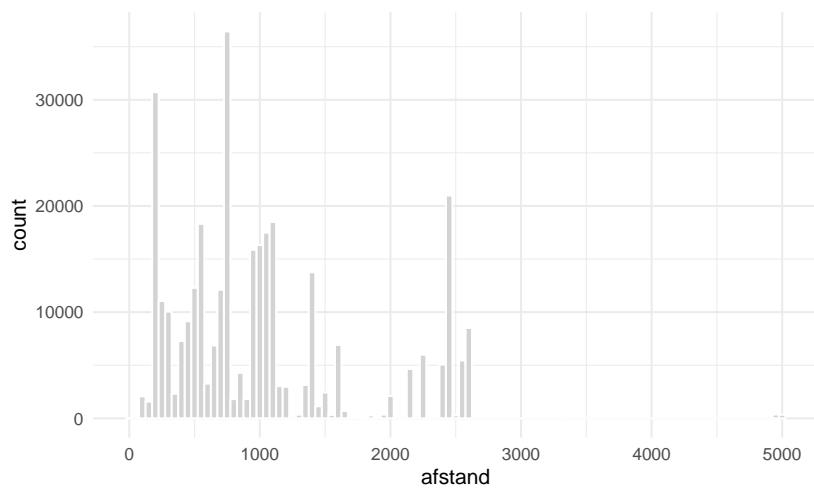


Figure 2.12: Histogram with binwidth 50

- De waarde van de y-as heeft geen directe betekenis.

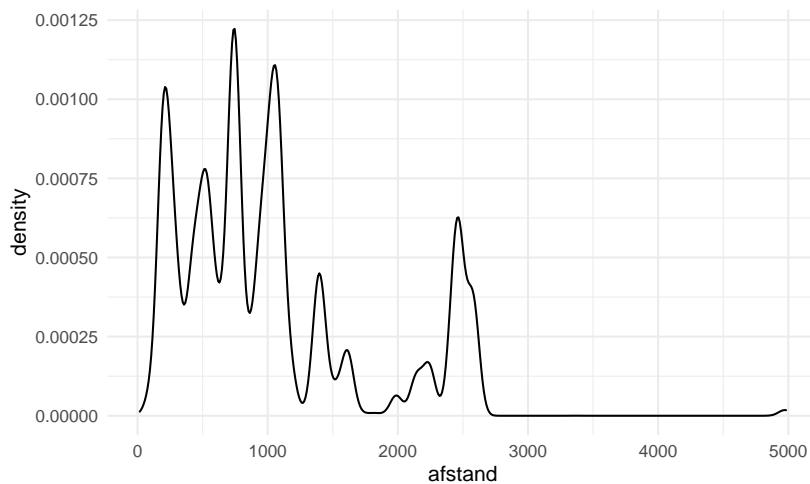


Figure 2.13: Density plot

Boxplot

- De lijn in het midden duidt de mediaan aan. Dit betekent dat 50% van je data onder deze lijn ligt, terwijl 50% er boven ligt. (Fig. 2.14)
- De box in het midden duidt de middelste 50% van je data aan. Dit wordt ook de interkwartiel-box genoemd. Dit betekent dat 25% van je data onder deze box zit en nog eens 25% boven deze box ligt. Hoe groter de box, des te meer de data gespreid is.
- Indien de box aan één zijde van de mediaanlijn groter is dan aan de andere zijde, dan wijst dit er op dat de data meer gespreid is aan die kant.
- De “whiskers” geven de laatste datapunten aan die als “normaal” beschouwd worden. Datapunten buiten deze grenzen beschouwt een boxplot als outliers of extreme waarden.
- De grens waar data van normaal naar extreem overgaat wordt door de boxplot bepaald door anderhalf keer de grootte van de interkwartiel-box op te tellen (en af te trekken) van de bovenste (onderste) grens van de interkwartiel-box. Punten die hier buiten liggen zijn outliers en worden als aparte punten aangeduid. De uitersten van de whiskers duiden de laatste datapunten aan binnen deze grenzen.
- Het is niet abnormaal dat er outliers in je data aanwezig zijn.
- Bij normaal verdeelde data zal je gemiddeld 7 outliers per 1000 datapunten mogen verwachten.

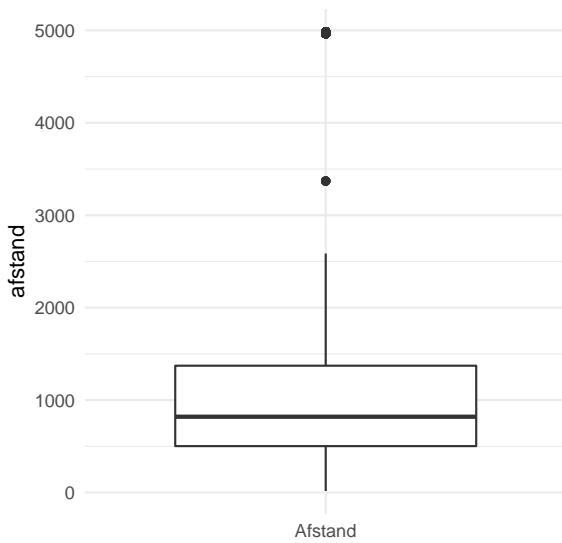


Figure 2.14: Verticale boxplot
vertrekvertraging

- Een normale verdeling is een bepaalde manier waarop data waarden verdeeld kunnen zijn die in de realiteit vaak voorkomt.
- Indien je echter veel meer outliers ziet op je boxplot visualisatie, dan is de kans reëel dat er meer aan de hand is:
 - Er zijn bijvoorbeeld systematische meetfouten
 - De objecten in je data zijn in feite op bepaalde aspecten significant verschillend waardoor je ze apart moet bestuderen.
- Je kan een boxplot ook roteren. (Fig 2.15)
- Boxplots komen beter tot hun recht bij bivariate analyses dan bij univariate analyses.



Figure 2.15: Horizontale boxplot
vertrekvertraging

Violin plot

- Een violin plot kan je beschouwen als een combinatie van een histogram en een boxplot. (Fig. @ref(fig:2_10a))
- Net als bij een boxplot wordt op verticale wijze getoond hoe de data verspreid is.
- Opnieuw kan je ervoor kiezen de grafiek te roteren. (Fig. @ref(fig:2_10b))

- Net als bij een histogram kan je goed zien waar het volume (de massa) van de data zich bevindt.
- Net als bij een histogram kan je detecteren hoeveel modi de data bezit.
- In tegenstelling tot de boxplot, kan je bij een violinplot wel niet duidelijk zien waar bijvoorbeeld het ‘midden’ van je data is.

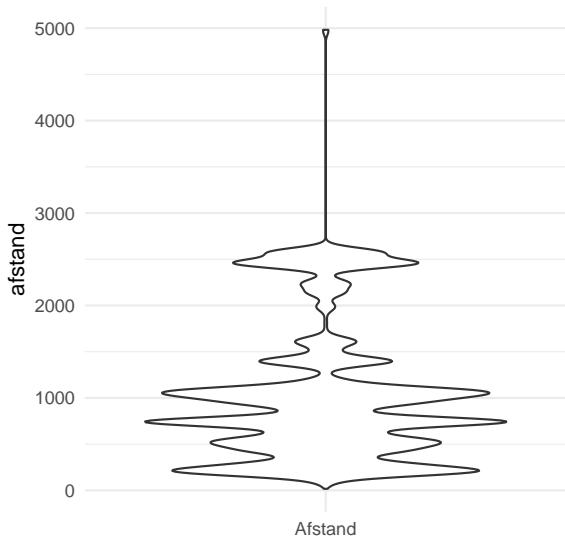


Figure 2.16: Verticale violin plot afstand

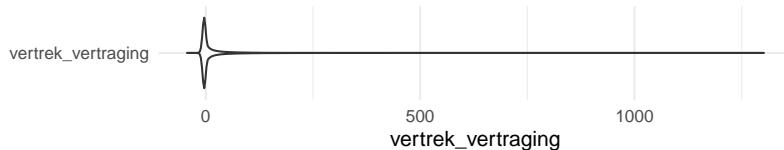


Figure 2.17: Horizontale violin plot vertrekvertraging

Jitter plot

- puntenwolk waarbij willekeurige “noise” (ruis) wordt toegevoegd.
- de ruis zorgt ervoor dat datapunten niet overlappen, en dat het duidelijk is waar de massa zich bevindt.
- Fig. 2.18 toont een vergelijking van violin, boxplot, point en jitter plot.

2.2 Bivariate visualisatie (2 variabelen)

- Wanneer we de relatie tussen 2 variabelen bekijken is het eenvoudig te denken in *oorzaak* en gevolg *termen*.¹

¹ Zie opmerking i.v.m. correlatie versus causaliteit, 2.5.

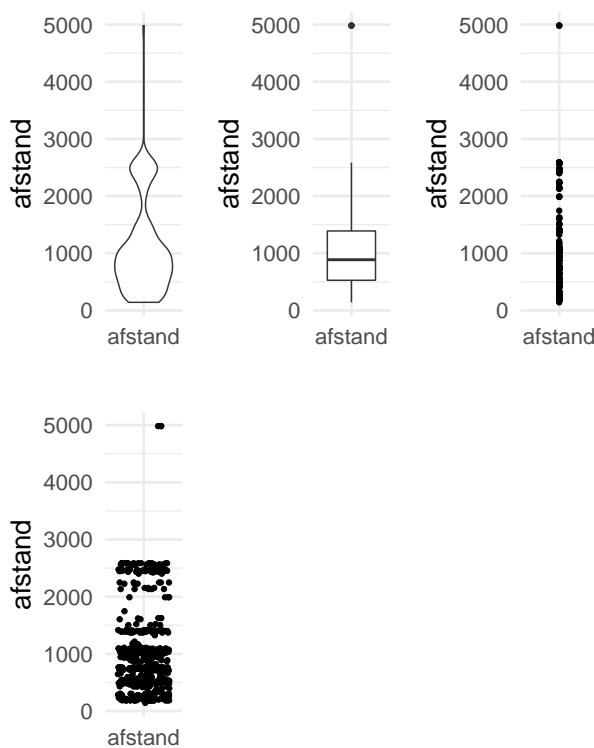


Figure 2.18: Violin, boxplot, point en jitter

- De variabele die we het label “oorzaak” geven, zullen we voortaan “onafhankelijke variabele” noemen.
- De variabele die we het label “gevolg” geven, zullen we voortaan “afhankelijke variabele” noemen.
- Waar we eigenlijk in geïnteresseerd zijn bij een visualisatie van 2 variabelen is de impact van de onafhankelijke variabele op de afhankelijke variabele weer te geven.
- Alle vragen die we kunnen stellen bij de visualisatie van één variabele, kunnen we nog steeds stellen, met telkens de bijkomende vraag of het waargenomen patroon verandert als de onafhankelijke variabele van waarde verandert.

2.2.1 Situatie 1: De onafhankelijke variabele is categorisch

Indien de afhankelijke variabele een continue variabele is kan je:

- meerdere boxplots op 1 grafiek visualiseren, met telkens 1 boxplot per waarde van de onafhankelijke variabele. (Fig. 2.19)
- meerdere violinplots op 1 grafiek tonen, met telkens 1 violinplot per waarde van de onafhankelijke variabele. (Fig. 2.20)

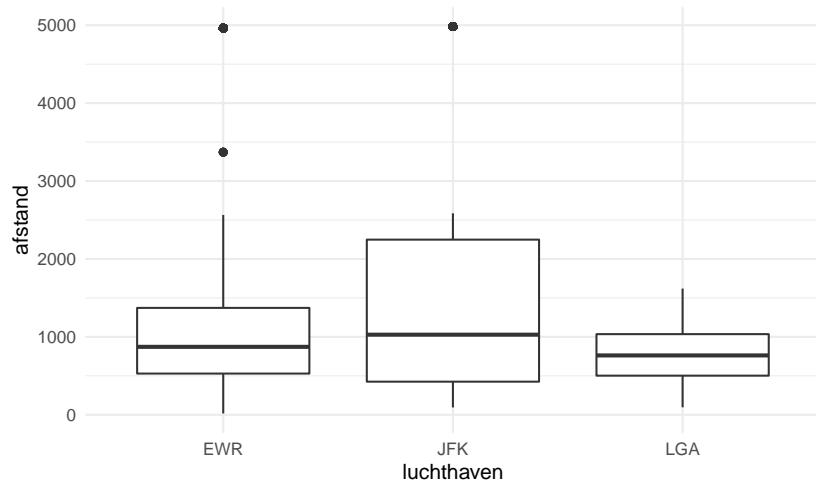


Figure 2.19: Bivariate boxplot

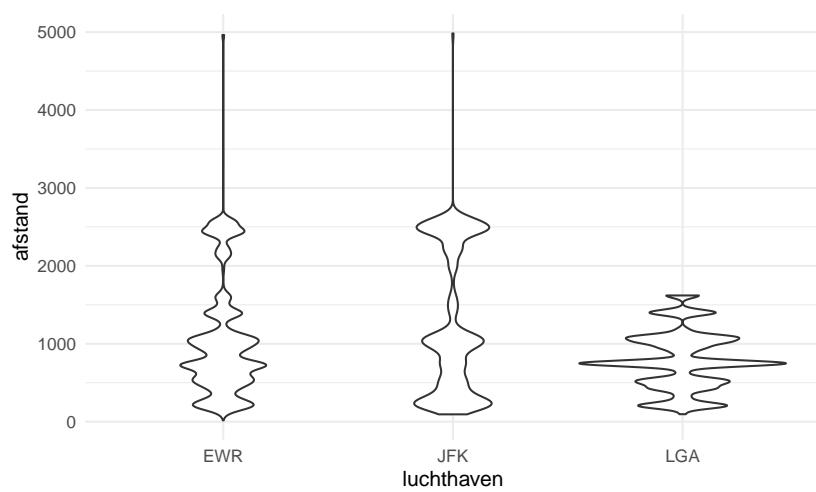


Figure 2.20: Bivariate violin plot

- meerdere histogrammen op 1 grafiek tonen
 - Hiervoor gebruiken we facetten: we tekenen voor elke waarde van de onafhankelijke variabele een apart assenstelsel. (Fig. 2.21)

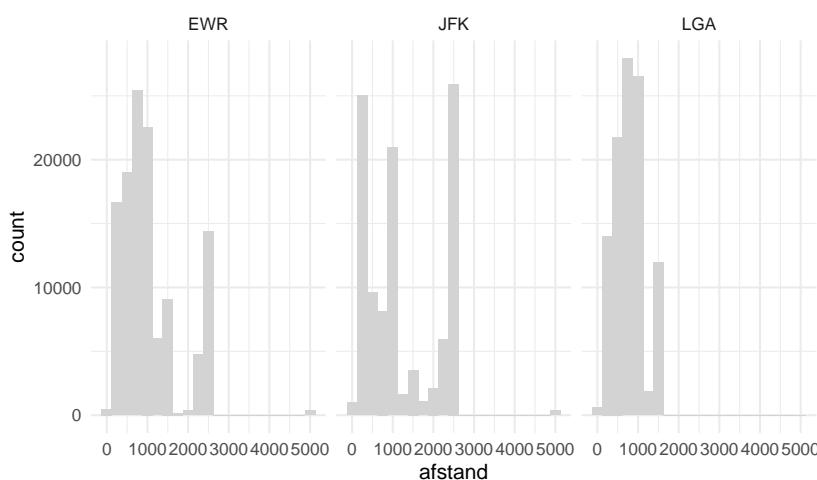


Figure 2.21: Bivariate histogram plot

- meerdere density plots
- Hiervoor kunnen we facetten gebruiken, ofwel de density plots over elkaar tekenen en onderscheiden met kleur. (Fig. 2.22-2.23)

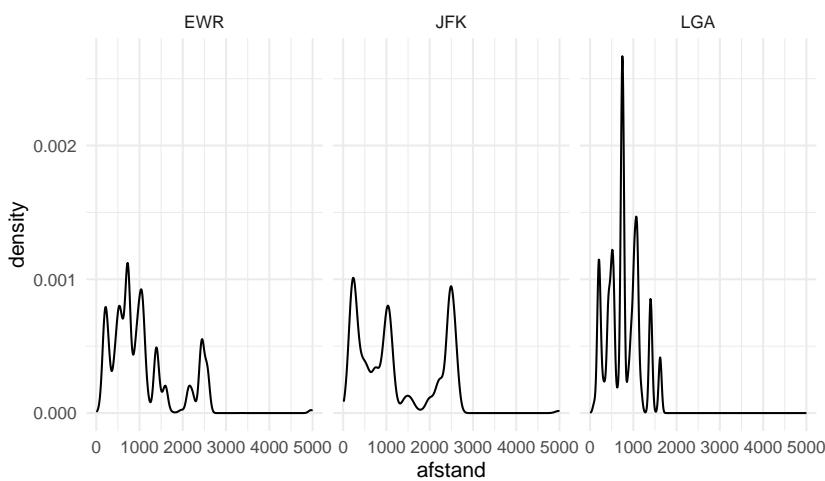


Figure 2.22: Bivariate density plot - apart

Indien de afhankelijke variabele een **categorische variabele** is:

- Kan je meerdere barplots op 1 grafiek visualiseren, met telkens de bars gegroepeerd per waarde van de onafhankelijke variabele.

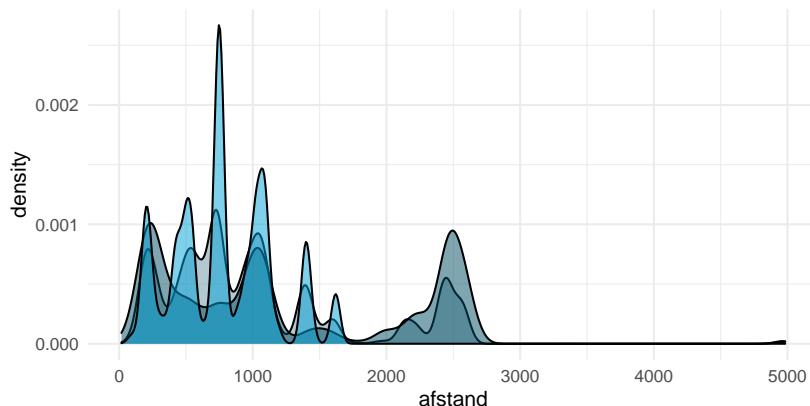


Figure 2.23: Bivariate density plot - overlappend

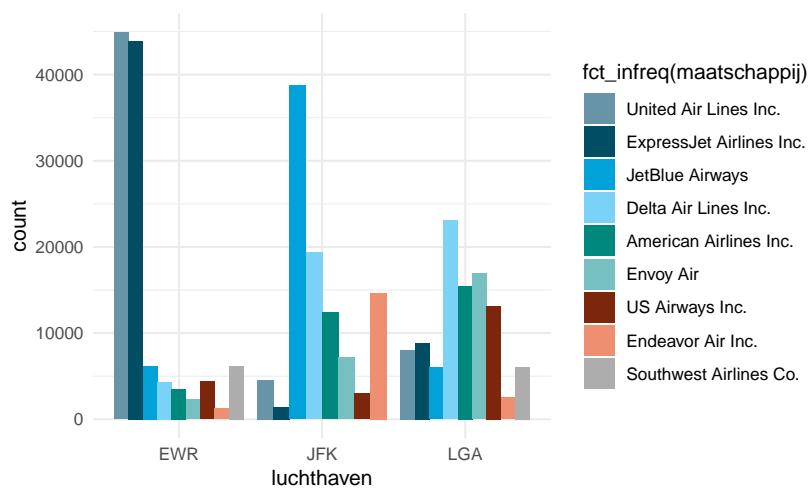
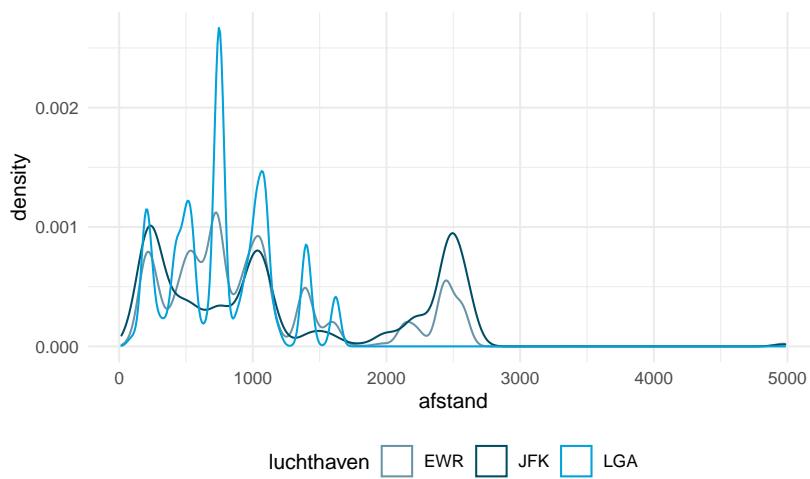


Figure 2.24: Bivariate barplot

- Kan je meerdere stacked barplots op 1 grafiek plaatsen, met telkens een volledige stack per waarde van de onafhankelijke variabele.

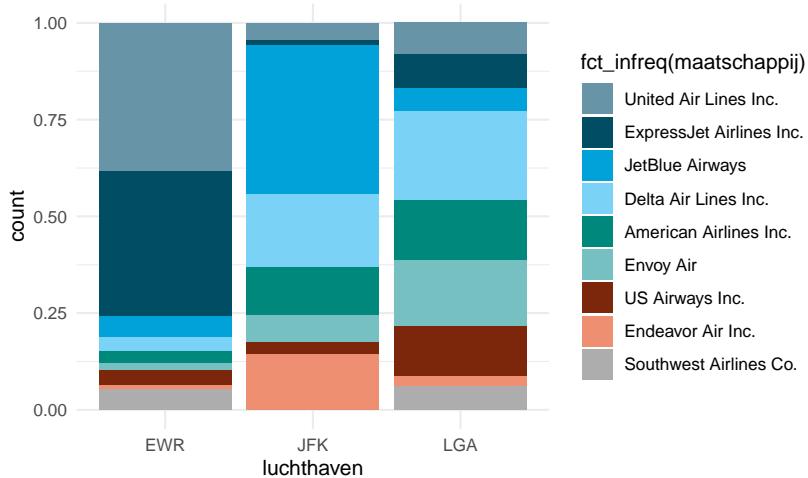
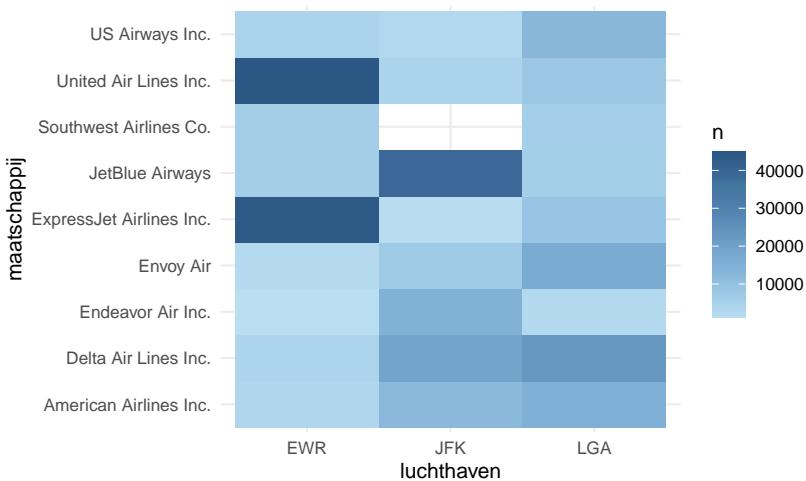


Figure 2.25: Bivariate stacked barplot

- Kan je een heatmap (of tile plot) gebruiken. Hier bij plaats je 2 categorische variabelen op de x-as en y-as, respectievelijk.
 - Voor elke combinatie van waarden is er een tegel die je kan inkleuren volgens de frequentie van de combinatie.



- Je kan bijkomende ook de exacte waarde in elke tegel plotten.



Let op wanneer beide variabelen categorisch zijn, is het nog steeds van belang welke je beschouwd als afhankelijke en welke als onafhankelijke. Technisch kan je ze omdraaien, maar de betekenis van je visualisatie is niet dezelfde!

Andere mogelijkheden:

- treemap (Fig. 2.27)
- mosaic plot (Fig. 2.28)

2.2.2 Situatie 2: De onafhankelijke variabele is continu

In dit geval kan je geen aparte plot per mogelijke waarde van de onafhankelijke variabele maken omdat er mogelijk oneindig veel waarden zijn.

Indien de afhankelijke variabele continu is, dan kan je een scatterplot maken.

- Iedere observatie is een punt in je grafiek, waarbij de x-waarde op de grafiek overeenkomt met de waarde van de onafhankelijke variabele en de y-waarde op de grafiek overeenkomt met de waarde van de afhankelijke variabele.
- Om patronen beter te herkennen kan je een “trend-lijn” toevoegen.
- Bij scatterplots is er gevaar voor overplotting
- Mogelijke oplossingen
 - 2D histogram: verdeel veld op in vierkante bins en tel per bin hoeveel data punten er zijn

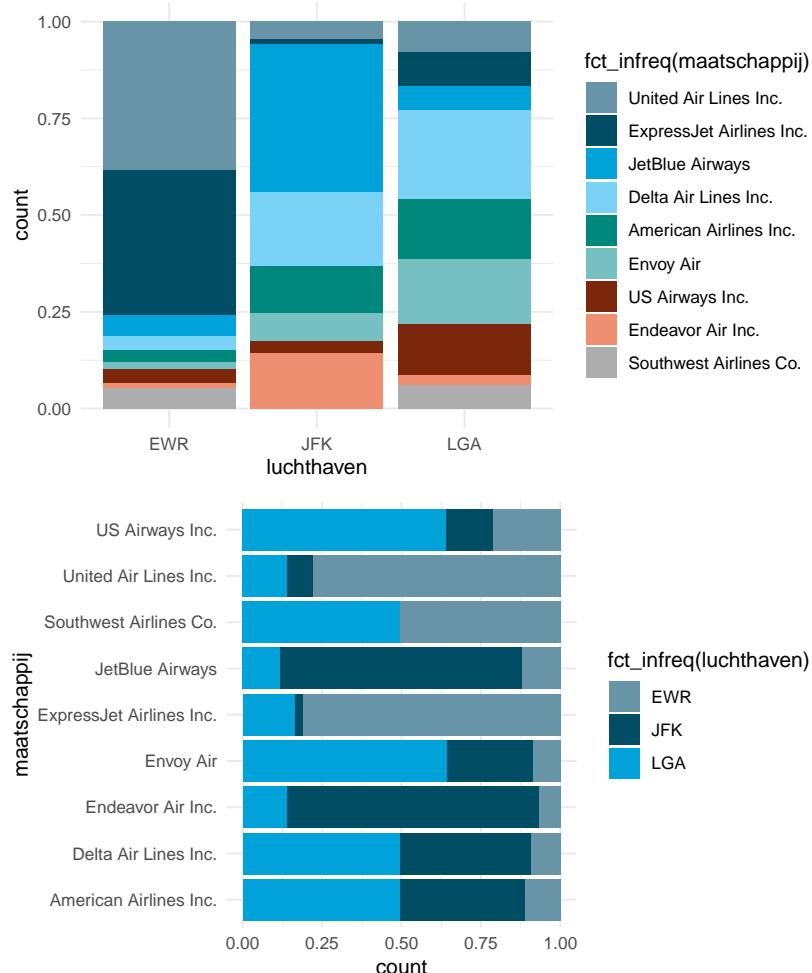


Figure 2.26: Twee verschillende stacked barcharts van luchthaven en maatschappij.

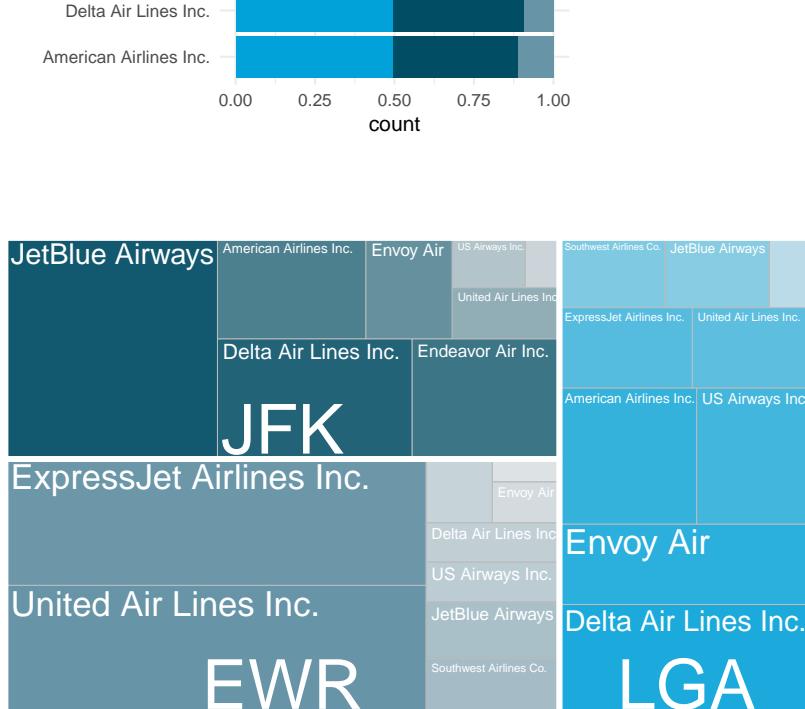


Figure 2.27: Treemap luchthaven en maatschappij.

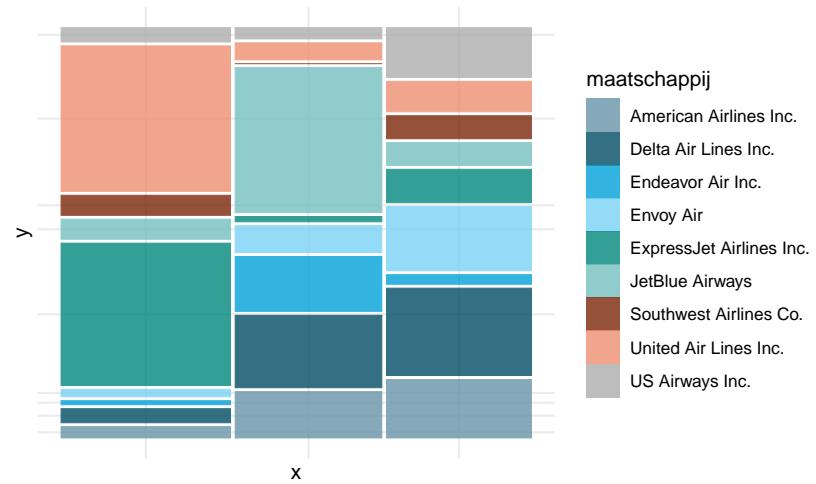


Figure 2.28: Mosaic plot luchthaven en maatschappij.

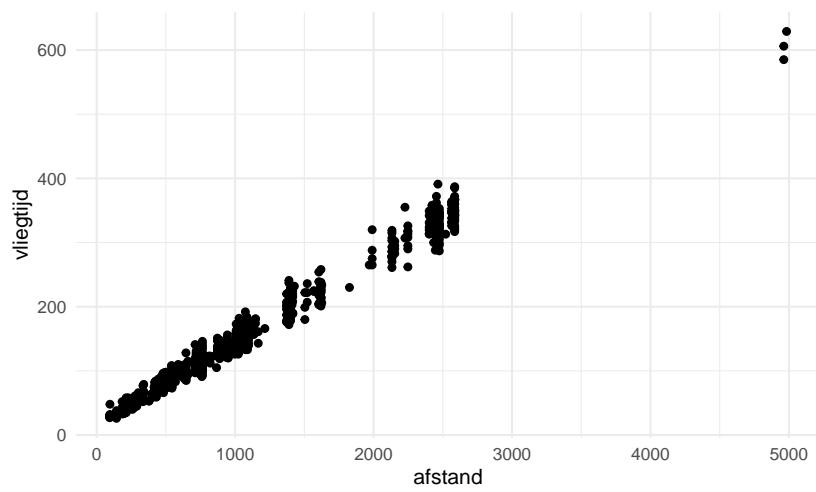


Figure 2.29: Scatterplot

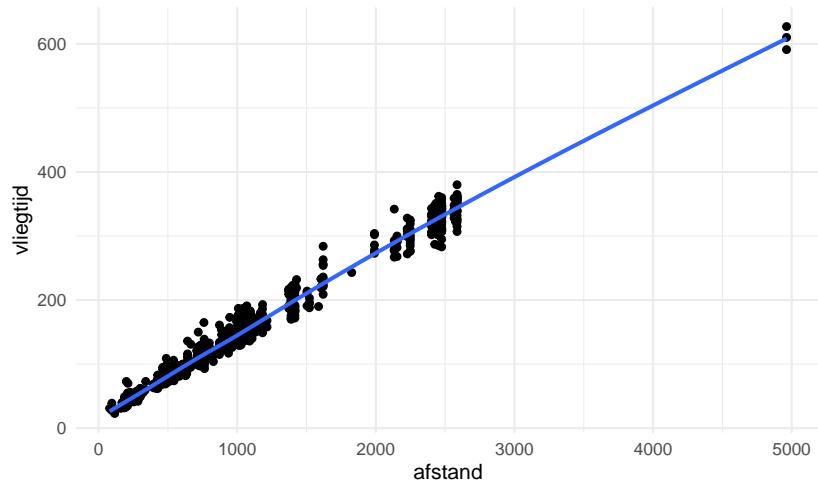


Figure 2.30: Scatterplot met trendlijn

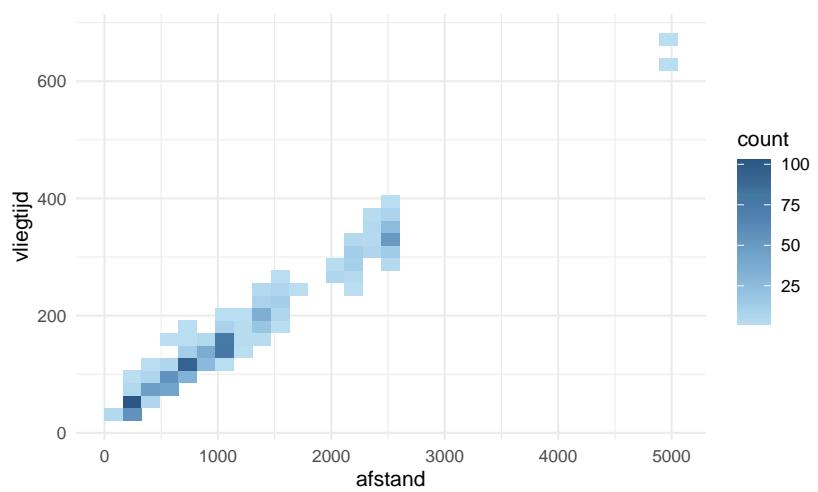


Figure 2.31: Scatterplot met trendlijn

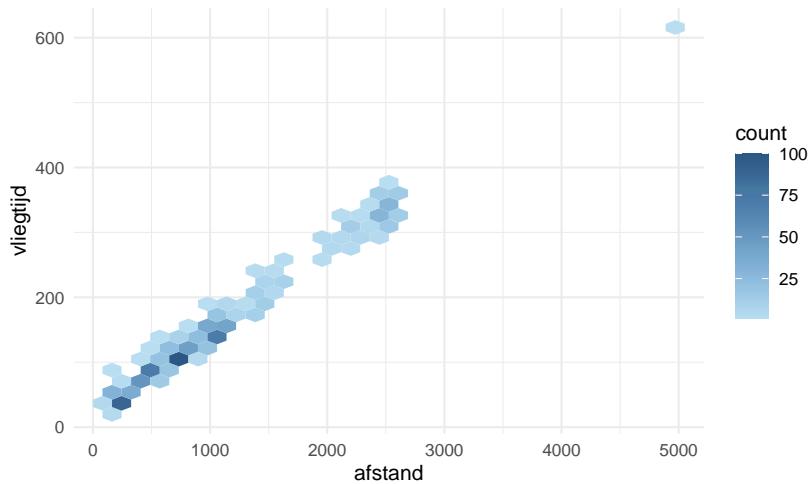


Figure 2.32: Hexplot met trendlijn

- Hexplot: analoog, maar gebruik zeshoekige bins ipv vierkanten. Voordeel: punten binnen elke zeshoek liggen dichter bij het middelpunt van de bin.

Indien de afhankelijke variabele categorisch is, dan kan je niet rechtstreeks een betekenisvolle plot maken omdat er waarschijnlijk te weinig datapunten zijn voor iedere mogelijke waarde van de onafhankelijke variabele.

- Wat je dan best kan doen, is de onafhankelijke continue variabele categorisch maken door deze in te delen in bins/intervallen. En dan ben je terug in de situatie waarbij de onafhankelijke variabele categorisch is. We komen hierop terug in het hoofdstuk over Data Voorbereiding.

2.2.3 Situatie 3: De onafhankelijke variabele is tijd

- Tijd kunnen we zien als continue variabele
 - Bijgevolg zelfde grafieken mogelijk als wanneer onafhankelijke variabele continue is
 - * Tijd + continue afhankelijk -> scatterplot, 2D histograms, hex bins
 - * Tijd + categorisch afhankelijk -> probleem: tijd categoriseren (zie verder).
- Wanneer we één enkele variabele voorstellen doorheen de tijd is er per tijdseenheid maar 1 data punt. Hieronder wordt de gemiddelde vertrekvertraging per dag getoond.

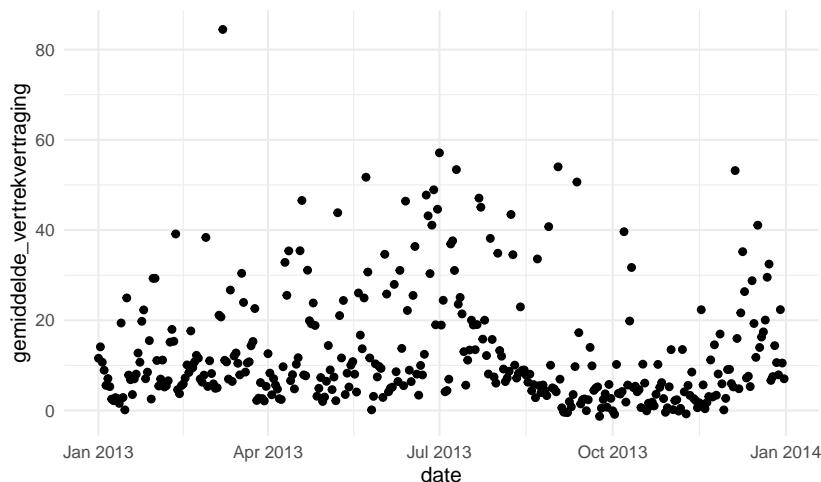


Figure 2.33: Puntenwolk met tijd op x-as

In dat geval is het beter om in plaats van punten een lijngrafiek te gebruiken.

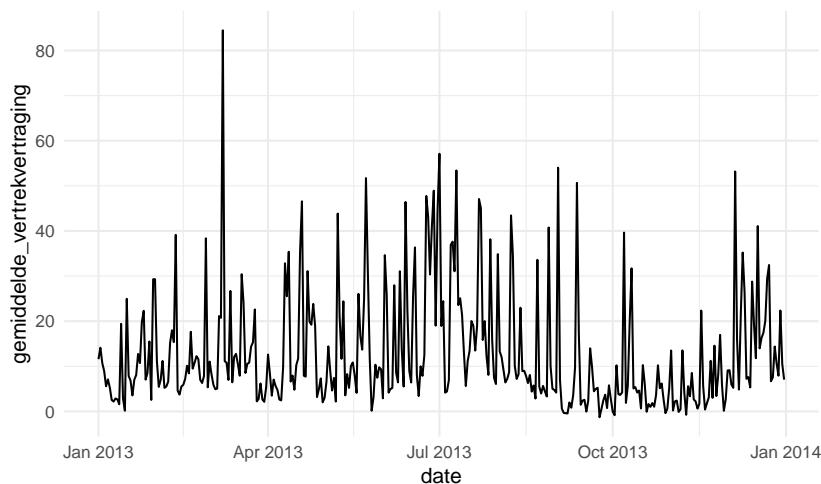


Figure 2.34: Lijngrafiek

Indien je een beperkt aantal punten hebt (hieronder bijvoorbeeld één maand van de vluchtgegevens) kan je ervoor kiezen om zowel punten als lijnen te tonen. Op die manier is het makkelijker individuele data punten af te lezen.

Indien we veel datapunten hebben, wat hier het geval is, kan een lijngrafiek zeer chaotisch worden. We kunnen daarom ervoor kiezen om onze tijd in te delen in categorieën. Bijvoorbeeld, in plaats van de dagelijkse gemiddelde vertrekvertraging, kunnen we de gemiddelde vertrekvertraging per maand berekenen en tonen.

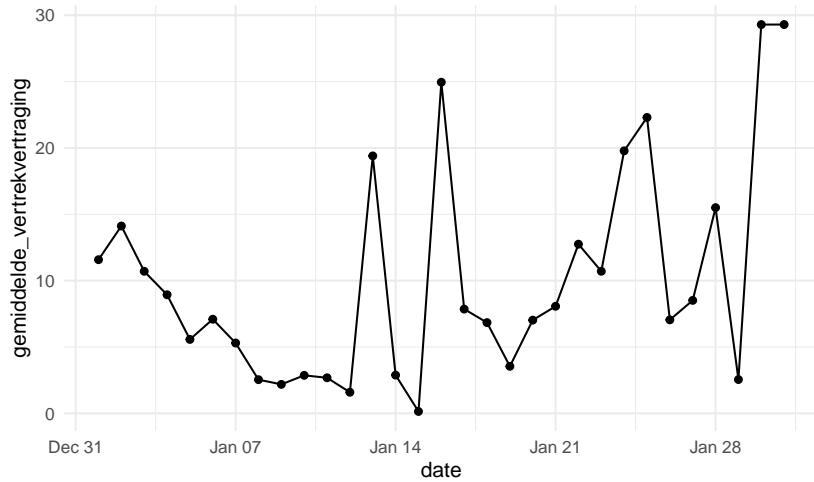


Figure 2.35: Lijn grafiek met punten

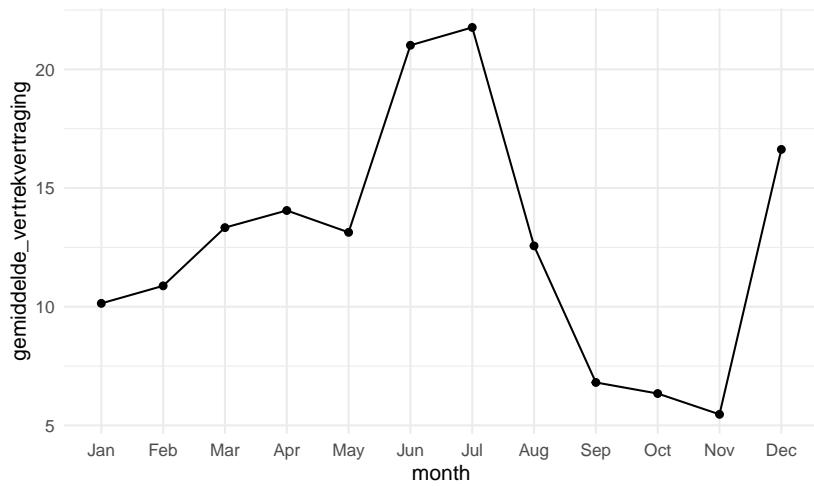


Figure 2.36: Lijngrafiek van gemiddelde vertrekvertraging per maand.

- Op dit moment verliezen we daardoor wel veel informatie. Maar we kunnen dit nu ook beschouwen als een visualisatie van een categorische variabele (maand) t.o.v. een continue. Waardoor we de technieken voor dit type bivariate visualizaties kunnen toepassen. Bijvoorbeeld boxplots. We zien nu zowel de algemene trend als outliers. In februari was er bijvoorbeeld een dag waar de gemiddelde vertrekkering ver boven de normale trend lag.

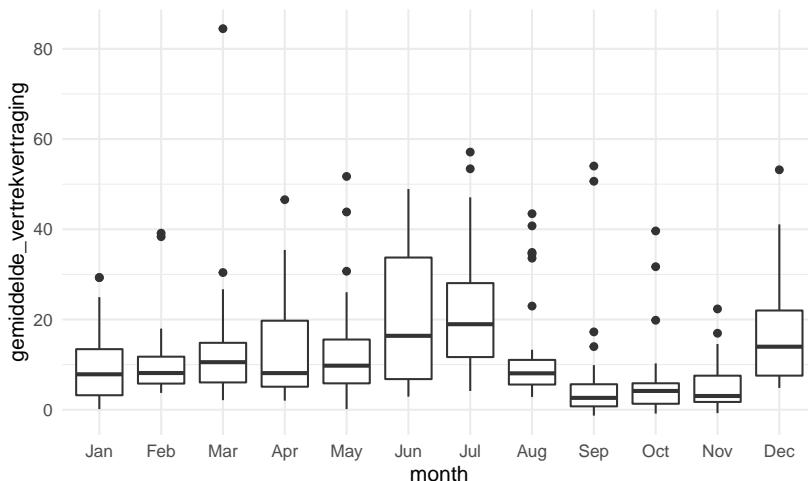


Figure 2.37: Boxplots van gemiddelde dagelijkske vertrekvertraging voor elke maand.

- Wanneer we de tijd gecategoriseerd hebben kunnen we ook categorische variabelen weergeven als afhankelijke. Bijvoorbeeld, zijn er verschillen in het aantal vluchten per maatschappij doorheen de tijd. We kunnen hier dezelfde types grafieken als voor bivariate cat+cat visualizaties gebruiken, bijvoorbeeld stacked barcharts.
- We kunnen categorizeren op maand, jaar, etc. Maar ook op tijdspecifiekere kenmerken, zoals bijvoorbeeld de dag van de week
- Of het uur van de dag

2.3 Multivariate visualisaties (meer dan 2 variabelen)

- Datavisualisatie van patronen tussen meer dan 2 variabelen worden snel te complex om te interpreteren.
- Het basisprincipe is wel eenvoudig.
 - Je hebt typisch 1 afhankelijke variabele (Y) en een aantal onafhankelijke variabelen (A, B, ...).

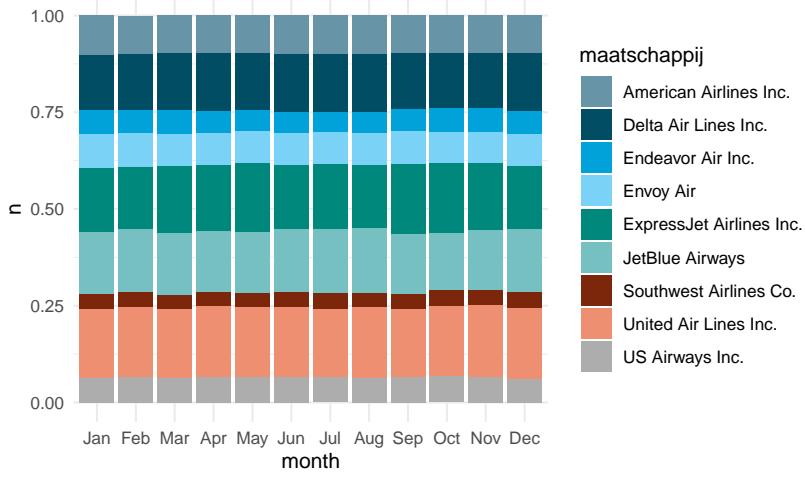


Figure 2.38: Verdeling van aantal vluchten over maatschappijen per maand.

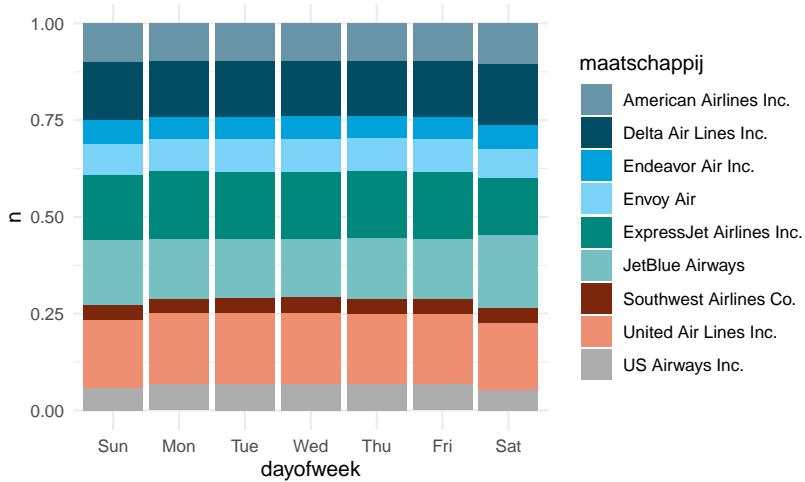


Figure 2.39: Verdeling van aantal vluchten over maatschappijen per dag van de week.

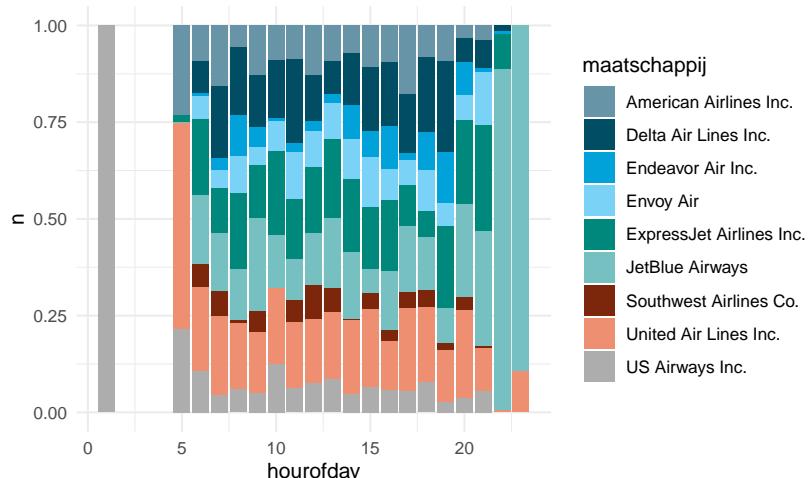


Figure 2.40: Verdeling van aantal vluchten over maatschappijen per vertrekuur.

- Je visualizeert eerst Y en A (bivariaat)
- Je voegt dan de volgende variabelen (B, c, ...) stap voor stap toe aan de grafiek.
 - * Door de bivariate grafiek te herhalen in verschillende facetten (een voor elke waarde van B).
 - * Door verschillende kleuren te gebruiken voor elke waarde van B
- Bij multivariate visualisaties zijn er afhankelijk van de data types oneindig veel mogelijke grafieken die je kan maken.
 - Het is vaak afhankelijk van de data welke grafiek het “best past”
 - Enkel wanneer de onafhankelijk variabele continu is zijn de keuzes beperkt en ben je vaak genoodzaakt om deze om te zetten naar categoriën.

2.3.1 Voorbeeld: In welke mate hangt de vertrek vertraging af van de luchthaven en de afstand?

Stap 1. Vertraging vs. afstand

- Beide continue: scatterplot

Stap 2. Voeg invloed van luchthaven toe.

- Optie 1: gebruik kleur om de verschillende luchthavens te differenteren. Een trendlijn kan hier helpen.
- Geen geweldige resultaat in dit geval.
- Optie 2: Gebruik facetten voor de verschillende luchthavens.

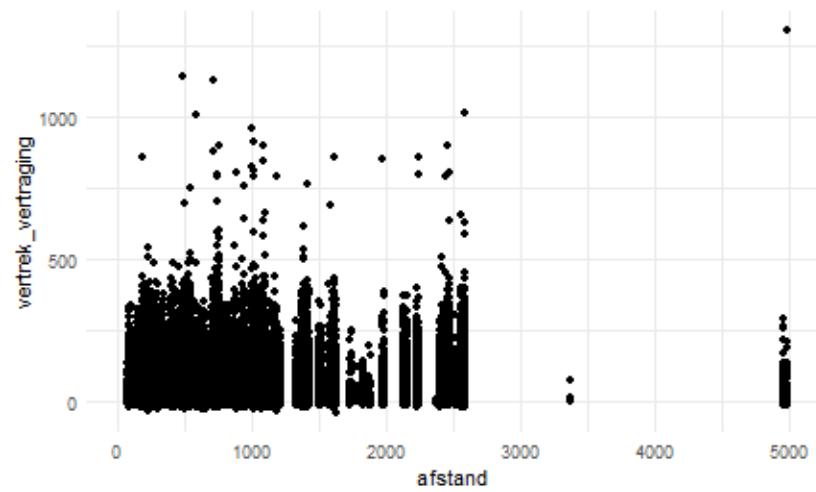


Figure 2.41: Vertrekvertraging vs afstand

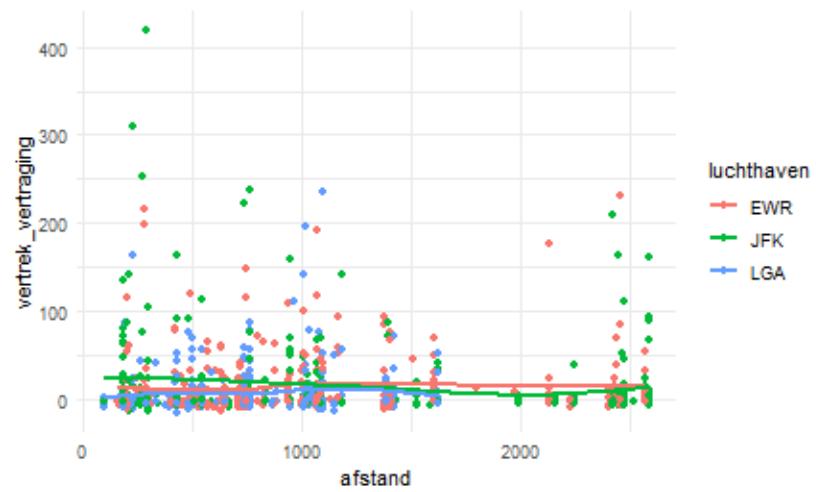


Figure 2.42: Vertrekvertraging vs afstand en luchthaven

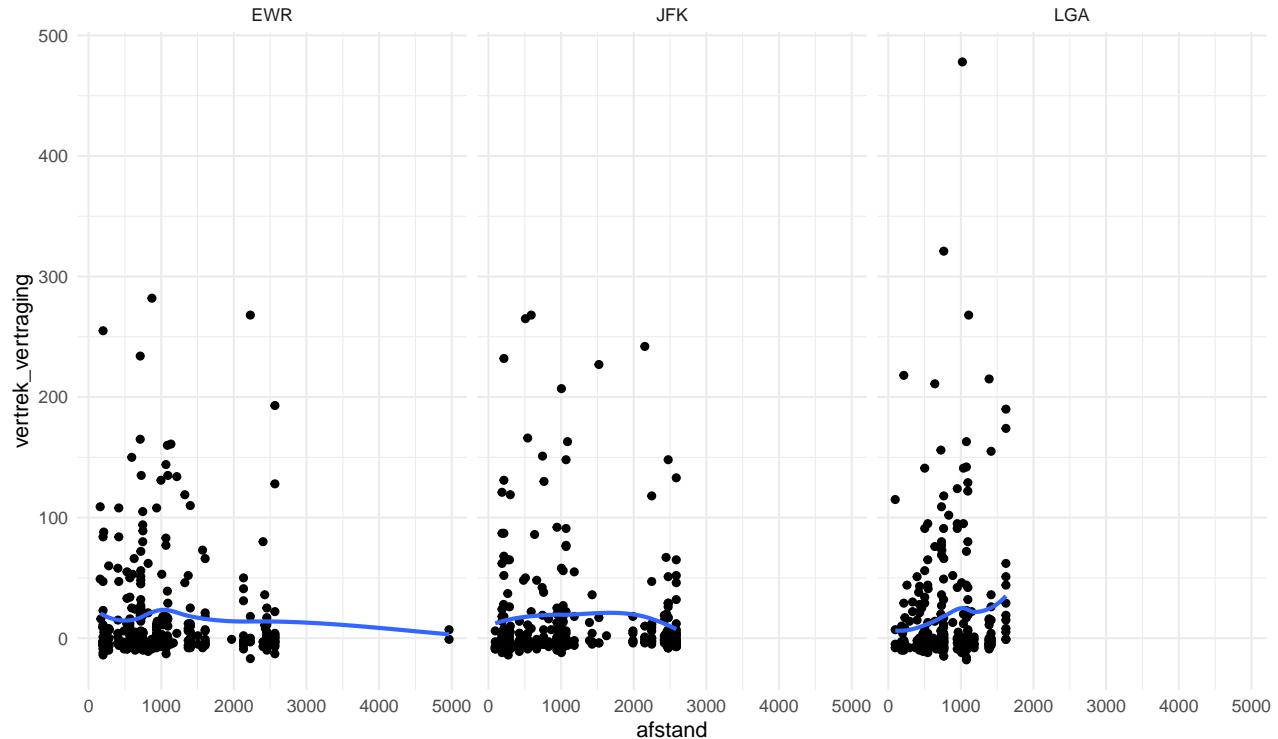


Figure 2.43: Vertrekvertraging vs afstand en luchthaven

- Optie 3: Facets, maar gebruik hex bins

2.3.2 Voorbeeld: multivariaat tijd

Situatie 1: Variabelen hebben dezelfde eenheid.

Voorbeeld: vertrekvertraging en aankomstvertraging. Je kan lijngrafieken tekenen met meerdere lijnen op hetzelfde assenstelsel.

- Of je kan er voor kiezen elke lijn in een afzonderlijk paneel te tonen

Situatie 2: Variabelen hebben niet dezelfde eenheid

Voorbeeld: de gemiddelde levensverwachting en gdp per capita doorheen de tijd. In dit geval ben je genoodzaakt 2 panelen te gebruiken.

Optie 2: Maak een connected scatterplot. Toon een punt voor elke meting, waarbij x en y elk een variabele voorstellen. Verbindt dat elk punt in chronologische volgorde.

Variant, per continent:

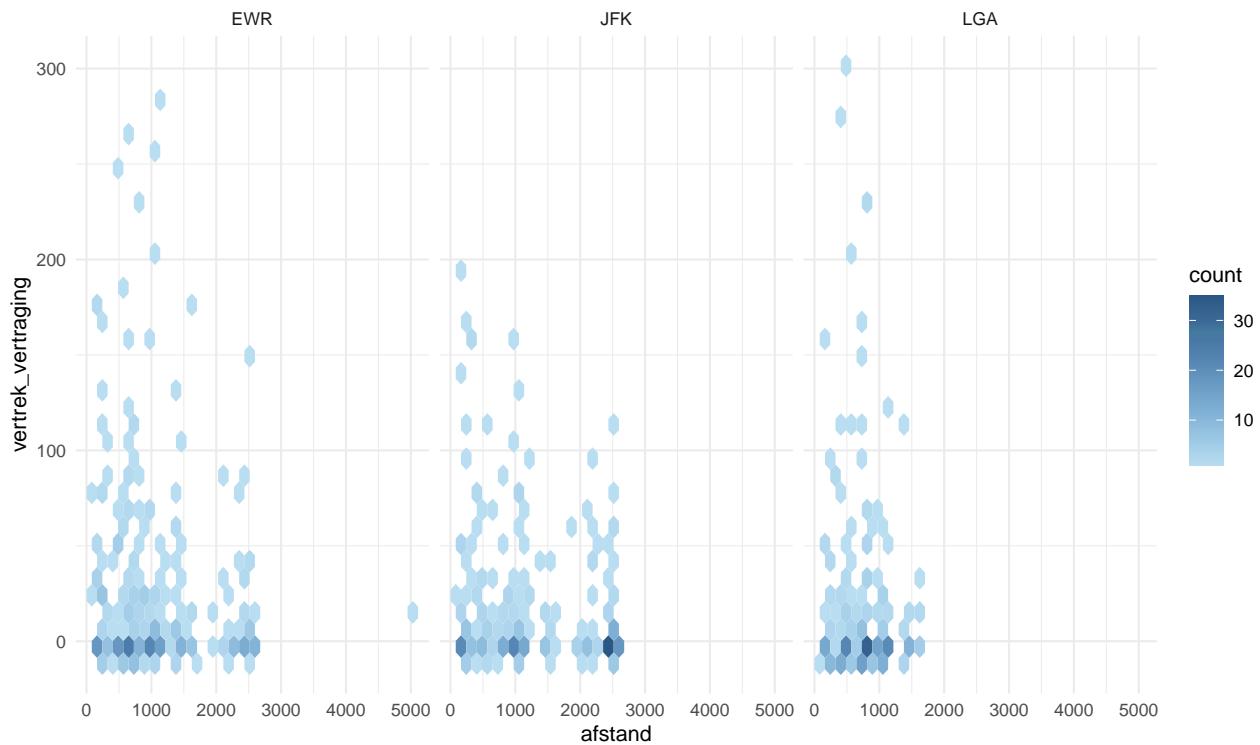


Figure 2.44: Vertrekvertraging vs afstand en luchthaven, hexbins

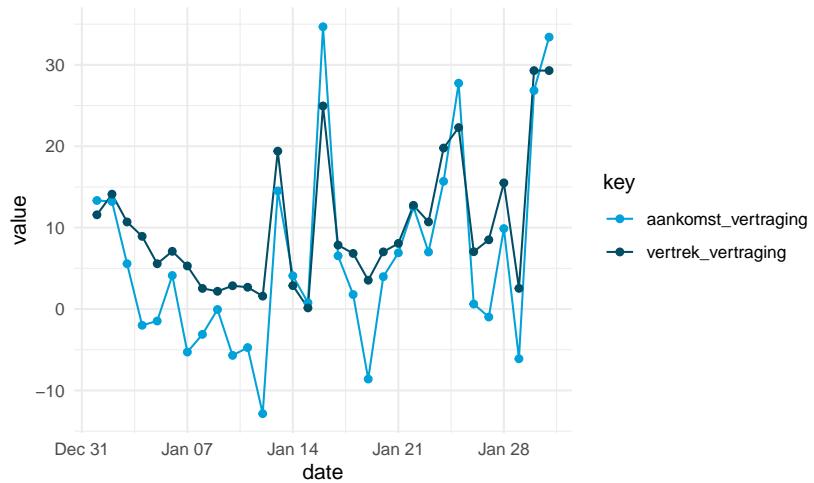


Figure 2.45: Evolutie van 2 variabelen over tijd in één grafiek (zelfde meeteenheid)

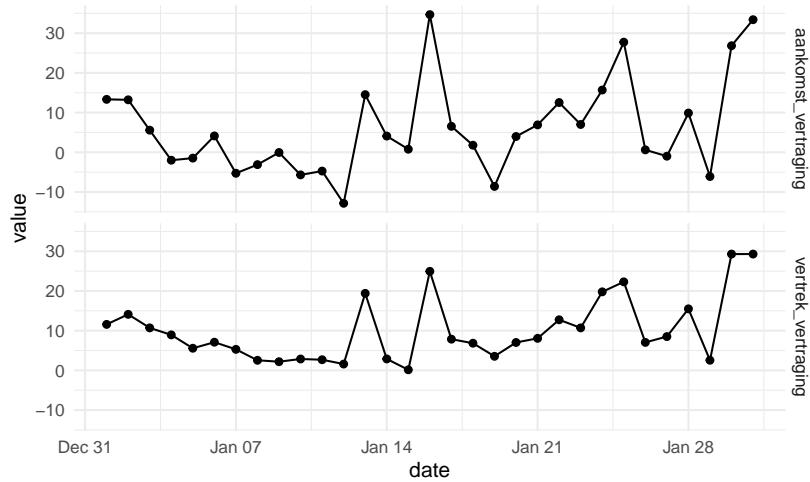


Figure 2.46: Evolutie van 2 variabelen over tijd in afzondelijke panels.

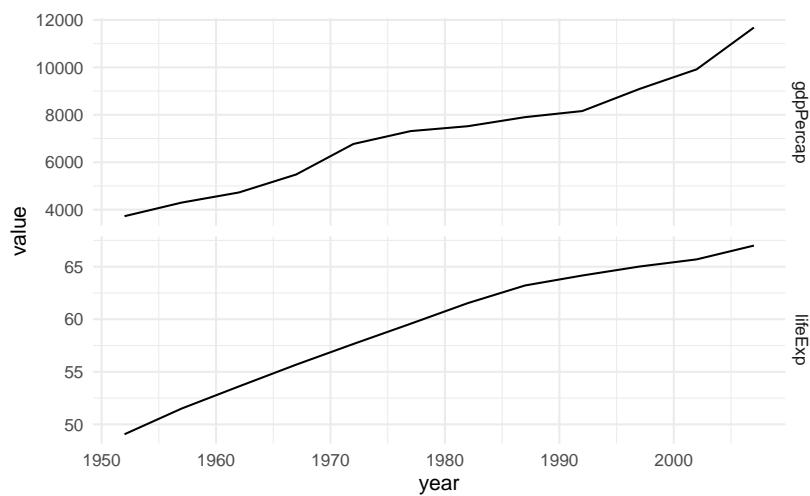


Figure 2.47: Evolutie van 2 variabelen met andere eenheden in afzonderlijke panels.

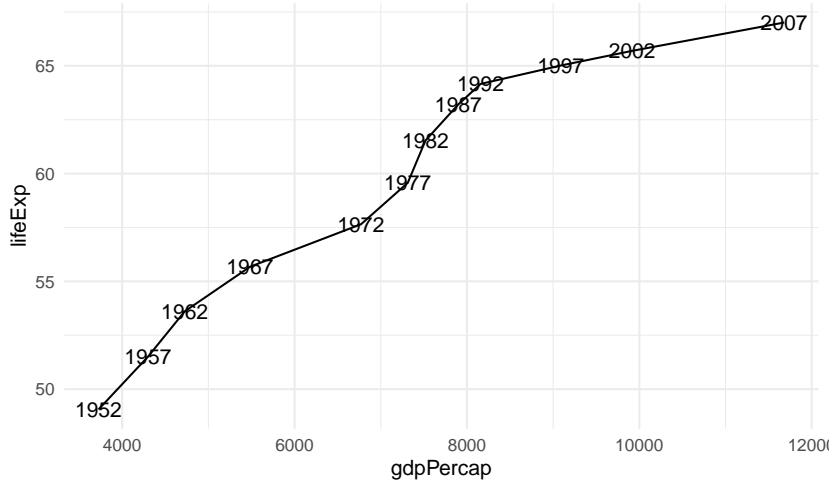


Figure 2.48: Evolutie van 2 variabelen (levensverwachting en inkomen per capita) aan de hand van connected scatterplot.

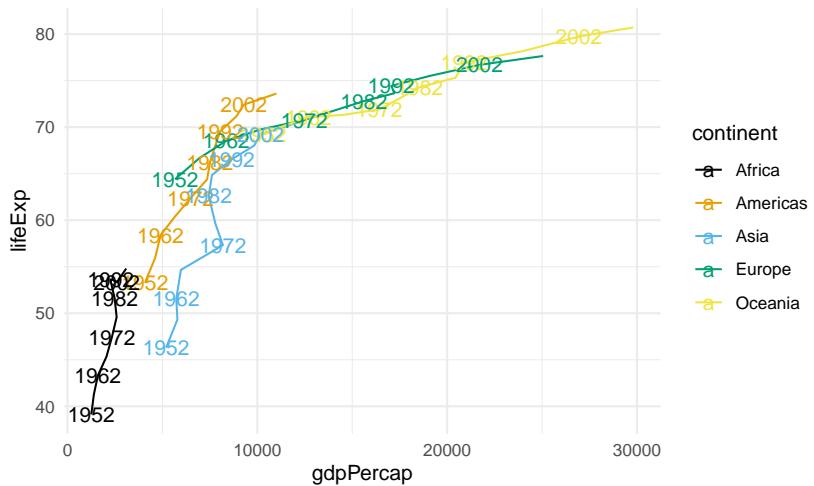


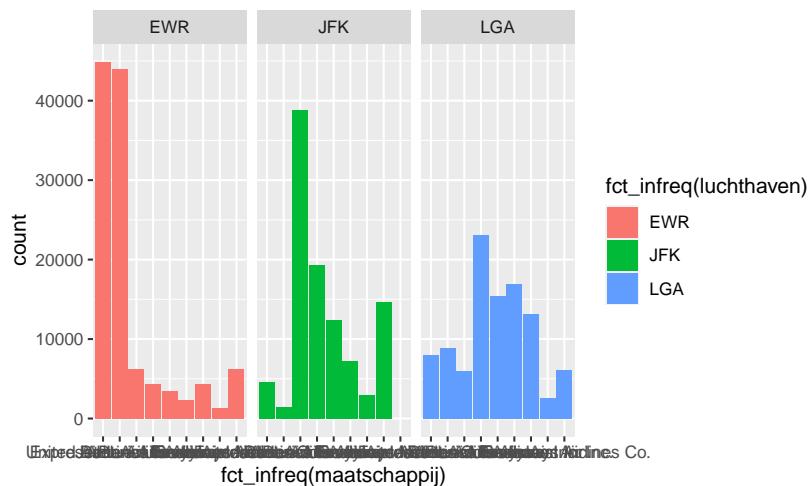
Figure 2.49: Evolutie van 2 variabelen aan de hand van connected scatterplot - verschillende groepen.

2.4 Visualisaties voor communicatie

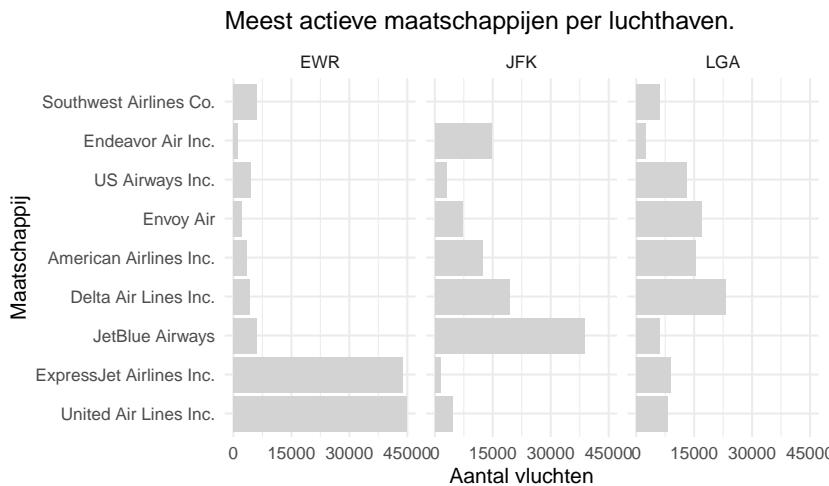
Wanneer uiteindelijk beslist om een visualisatie te gebruiken om te communiceren, zorg ervoor dat

- de grafiek leesbaar is
- je kleur enkel gebruikt waar nodig.
- je correcte axis-labels gebruikt
- je geen thema gebruikt dat te druk/overheersend is
- je een gepaste titel voorziet.

2.4.1 Voorbeeld: voor ~ goed voor exploratie



2.4.2 Voorbeeld: na ~ goed voor communicatie



Merk op: ver van alle grafieken getoond in dit hoofdstuk zijn goed voor communicatie zonder aanpassingen.

2.5 How charts lie

2.5.1 Causaliteit vs correlatie

- Van zodra er twee (of meer) variabelen zijn, gaan we op zoek naar patronen in relaties tussen de variabelen.
- Het is belangrijk en essentieel te beseffen dat mensen een automatische reflex hebben om te denken in termen van oorzaak-gevolg als we kijken naar relaties tussen twee variabelen.
 - Het is echter niet omdat er een duidelijke relatie bestaat tussen twee variabelen (correlatie), dat hier sprake is van een oorzaak-gevolg verband (causaliteit).
 - Bijvoorbeeld: Indien in de zomer de verkoop van paraplu's sterk stijgt, dan zal de graanopbrengst in het najaar dalen. Dit betekent niet dat de verkoop van paraplu's een impact heeft op de graanopbrengst. Wat hier waarschijnlijk gebeurt, is dat door hevige regenval in de zomermaanden, de verkoop van paraplu's is toegenomen en de graanoogst tegenvalt.
 - * Soms is het intuïtief zeer onwaarschijnlijk dat de waargenomen correlatie causaliteit impliceert. Kijk hiervoor maar eens naar de voorbeelden op <http://www.tylervigen.com/spurious-correlations>
 - * Wanneer het echter plausibel is dat de waargenomen correlatie causaliteit voorstelt, is het belangrijk dat we tegen onze natuurlijke reflex in gaan en niet in termen van oorzaak-gevolg denken.
 - * Het aantonen van causaliteit is nooit mogelijk met descriptieve en exploratieve data analyse!

2.6 Referenties

- Information is Beautiful
- Fundamentals of Data Visualization
- R Graph Gallery
- Data to viz
- Spurious correlations
- Misleading election map

3

[Tutorial] Data visualisatie

3.1 Voor je begint

Voordat je met deze zelfstudie begint, moet je eerst het pakket `ggplot2` installeren, als je dat nog niet gedaan hebt. Je kunt dit doen met de volgende regel code:

```
install.packages("ggplot2")
```

In ieder geval, moet je het pakket in je sessie laden.

```
library(ggplot2)
```

Je hebt ook twee datasets nodig, `movies` en `diamonds`. Beide worden als .RDS bestand bij deze tutorial geleverd.

```
movies <- readRDS("movies.RDS")
diamonds <- readRDS("diamonds.RDS")
```

3.2 Introductie

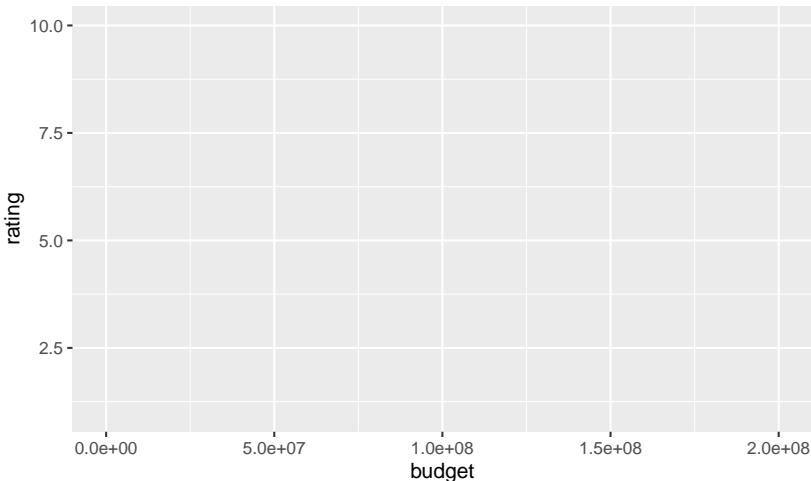
Het maken van een plot met `ggplot2` begint met de `ggplot()` functie. De `ggplot` functie heeft twee belangrijke argumenten

- **data**: dit definieert de dataset die voor de plot moet worden gebruikt. Dit moet een `data.frame` zijn.
- **mapping**: de mapping zal bepalen hoe de variabelen worden *mapped* op de esthetica¹ van de plot, zoals verderop zal worden uitgelegd. Deze mapping moet altijd worden gemaakt met de `aes()` functie .

We willen bijvoorbeeld een scatterplot maken van de films, waarbij we de x-as gebruiken voor hun budget en de y-as voor hun waardering. We roepen `ggplot` dan als volgt aan:

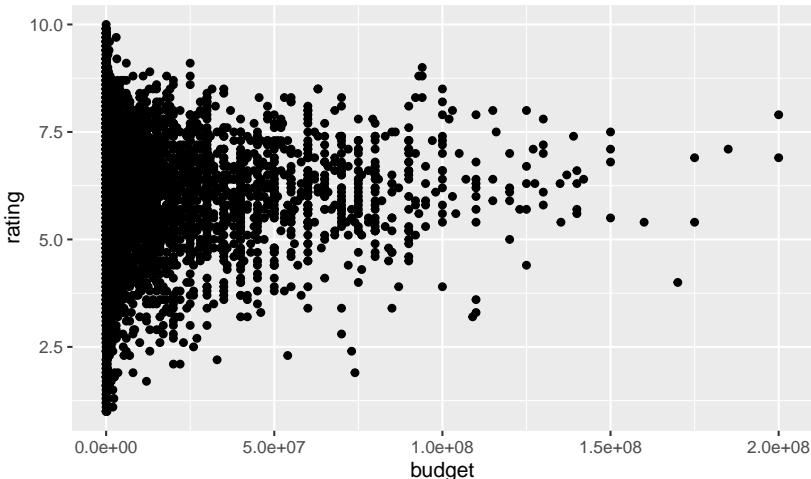
```
ggplot(data = movies, mapping = aes(x = budget, y = rating))
```

¹ Hoewel het bepaalde analyses interessanter maakt als je bekend bent met de betekenis van de verschillende variabelen, is geen specifieke kennis van auto's vereist om deze tutorial te voltooien.



Zoals je ziet, creëert deze regel code een plot met de assen zoals gedefinieerd. Er worden echter geen gegevens gevisualiseerd. De reden hiervoor is dat ggplot nog niet weet hoe we het willen visualiseren. We moeten wat genoemd wordt *een geometrische layer* toevoegen. Om een scatterplot te maken, die uit *punten* bestaat, voegen we `geom_point` aan de plot toe.

```
ggplot(data = movies, mapping = aes(x = budget, y = rating)) +
  geom_point()
```

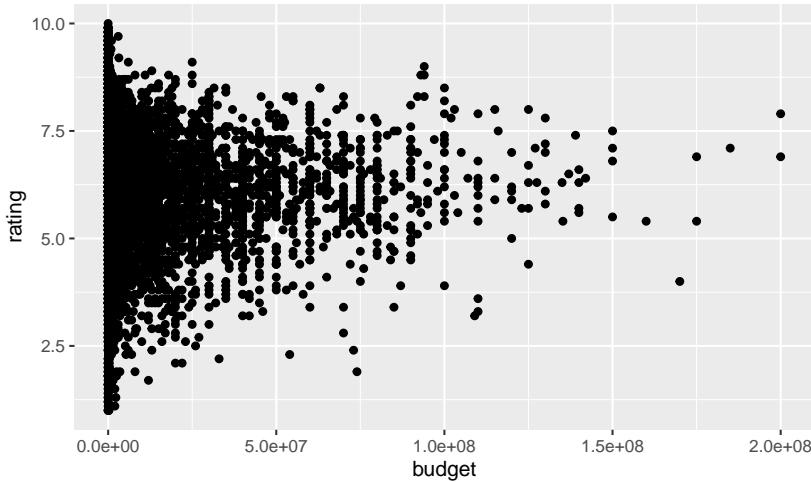


Dit lijkt er al meer op. Merk op dat de geometrische laag aan de plot is toegevoegd door gebruik te maken van het `+` symbool. Op deze manier kunnen meerdere lagen aan dezelfde plot worden toegevoegd, evenals titels, labels, en configuraties van de lay-out, zoals we verderop zullen zien.

Merk op dat we de mapping van de *aesthetics* ook in de geometrische laag zelf kunnen plaatsen. Dit maakt onze code voorlopig iets leesbaarder, omdat we de geometrische layer en de mapping ervan op

dezelfde regel plaatsen.

```
ggplot(data = movies) +
  geom_point(mapping = aes(x = budget, y = rating))
```



We hebben nu onze allereerste plot gemaakt! In de volgende secties zullen we leren hoe we verschillende geom-layers en *aesthetics* kunnen gebruiken en hoe we de lay-out van onze grafieken kunnen verbeteren.

3.3 Verschillende geometries

Naast `geom_point` bestaan er nog veel meer verschillende geometrieën om gegevens in ggplot te plotten. Je kunt ze bekijken door ‘`geom_`’ in het console in te typen en door de auto-complete-lijst te navigeren. Elk van de geom-layers komt met zijn eigen specifieke set van *aesthetics* die in kaart gebracht kan (en soms moet) worden. In deze tutorial zullen we ons vooral richten op de volgende geometrische lagen:

- `geom_point`
- `geom_histogram`
- `geom_boxplot`
- `geom_violin`
- `geom_bar`
- `geom_col`

3.3.1 `geom_point`

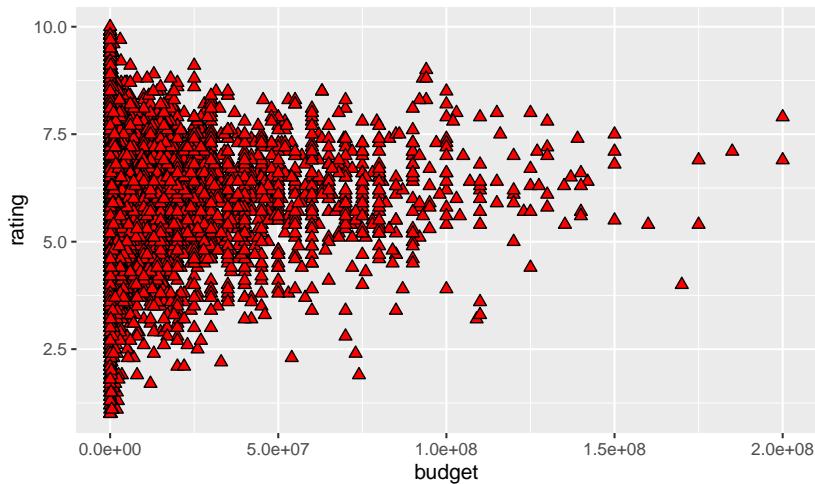
We hebben `geom_point` al gebruikt om onze eerste grafiek te maken, waarbij we twee variabelen in kaart brachten op de esthetica *x* en *y*. Er zijn echter nog enkele andere aesthetics's die met deze laag kunnen worden ingesteld. Laten we ze eens in meer detail bekijken.

- **x**: dit bepaalt de positie van de punten langs de x-as

- **y:** hiermee bepaal je de positie van de punten langs de y-as
- **color:** hiermee bepaal je de kleur van de punten
- **shape:** hiermee bepaal je het type punten dat uitgezet moet worden²
- **fill:** hiermee bepaal je hoe de punten gevuld worden (voor vormen 21-25)
- **size:** hiermee bepaal je de grootte van de punten
- **stroke:** hiermee bepaal je de breedte van de rand
- **alpha:** hiermee bepaal je de mate van doorzichtigheid

De onderstaande grafiek bijvoorbeeld toont zwarte driehoeken, gevuld met rood, met een grootte van 2. Merk op dat de positie van de driehoeken precies dezelfde is als de positie van de punten in de vorige grafiek.

```
ggplot(data = movies) +
  geom_point(mapping = aes(x = budget, y = rating),
             shape = 24,
             fill = "red",
             color = "black",
             size = 2)
```

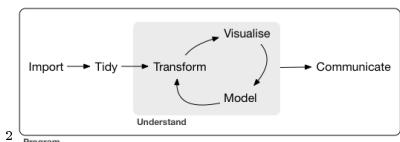


Maar wacht, er is hier iets belangrijks aan de hand! Terwijl de x- en y-aesthetics binnen de aes-mapping werden gedefinieerd, werden de andere aesthetic er buiten gedefinieerd. Waarom is dat?

In feite kunnen aesthetics op twee verschillende manieren worden ingesteld:

1. Ze kunnen worden *mapped* naar een variabele in de dataset.
2. Ze kunnen worden ingesteld op één vaste waarde.

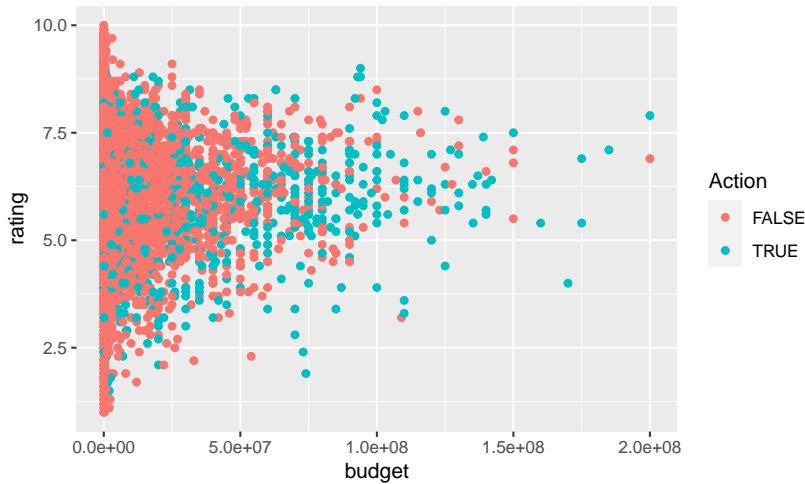
In ons voorbeeld worden x en y gekoppeld aan twee variabelen in de gegevens, nl. budget en rating, terwijl de andere aesthetics, vorm,



Het **tidyverse** is een set van pakketten voor data science die in harmonie werken omdat ze gemeenschappelijke data representaties en API ontwerpen delen. Het **tidyverse**-pakket is ontworpen om het gemakkelijk te maken om kernpakketten van de tidyverse te installeren en te laden met een enkel commando. De tidyverse bevat pakketten zoals: ggplot2, dplyr, tidyr, readr, purrr, tibble, hms, stringr, lubridate,forcats, jsonlite, readxl, broom, en anderen. Hadley Wickham kan beschouwd worden als de grondlegger van het tidyverse. De beste plaats om deze pakketten te leren kennen is door dit boek te lezen: R for Data Science, geschreven door Hadley en Garrett Grolemund.

fill, color en size, worden ingesteld op vaste waarden. Hoewel sommige aesthetics typisch altijd gemapt worden, zoals x- en y-posities, kunnen sommige andere zowel een vaste waarde als een gemapte variabele zijn. Bijvoorbeeld, wat gebeurt er als we de kleur van punten toewijzen aan een variabele, zeg de variabele Action.

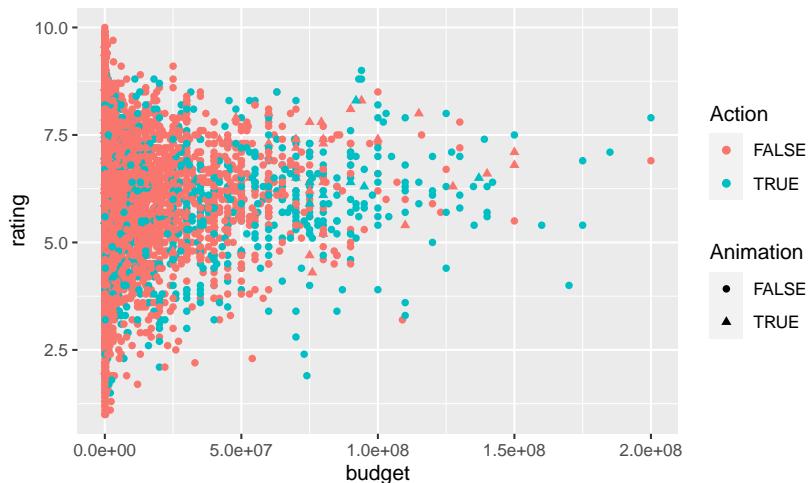
```
ggplot(data = movies) +
  geom_point(mapping = aes(x = budget,
                           y = rating,
                           color = Action))
```



We zien dat de punten nu gekleurd zijn met betrekking tot de waarde in de variabele Action. Action films krijgen een groene kleur, terwijl andere films een rode kleur krijgen, wat we kunnen zien in de legende die verscheen.

Ook de andere vaste aesthetics kunnen worden gebruikt in een mapping. Het volgende voorbeeld gebruikt de variabele Animatie voor de shape.

```
ggplot(data = movies) +
  geom_point(mapping = aes(x = budget,
                           y = rating,
                           color = Action,
                           shape = Animation))
```



Geweldig! We begrijpen nu volledig de geom_point laag, en de werking van de aesthetics-mapping. Nu is het tijd om enkele andere geometrische layers te bekijken. We beginnen met histogrammen.

3.3.2 geom_histogram

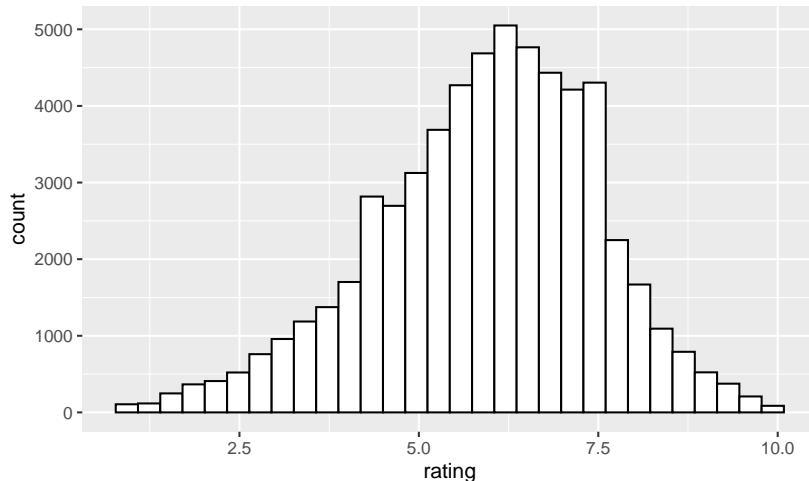
De *geom_histogram* layer kan worden gebruikt om een histogram uit te zetten. Zoals je al zou moeten weten, geeft een histogram de verdeling van **een** continue variabele weer. Bijgevolg moet er alleen een x-aesthetic worden ingesteld, en geen y-aesthetic. De volledige lijst van aesthetic's is als volgt:

- **x**: dit bepaalt de variabele die gebruikt moet worden
- **color**: hiermee bepaal je de kleur van de randen
- **fill**: hiermee bepaal je met welke kleur het histogram gevuld wordt
- **size**: hiermee bepaal je de grootte van de rand
- **linetype**: hiermee bepaal je het type van de rand ³
- **alpha**: hiermee bepaal je de mate van doorzichtigheid
- **weight**: hiermee bepaal je hoe de waarnemingen gewogen moeten worden. Standaard wordt elke waarneming als één gewogen.

³ Let op de hoofdletter I

Gebruik makend van onze kennis over het gebruik van aesthetics van voorheen, is het nu heel eenvoudig om een histogram te maken. Laten we een histogram maken voor de beoordeling van films. We geven het een zwarte rand met een witte vulling. Klaar om te proberen?

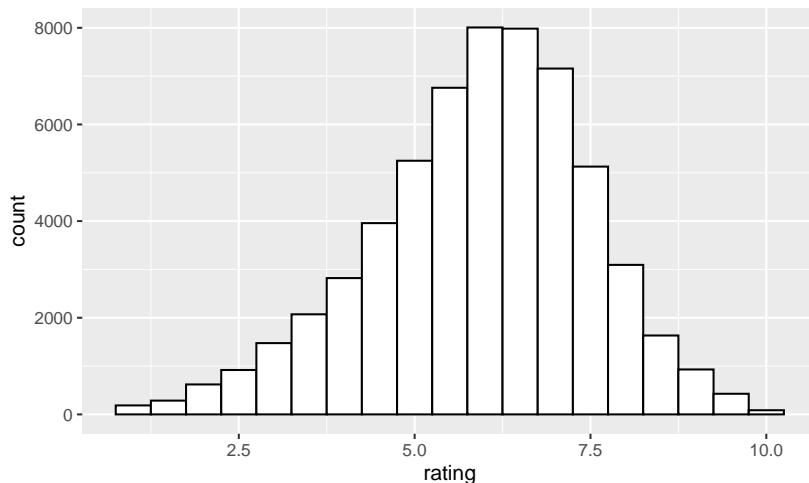
```
ggplot(movies) +
  geom_histogram(aes(rating), color = "black", fill = "white")
```



Merk op dat verschillende dingen werden weggelaten in deze twee lijnen van codes. In het bijzonder de argumentnaam *data* in ggplot, *mapping* in geom_histogram en *x* in aes. Aangezien we weten dat dit de eerste argumenten van deze functies zijn, kunnen we ze veilig weglaten, zolang we de juiste volgorde van argumenten aanhouden. We kunnen echter *color* en *fill* niet weglaten, omdat dit niet het tweede en derde argument van geom_histogram zijn. Speel in geval van twijfel op veilig en schrijf de juiste argumentnamen.

Dat is echter niet het enige dat hier opvallend is. Inderdaad, er verschijnt een waarschuwing: `stat_bin() using bins = 30. Pick better value with binwidth.` Deze waarschuwing herinnert ons aan het feit dat een standaard waarde voor het aantal bins is gekozen door geom_histogram, die waarschijnlijk niet geschikt is voor onze grafiek. We kunnen de binbreedte veranderen door deze als argument toe te voegen aan de aanroep geom_histogram.

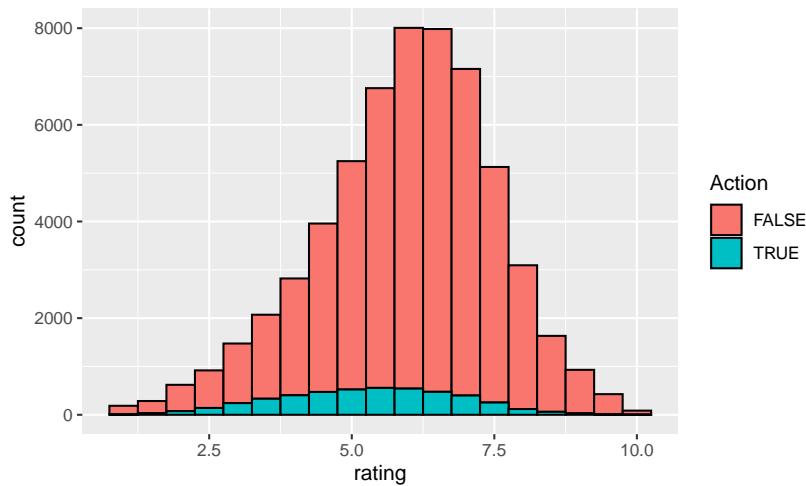
```
ggplot(movies) +
  geom_histogram(aes(rating), color = "black", fill = "white", binwidth = 0.5)
```



Vaak heeft de binbreedte een belangrijke invloed op hoe het verkregen histogram eruit ziet. Zorgvuldig configureren van dit argument door te experimenteren met verschillende waarden is daarom belangrijk.

In de laatste plot hebben we een vaste kleur gebruikt voor het vullen van de balken van het histogram. Maar zoals we ondertussen al weten, kunnen we die ook toewijzen aan een variabele in de gegevens. Laten we de variabele *Action* nog een keer gebruiken.

```
ggplot(movies) +
  geom_histogram(aes(rating, fill = Action), color = "black", binwidth = 0.5)
```



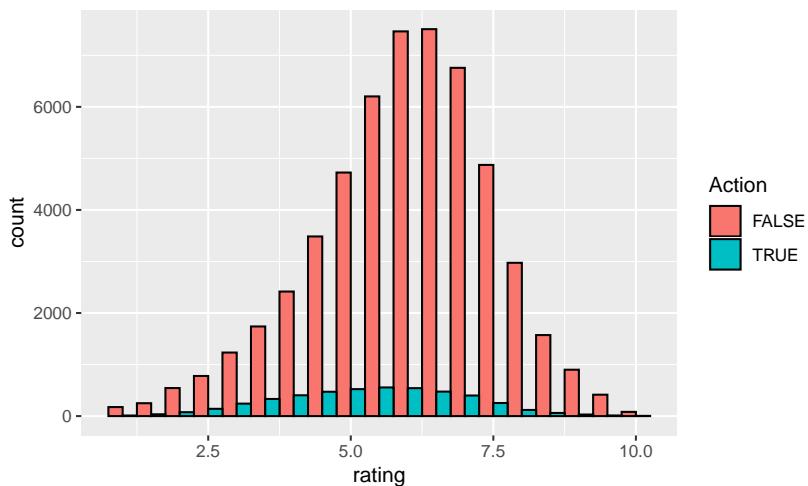
Merk op hoe we het fill-argument binnen de `aes` functie hebben geplaatst. Nu is elke staaf gevuld met twee kleuren: een deel voor actiefilms, en de rest voor andere films. Hoewel het niet erg duidelijk is in deze grafiek, lijkt het erop dat het centrum van het histogram voor actiefilms iets meer naar links ligt.

Merk op hoe de notatie verandert bij de overgang naar de `aes`-

mapping: variabelennamen worden altijd zonder aanhalingstekens gebruikt, terwijl vaste aesthetics (kleuren, vormen, linetypes) met aanhalingstekens worden gebruikt (behalve voor getallen). Het is belangrijk om dit niet door elkaar te halen! Nooit aanhalingstekens rond namen van variabelen!

Standaard zijn de histogrammen voor de verschillende *fills stacked*, d.w.z. boven elkaar geplaatst. We kunnen echter het *position* argument van `geom_histogram` gebruiken om de staven naast elkaar te plaatsen, of *dodged*.

```
ggplot(movies) +
  geom_histogram(aes(rating, fill = Action),
                color = "black", binwidth = 0.5, position = "dodge")
```



Met `position = "dodge"` worden de balken voor actiefilms en niet-actiefilms naast elkaar geplaatst, in plaats van boven elkaar. We kunnen teruggaan naar de oorspronkelijke grafiek door `position = "stack"` te gebruiken, of door dit argument weg te laten. Later zullen we zien hoe we beter met dergelijke zaken kunnen omgaan door gebruik te maken van rasters van verschillende plots, of zogenaamde *facets*.

Alles goed tot nu toe? Laten we eens kijken naar een andere manier om de verdeling van continue variabelen te visualiseren, namelijk de boxplot.

3.3.3 `geom_boxplot`

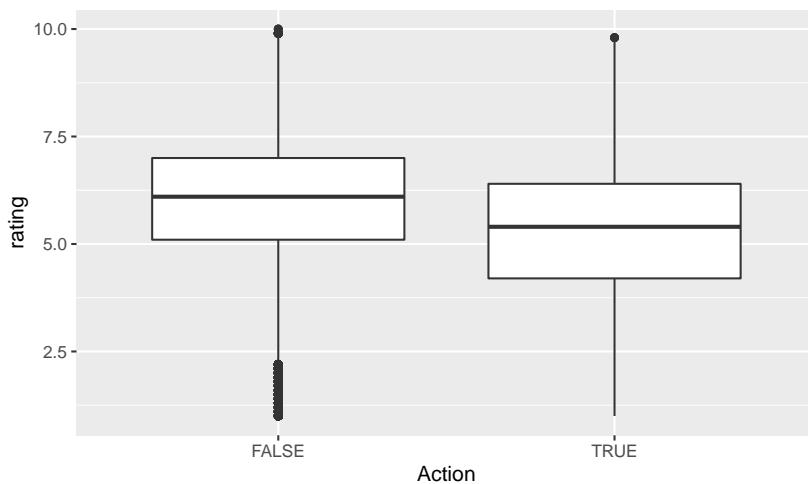
Een boxplot plaatst de waarden van de variabele op de y-as. Dus, als we een boxplot willen maken voor ratings, moeten we `aes(y = ratings)` gebruiken. Er is hier echter iets tricky aan de hand... De aesthetics voor `geom_boxplot` is de volgende

- `x`: dit definieert de variabele die voor de x-as wordt gebruikt

- **y:** dit definieert de variabele voor de y-as
- **color:** hiermee bepaal je de kleur van de randen
- **fill:** hiermee bepaal je hoe de boxplot wordt opgevuld
- **size:** hiermee bepaal je de grootte van de rand
- **linetype:** hiermee bepaal je het type van de rand
- **alpha:** Hiermee bepaal je de mate van doorzichtigheid

Dus, de boxplot heeft zowel een x-variabele als een y-variabele nodig? Dat lijkt op het eerste gezicht vreemd. De reden hierachter is dat in de filosofie van ggplot, altijd *iets* moet geplot worden op zowel de x- als de y-as. Hoewel alleen een x-variabele wordt gegeven aan een histogram, zal het frequenties berekenen om op de y-as te plotten. Bij boxplots gebeurt dat echter niet. Bijgevolg moet de x-as worden gebruikt om verschillende categorieën in kaart te brengen waarvan de verdeling vervolgens kan worden vergeleken. Zo kunnen we bijvoorbeeld de waardering voor actiefilms vergelijken met die voor andere films.

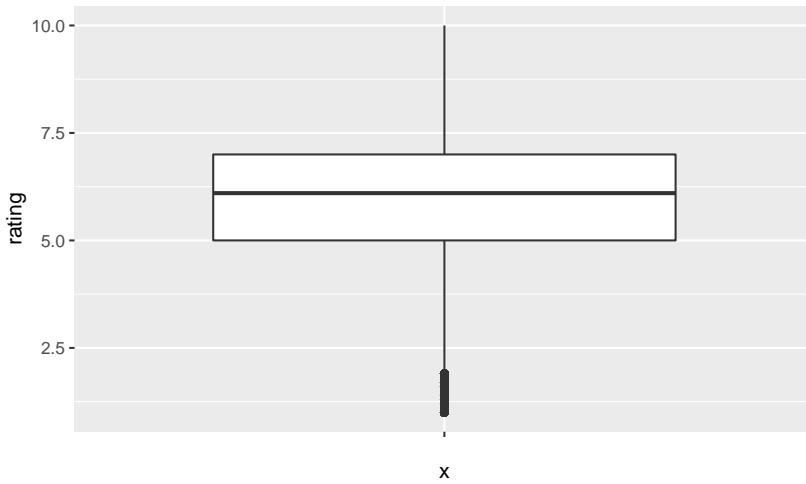
```
ggplot(movies) +
  geom_boxplot(aes(Action, rating))
```



Hier zien we dat, zoals we al vermoedden, actiefilms een lagere waardering hebben in vergelijking met andere films. We kunnen verder de kleur en de vulling van de boxplot veranderen zoals voorheen, alsook het linetype, de grootte van de rand, of de transparantie.

Maar wat als we gewoon een boxplot willen tekenen van de totale waardering, zonder een variabele te moeten specificeren voor de x-as? Een kleine workaround is hier nodig. Een mogelijkheid is om een *empty string* te gebruiken voor de x-as toewijzing.

```
ggplot(movies) +
  geom_boxplot(aes("", rating))
```



Merk op dat dit het label “x” creëert voor de x-as, waar we normaal de naam van de variabele zouden vinden die erop is weergegeven.

Later zullen we zien hoe we dit label kunnen weglaten om onze grafiek een beetje mooier te maken.

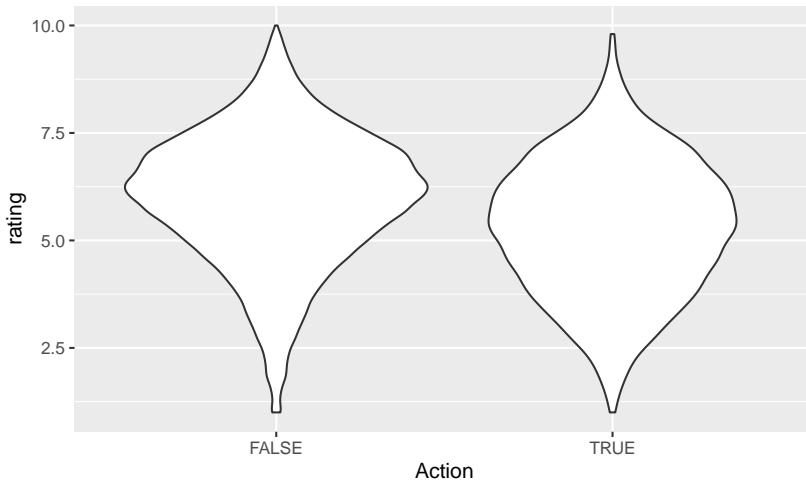
3.3.4 *geom_violin*

De violin-plot is vergelijkbaar met de boxplot, maar geeft in meer detail weer waar de massa van de waarden zich bevindt. De aesthetics is dezelfde als bij een boxplot.

- **x**: dit definieert de variabele die voor de x-as wordt gebruikt
- **y**: dit definieert de variabele voor de y-as
- **color**: hiermee bepaal je de kleur van de randen
- **fill**: hiermee bepaal je hoe de plot wordt opgevuld
- **size**: hiermee bepaal je de grootte van de rand
- **linetype**: hiermee bepaal je het type van de rand
- **alpha**: Hiermee bepaal je de mate van doorzichtigheid

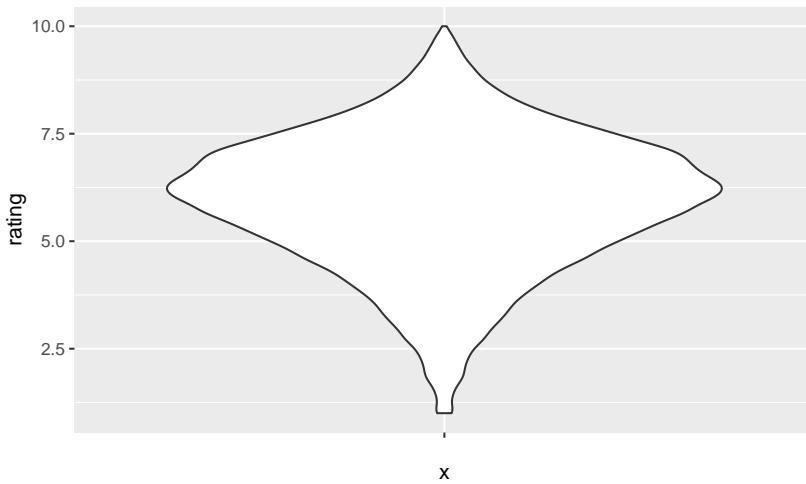
Laten we dezelfde grafieken maken, nu met de violin-plot.

```
ggplot(movies) +
  geom_violin(aes(Action, rating))
```



Het is nu waarschijnlijk wel duidelijk waar dit type grafiek zijn naam aan te danken heeft. Zoals je kunt zien, houden violin-plots het midden tussen boxplots en histogrammen. Omdat hun breedte genormaliseerd is, kunnen ze beter gebruikt worden voor vergelijkingen. Ook hier kunnen we dezelfde workaround gebruiken als we de algemene verdeling willen plotten.

```
ggplot(movies) +
  geom_violin(aes("", rating))
```



Tot nu toe hebben we drie verschillende manieren gezien om de verdeling van continue variabelen te analyseren. Nu gaan we kijken naar barplots, die kunnen worden gebruikt om categorische verdelingen weer te geven.

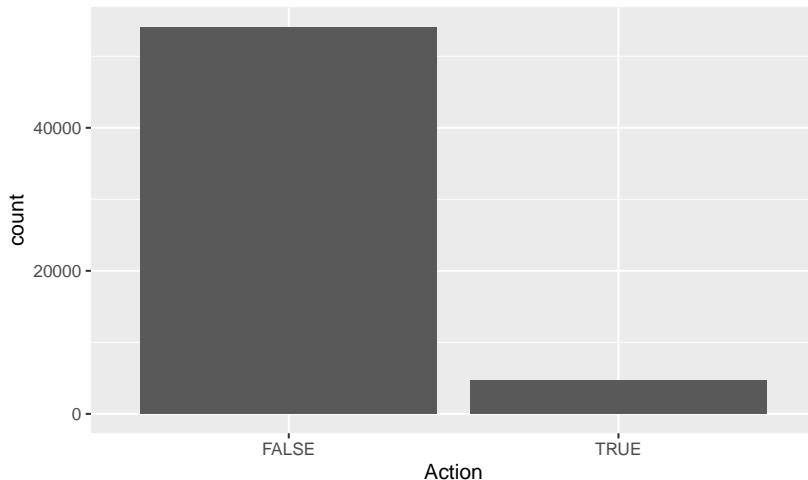
3.3.5 geom_bar

Net als een histogram, heeft een barplot alleen een x variabele nodig. Het verschil is dat deze variabele categorisch moet zijn, terwijl ze voor histogrammen continu moet zijn. De volledige lijst van aesthetics is de volgende:

- **x**: dit bepaalt de variabele die voor de x-as gebruikt wordt
- **color**: hiermee bepaal je de kleur van de randen
- **fill**: hiermee bepaal je hoe de balken gevuld worden
- **size**: hiermee bepaal je de grootte van de rand
- **linetype**: hiermee bepaal je het type van de rand
- **alpha**: hiermee bepaal je de mate van transparantie
- **weight**: hiermee bepaal je hoe de waarnemingen gewogen moeten worden. Standaard wordt elke waarneming als één gewogen.

We kunnen een eenvoudig staafdiagram maken dat laat zien hoeveel actiefilms er zijn, en hoeveel andere films, en wel als volgt.

```
ggplot(movies) +
  geom_bar(aes(Action))
```



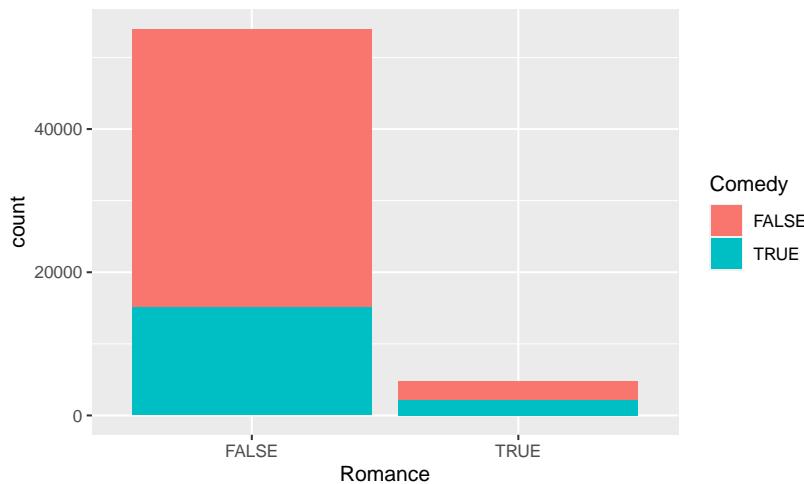
Verder kunnen we hier nog kleuren aan toevoegen, volgens het aantal Animatiefilms. We zien onmiddellijk dat er bijna geen actiefilms zijn die ook animatiefilms zijn.

```
ggplot(movies) +
  geom_bar(aes(Action, fill = Animation))
```



We kunnen hetzelfde doen voor Romantische en Komedie films.

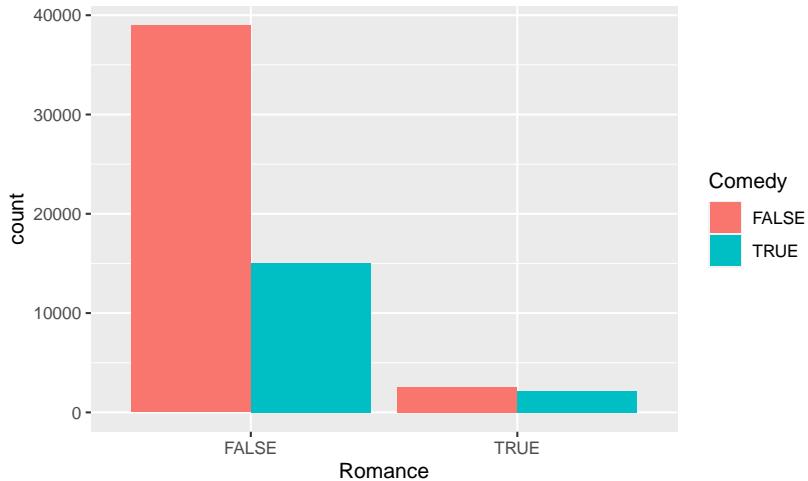
```
ggplot(movies) +
  geom_bar(aes(Romance, fill = Comedy))
```



Daarentegen is hier te zien dat ongeveer de helft van de romantische films ook komedies zijn, wat meer is in vergelijking met niet-romantische films.

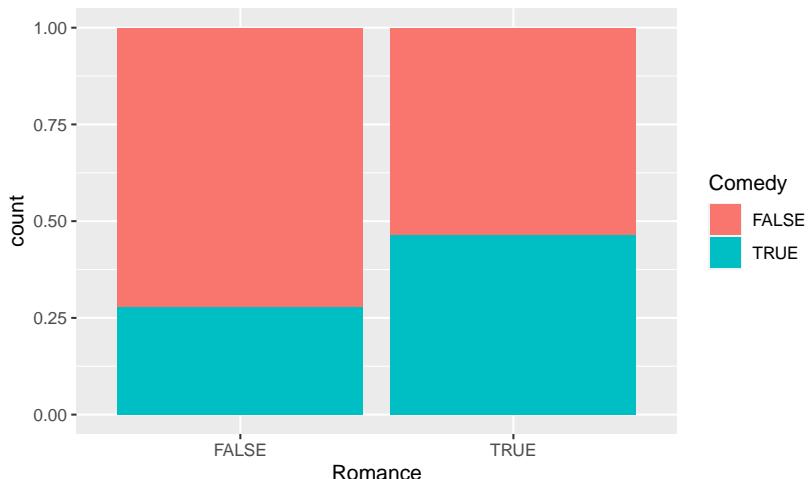
Vergeet niet dat we in het geval van histogrammen de position kunnen veranderen in "dodge", waardoor de balken naast elkaar kwamen te staan. Hetzelfde kan hier worden gedaan.

```
ggplot(movies) +
  geom_bar(aes(Romance, fill = Comedy), position = "dodge")
```



Een derde mogelijkheid die voor de position beschikbaar is, is de staven te verlengen zodat zij dezelfde hoogte hebben. Het resultaat is dat we de verdeling van de waarden als een deel van een geheel zullen waarnemen. In plaats van de absolute frequentie zullen de labels op de y-as nu de procentpunten weergeven.

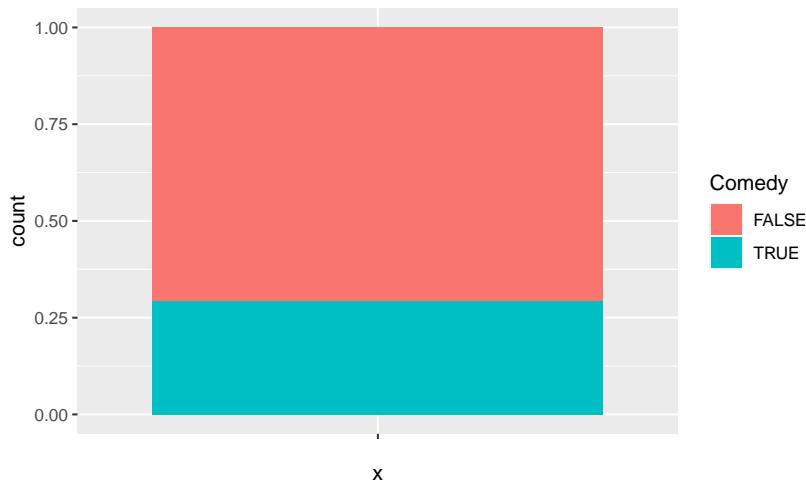
```
ggplot(movies) +
  geom_bar(aes(Romance, fill = Comedy), position = "fill")
```



Tenslotte, als we dit willen doen om de verdeling van één variabele te tonen, kunnen we dezelfde workaround gebruiken als voorheen en de x-aesthetic op "" zetten. De plot hieronder zal het deel van alle films tonen die komedies zijn.⁴

```
ggplot(movies) +
  geom_bar(aes("", fill = Comedy), position = "fill")
```

⁴ Merk op dat het in dergelijke gevallen volkomen logisch is de plot minder breed te maken, of hem 90 graden om te draaien en minder hoog te maken. We komen hier later op terug.



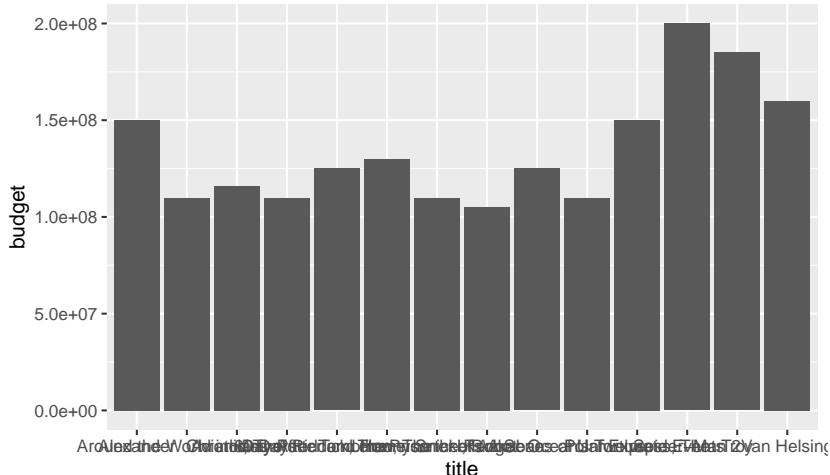
3.3.6 geom_col

Wanneer we `geom_bar` gebruiken, wordt de hoogte van de balken berekend aan de hand van de frequentie van de categorische variabele. Soms willen we echter een staafdiagram plotten met waarden die al in de data zitten, of waarden die we zelf hebben berekend. Bijvoorbeeld, wat als we een staafdiagram willen met het budget van een reeks films? In zo'n geval kunnen we `geom_col` gebruiken. “col” geeft aan dat we een kolom in de gegevens willen gebruiken om de hoogte van de balken in te stellen. De aesthetics is hetzelfde voor `geom_bar`, alleen moeten we nu een variabele specificeren voor de y-as uiteraard.

- **x:** dit bepaalt de variabele die voor de x-as gebruikt wordt
- **x:** dit bepaalt de variabele die voor de y-as gebruikt wordt
- **color:** hiermee bepaal je de kleur van de randen
- **fill:** hiermee bepaal je hoe de balken gevuld worden
- **size:** hiermee bepaal je de grootte van de rand
- **linetype:** hiermee bepaal je het type van de rand
- **alpha:** hiermee bepaal je de mate van transparantie
- **weight:** hiermee bepaal je hoe de waarnemingen gewogen moeten worden. Standaard wordt elke waarneming als één gewogen.

Laten we een staafdiagram maken van het budget van alle films uit 2004 waarvan het budget hoger was dan 100 miljoen.

```
filter(movies, year == 2004, budget > 100000000) %>%
  ggplot() +
  geom_col(aes(title, budget))
```



Zie je iets vreemds in de code? Maak je geen zorgen als je de eerste regel niet begrijpt. Al wat je moet weten is dat we films uit 2004 hebben gefilterd met een budget hoger dan 100 miljoen. Het vreemde uitzienende `%>%` symbool zal ervoor zorgen dat deze gegevens doorgegeven worden aan ggplot. We zullen hier in een andere sessie op terugkomen.⁵

We hebben nu filmtitels uitgezet op de x-as en budget op de y-as. Geweldig! Of toch niet? De waarden op de x-as zijn wat onoverzichtelijk en onleesbaar. Het is nu tijd om aandacht te besteden aan de layout van onze plots!



5

Het piping-symbool werd voor het eerst geïntroduceerd in het pakket `magrittr`, genoemd naar de Belgische surrealistische kunstenaar René Magritte, bekend van zijn schilderij *The Treachery of Images*, oftewel *Ceci n'est pas un pipe*.

3.3.7 Other geometrics

Tot dusver hebben we de belangrijkste geom-layers gebruikt om een-eenvoudige visualisaties te maken: scatterplots, histogrammen, boxplots, violinplots en barplots. We hebben echter slechts het topje van de ijsberg besproken, want er bestaan nog veel meer types, sommige een-eenvoudig en sommige meer geavanceerd. Een overzicht van alle geoms en hun toepassingen kan gevonden worden in de ggplot Cheat Sheet, waarvan hier een uittreksel wordt getoond. Wees niet bang om iets uit te proberen!

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.	
Graphical Primitives <pre>a <- ggplot(seals, aes(x = long, y = lat)) b <- ggplot(economics, aes(date, unemploy)) a + geom_blank() # (Useful for expanding limits) a + geom_curve(aes(yend = lat + delta_lat, xend = long + delta_long, curvature = 2)) x, y, end, alpha, angle, color, curvature, line_type, size b + geom_path(lineend = "butt", linejoin = "round", linemitre = 1) x, y, alpha, color, group, line_type, size b + geom_polygon(aes(group = group)) x, y, alpha, color, fill, group, line_type, size a + geom_rect(aes(min = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat)) xmax, xmin, ymax, ymin, alpha, color, fill, line_type, size b + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) x, y, alpha, color, fill, group, line_type, size a + geom_segment(aes(yend = lat + delta_lat, xend = long + delta_long)) x, y, end, alpha, color, line_type, size a + geom_spoke(aes(yend = lat + delta_lat, xend = long + delta_long)) x, y, angle, radius, alpha, color, line_type, size</pre>	Two Variables <pre>A C e : geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, just e : geom_point() x, y, alpha, color, fill, shape, size, stroke e : geom_quantile() x, y, alpha, color, group, line_type, size, weight e : geom_rug(sides = "b") x, y, alpha, color, line_type, size e : geom_smooth(method = lm) x, y, alpha, color, fill, group, line_type, size, weight C A B e : geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, just Discrete X, Continuous Y f <- ggplot(mpg, aes(class, hwy)) f + geom_bar(stat = "identity") x, y, alpha, color, fill, line_type, size, weight f + geom_boxplot() x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, line_type, shape, size, weight f + geom_dotplot(binaxis = "y", stackdir = "center") x, y, alpha, color, fill, group f + geom_violin(scale = "area") x, y, alpha, color, fill, group, line_type, size, weight</pre>
One Variable <pre>c <- ggplot(mpg, aes(hwy)) c + geom_area(stat = "bin") x, y, alpha, color, fill, line_type, size a + geom_area(aes(y = ..density..), stat = "bin") x, y, alpha, color, fill, group, line_type, size, weight c + geom_density(kernel = "gaussian") x, y, alpha, color, fill c + geom_freqpoly() x, y, alpha, color, group, line_type, size a + geom_freqpoly(aes(y = ..density..)) x, y, alpha, color, group, line_type, size, weight c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, line_type, size, weight a + geom_histogram(aes(y = ..density..)) x, y, alpha, color, fill, group, line_type, size, weight Discrete d <- ggplot(mpg, aes(rft)) d + geom_bar() x, alpha, color, fill, line_type, size, weight</pre>	Continuous X, Continuous Y <pre>e : geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, just e : geom_jitter(height = 2, width = 2) x, y, alpha, color, fill, shape, size e : geom_point() x, y, alpha, color, fill, shape, size, stroke e : geom_quantile() x, y, alpha, color, group, line_type, size, weight e : geom_rug(sides = "b") x, y, alpha, color, line_type, size e : geom_smooth(method = lm) x, y, alpha, color, fill, group, line_type, size, weight</pre>
	Continuous Bivariate Distribution <h>h < ggplot(diamonds, aes(carat, price))</h> <pre>h : geom_bin2d(binwidth = c(0.25, 500)) x, y, alpha, color, fill, line_type, size, weight</pre>
	Continuous Function <i>i < ggplot(economics, aes(date, unemploy))</i> <pre>i : geom_area() x, y, alpha, color, fill, line_type, size i : geom_line() x, y, alpha, color, group, line_type, size i : geom_step(direction = "hv") x, y, alpha, color, group, line_type, size</pre>
	Visualizing error <pre>j < ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se)) j + geom_crossbar(fatten = 2) x, y, ymax, ymin, alpha, color, fill, group, line_type, size j + geom_errorbar() x, ymax, ymin, alpha, color, group, line_type, size, width (also geom_errorbarh()) j + geom_linerange() x, y, ymax, ymin, alpha, color, group, line_type, size j + geom_pointrange() x, y, ymin, ymax, alpha, color, fill, group, line_type, shape, size</pre>
	Maps <pre>data <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests))) map <- map_data("state") k <- ggplot(data, aes(fill = murder)) k + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat) map_id, alpha, color, fill, line_type, size</pre>
	Three Variables <pre>seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)) l <- ggplot(seals, aes(long, lat)) l + geom_contour(aes(z = z)) x, y, z, alpha, colour, group, line_type, size, weight</pre>
	<pre>l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE) x, y, alpha, fill</pre>
	<pre>l + geom_tile(aes(fill = z)) x, y, alpha, color, fill, line_type, size, width</pre>

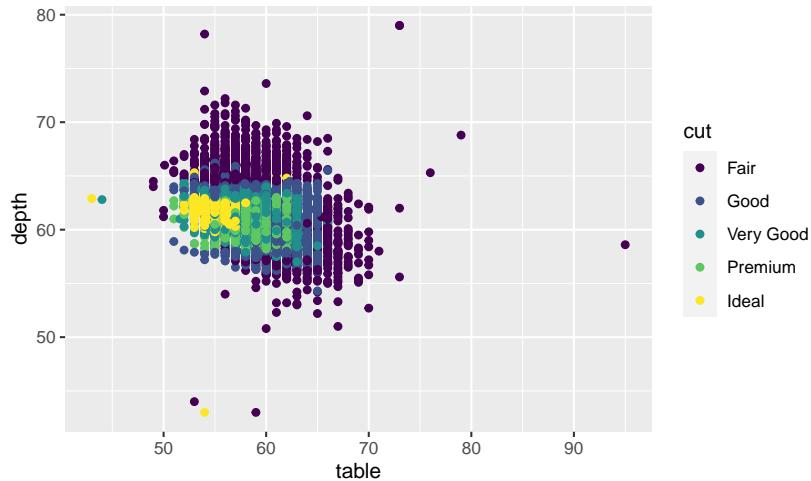
3.4 Layout van onze grafieken verbeteren

Tot nu toe hebben we vooral gekeken naar verschillende soorten plots en hoe we die op onze gegevens kunnen plotten. In deze sectie zullen we ons concentreren op de presentatie van de plot, bv. titels, kleuren, assen, enz. De in dit deel geïntroduceerde concepten kunnen voor elk type plot worden toegepast, ongeacht welk geometrisch object wordt gebruikt.

In dit deel zal de dataset “diamanten” worden gebruikt. De onderstaande plot zal als uitgangspunt worden gebruikt.⁶

```
ggplot(diamonds) +
  geom_point(aes(table, depth, color = cut))
```

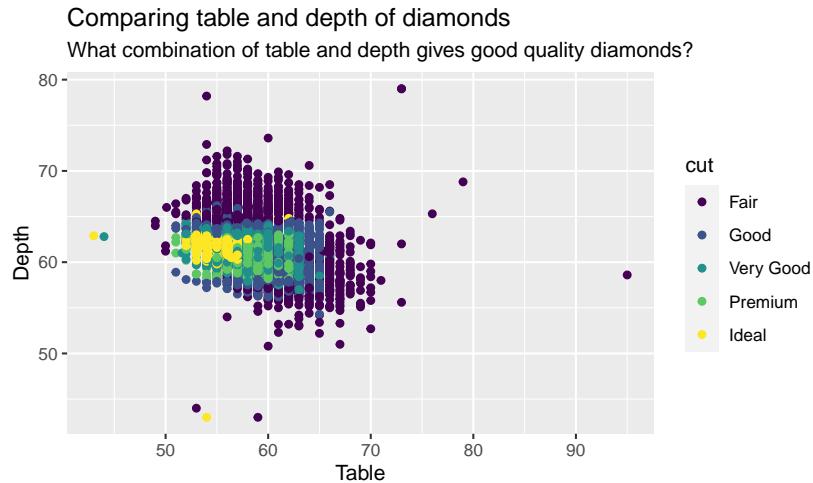
⁶ De tabel van een diamant verwijst naar het vlakke facet van de diamant dat kan worden gezien wanneer de steen naar boven wordt gekeerd. De diepte van een diamant is de hoogte (in millimeters) gemeten van de culet tot de tafel.



3.4.1 Titels

Een van de belangrijkste dingen om aan onze plot toe te voegen zijn titels. Titels worden gebruikt om betekenis te geven aan zowel de assen als de plot zelf. De meest eenvoudige manier om titels toe te voegen is door gebruik te maken van de functie `labs()`. In deze functie kunnen o.a. volgende argumenten worden ingesteld: de titel, de ondertitel, x voor het x label en y voor het y label. De `labs` functie kan gewoon aan de plot worden toegevoegd als een extra laag.

```
data("diamonds")
ggplot(diamonds) +
  geom_point(aes(table, depth, color = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth")
```



Je zult zien dat onze grafiek er al veel beter uitziet als er titels aan toegevoegd zijn! Er is echter nog veel meer te verbeteren.

3.4.2 Theme

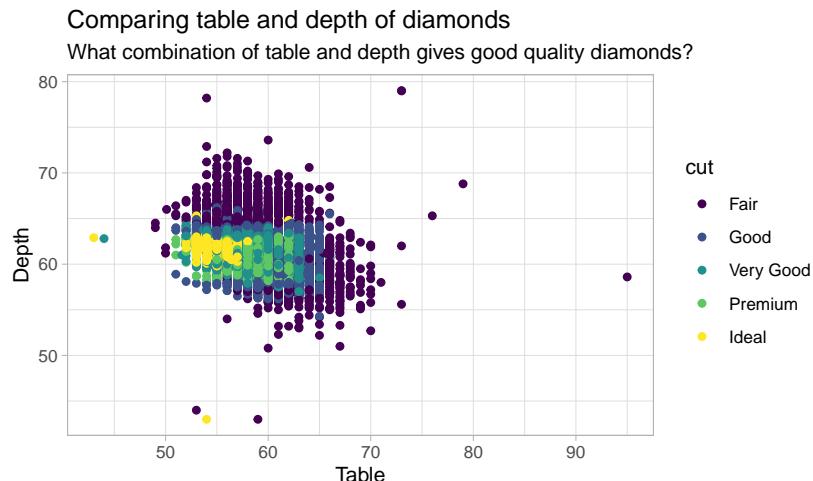
Het *theme* van een plot bepaalt het algemene uiterlijk: de rasterlijnen, de achtergrond, de grootte van de tekst, titels en legende, de positie van de legende, enz. Het thema kan handmatig worden gedefinieerd door een `theme()` laag toe te voegen aan de plot en door de benodigde argumenten in te stellen. (Je kunt kijken op `?theme` om te zien welke argumenten beschikbaar zijn). Dit is echter een omslachtige aanpak. Gelukkig zijn er enkele voorgedefinieerde thema's voorzien in ggplot:

- `theme_gray`: het standaardthema (tot nu toe gebruikt)
- `theme_bw`: een thema voor zwart-wit plots
- `theme_dark`: een donker thema voor contrast
- `theme_classic`: een minimaal thema
- `theme_light`: een ander minimaal thema
- `theme_linedraw`: nog een minimaal thema
- `theme_minimal`: nog een minimaal thema
- `theme_void`: een leeg thema

Voel je vrij om met sommige van deze thema's te experimenteren. Bij voorkeur kunt u een aantal van de minimale thema's gebruiken. Hier, gebruikten we het `theme_light` thema.⁷

```
ggplot(diamonds) +
  geom_point(aes(table, depth, color = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light()
```

⁷ Voor de meeste lagen is het niet belangrijk in welke volgorde ze aan een plot worden toegevoegd. Echter, als je handmatig wijzigingen aanbrengt met `theme`, zorg er dan voor dat je ze na een voorgedefinieerd thema plaatst, anders kunnen je wijzigingen overgeschreven worden.



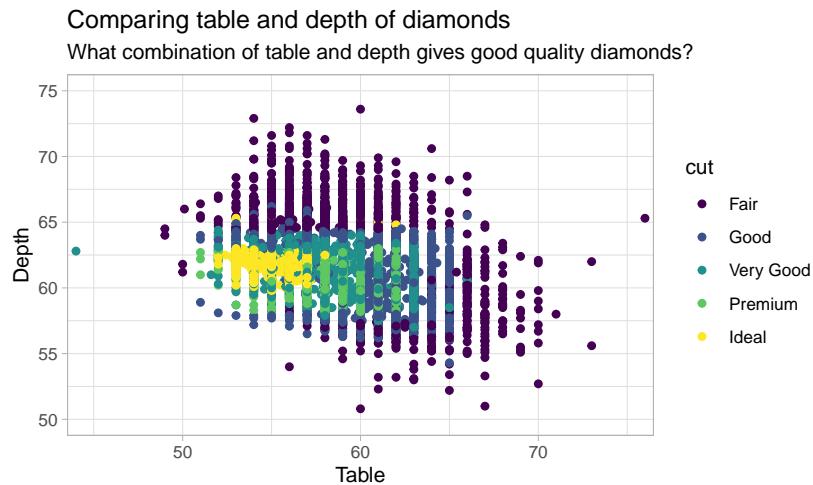
Wanneer je nog niet tevreden bent één van deze thema's, kunt je het pakket `ggthemes` installeren om nog meer thema's te verkrijgen, zoals het thema van *The Economist*, fivethirtyeight.com, of *Google Docs*.

3.4.3 Het coördinatenstelsel

Het uiterlijk van de grafiek wordt niet alleen bepaald door de titels en de grafieken. Ook de assen in het assenstelsel verdienen de nodige aandacht. Een van de dingen die moeten worden bepaald zijn de grenzen van het coördinatenstelsel. Dit kan worden gedaan met de functie `coord_cartesian` en zijn argumenten `xlim` en `ylim`. Beide argumenten verwachten een numerieke vector van lengte twee. Laten we eens kijken hoe dit werkt in ons voorbeeld.⁸

```
ggplot(diamonds) +
  geom_point(aes(table, depth, color = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45, 75), ylim = c(50, 75))
```

⁸ Een cartesisch coördinatenstelsel is een coördinatenstelsel dat elk punt op unieke wijze in een vlak specificeert door een paar numerieke coördinaten, die de getekende afstanden tot het punt zijn van twee vaste loodrecht op elkaar staande gerichte lijnen, gemeten in dezelfde lengte-eenheid. Het is vernoemd naar wetenschapper René Descartes.



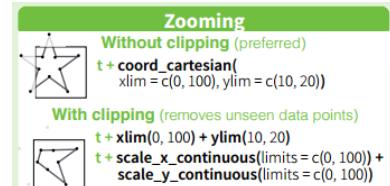
We hebben nu de x-as beperkt tot het interval van 45 tot 75, terwijl we de y-as hebben beperkt tot het interval 50 tot 75. Er zijn enkele alternatieven voor het cartesisch coördinatenstelsel die minder vaak worden gebruikt:

- coord_equal: een coördinatenstelsel waarbij de x-as en de y-as gelijk geschaald zijn (d.w.z. verhouding = 1)
- coord_fixed: een assenstelsel met een vaste verhouding (maar niet noodzakelijk 1)
- coord_polar: een coördinatensysteem voor polaire plots, of cirkeldiagrammen
- coord_map: een assenstelsel voor het plotten van geografische data.

Naast het instellen van de grenzen van het coördinatensysteem, kunnen we ook de breaks op de x-as en de y-as instellen. Dit kan respectievelijk met de functies `scale_x_continuous` en `scale_y_continuous`. Beide functies hebben een `breaks` argument. Dit argument kan worden gegeven als een vector van waarden die als labels op de as moeten worden geplot.⁹ We kunnen de functie `seq` gebruiken om deze vector te maken: d.w.z. `seq(0,10,5)` zal een vector teruggeven die begint bij 0 en oploopt tot tien met intervallen van 5.¹⁰

```
ggplot(diamonds) +
  geom_point(aes(table, depth, color = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45,75), ylim = c(50, 75)) +
```

⁹ Merk op dat de `scale_..._continuous` functies ook een argument `limits` hebben om de grenzen van de assen in te stellen, dat gebruikt kan worden in plaats van `coord_cartesian`. Er is echter een belangrijk verschil. `Coord_cartesian` zal inzoomen op de grenzen zonder andere datapunten weg te gooien. Het instellen van de grenzen binnen de `scale`-functies echter, zal datapunten weggooien en kan je visualisatie vertekenen.



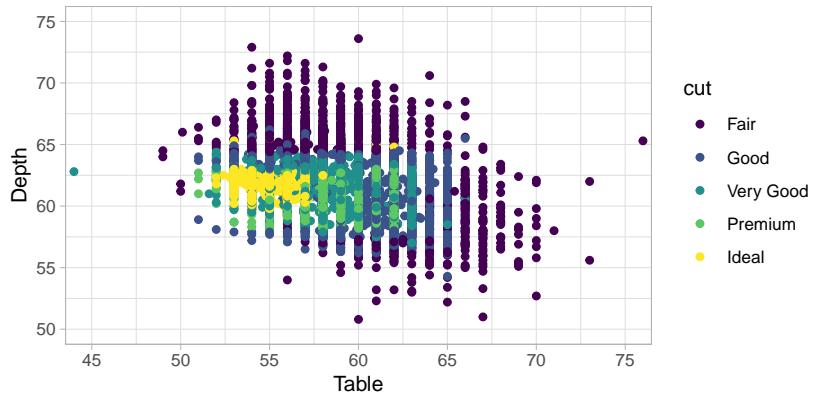
¹⁰ De titels van de assen die we gedefinieerd hebben met de `labs` functie kunnen ook ingesteld worden in de `scale`-functies met het argument `name`. Naarmate je meer vertrouwd raakt met het gebruik van `ggplot2`, zul je vaak merken dat er meerdere manieren zijn om hetzelfde doel te bereiken.

```
scale_x_continuous(breaks = seq(45,75,5)) +  
scale_y_continuous(breaks = seq(50,75,5))
```

Comparing table and depth of diamonds

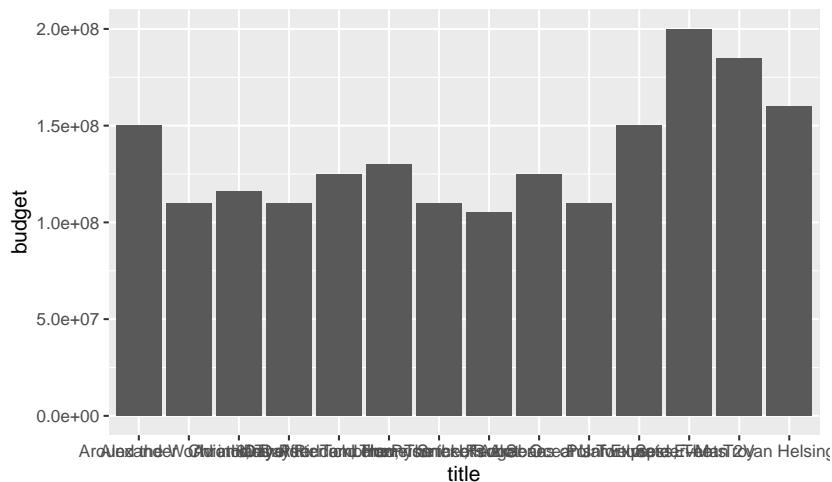
What combination of table

and depth gives good quality diamonds?



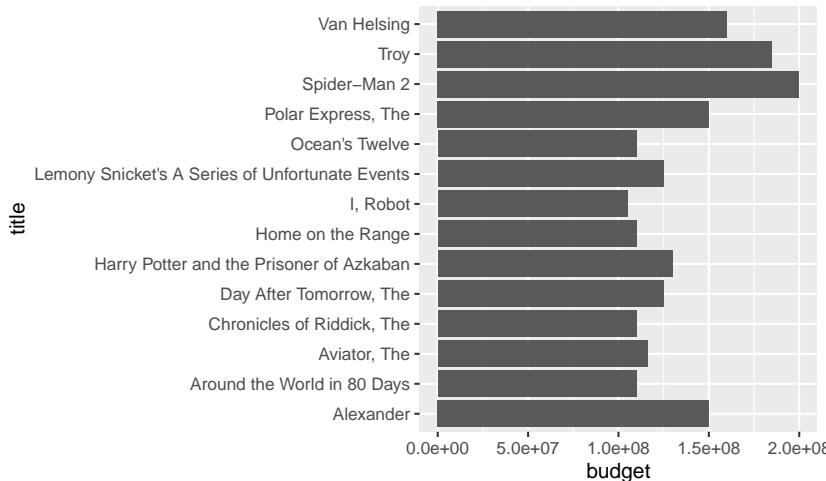
Een andere handige functie is de `coord_flip` functie, die we zullen illustreren met de volgende grafiek die we eerder zagen.

```
filter(movies, year == 2004, budget > 100000000) %>%  
  ggplot() +  
  geom_col(aes(title, budget))
```



Zoals je zich wellicht herinnert, overlapten de filmtitels op de x-as elkaar en waren daardoor onleesbaar. Een manier om dit te verhelpen is door de hele grafiek *om te draaien*, zodat de labels van de x-as op de y-as komen te staan, en horizontaal kunnen worden gelezen.

```
filter(movies, year == 2004, budget > 100000000) %>%  
  ggplot() +  
  geom_col(aes(title, budget)) +  
  coord_flip()
```



Een andere optie is om de oorspronkelijke oriëntatie te behouden, maar de oriëntatie van de x-as te veranderen. Je kunt ze bijvoorbeeld 45 of 90 graden draaien. Dit kan worden gedaan met de `theme` functie. Klaar om te experimenteren? Daag jezelf uit!

Naarmate onze code meer en meer regels bevat, wordt onze plot mooier en mooier! Goed gedaan! Het laatste op onze lijst zijn kleuren.

3.4.4 Color scales

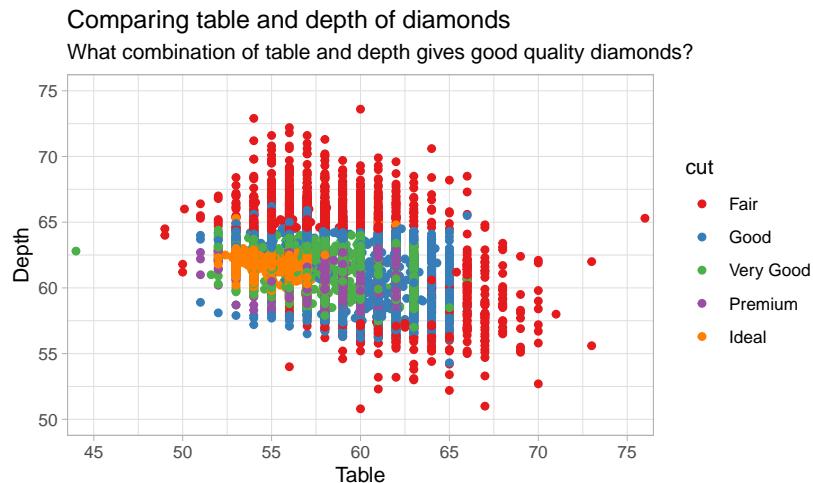
Vaak gebruiken we kleur of fill om categorische gegevens te visualiseren, zoals de kwaliteit in onze grafiek over diamanten. Standaard zal ggplot een regenboog-thema gebruiken. Er zijn echter veel meer paletten beschikbaar. We kunnen deze toevoegen door `scale_color_brewer` of `scale_fill_brewer` te gebruiken, afhankelijk van of het om een kleur of fill-kleur gaat. Beide layers hebben een palette argument, waarvan je hieronder de mogelijke waarden kunt vinden.

Bijvoorbeeld, laten we het Set1 palet gebruiken.

```
ggplot(diamonds) +
  geom_point(aes(table, depth, col = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45,75), ylim = c(50, 75)) +
  scale_x_continuous(breaks = seq(45,75,5)) +
  scale_y_continuous(breaks = seq(50,75,5)) +
  scale_color_brewer(palette = "Set1")
```

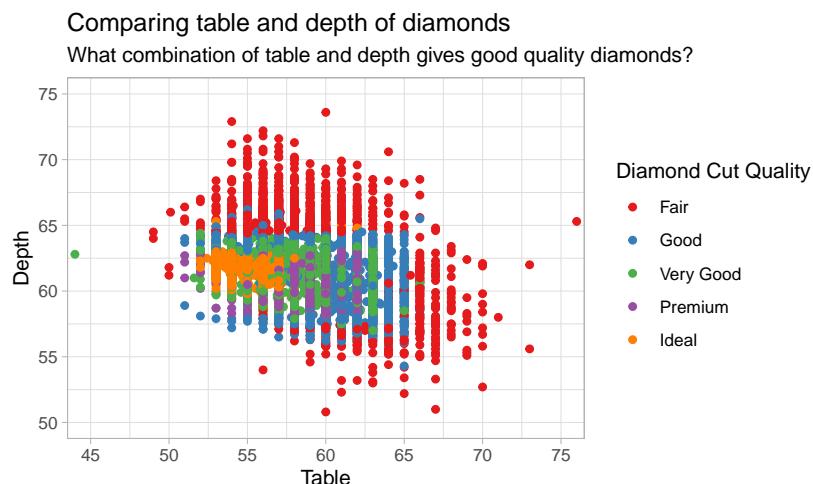
Figure 3.1: R color palettes





De scale_.brewer functies hebben ook het argument `name`, waarmee we de naam van de legende kunnen instellen, en het argument `guide`, waarmee de legende wordt verwijderd als deze op FALSE is gezet.

```
ggplot(diamonds) +
  geom_point(aes(table, depth, col = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45,75), ylim = c(50, 75)) +
  scale_x_continuous(breaks = seq(45,75,5)) +
  scale_y_continuous(breaks = seq(50,75,5)) +
  scale_color_brewer(palette = "Set1", name = "Diamond Cut Quality")
```



Naast de standaard kleurenpaletten die beschikbaar zijn, zijn er nog veel meer te vinden in de pakketten `ggthemes` en `ggsci`. Ze kunnen

worden gebruikt door `scale_color` of `scale_fill` + naam van het palet toe te voegen. Voel je vrij om er nog meer te ontdekken!

3.5 Geavanceerde plots

Vaak wil je plots vergelijken voor verschillende categorieën van een variabele. In ons voorbeeld hebben we gekeken voor welke combinaties van table en depth, de cut-quality van de diamant goed was. Nu willen we weten of er een verschil is tussen de 8 verschillende clarities in de gegevens. Een manier zou zijn om 8 verschillende plots te maken, één voor elk van de niveaus. Dit zou echter omslachtig zijn om te doen. Gelukkig kunnen we de functie `facet_grid` gebruiken om verschillende plots binnen een plot te maken.

3.5.1 Gebruikmaken van Facets

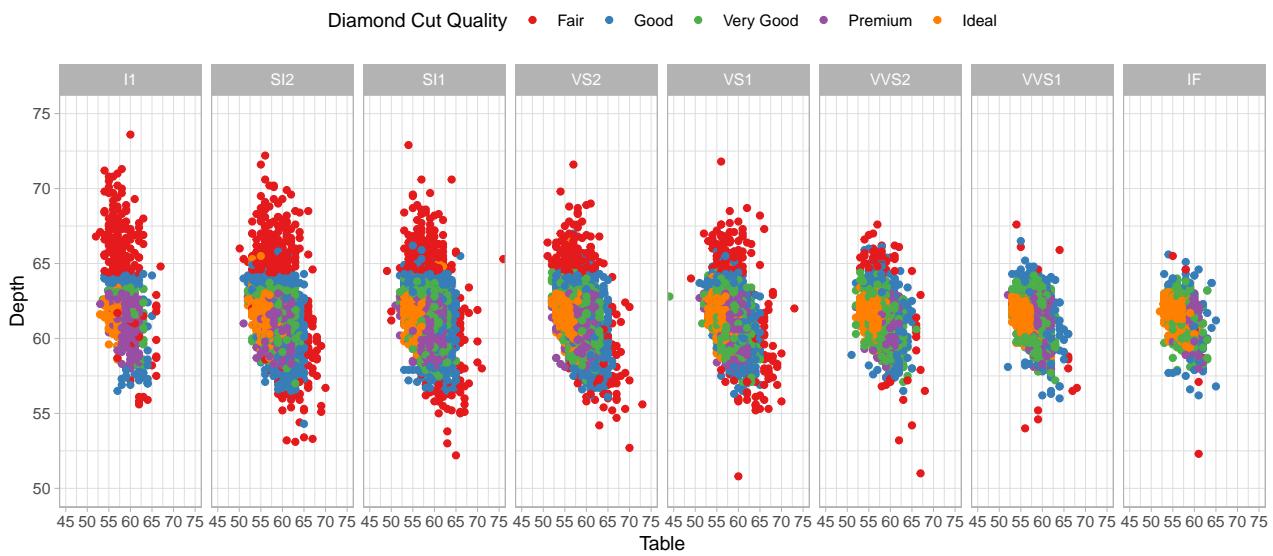
Deze functie verwacht een formule in de vorm van $A \sim B$ waarbij A en B twee categorische variabelen zijn. Voor elke combinatie van waarden van A en B wordt een andere plot geconstrueerd, en deze worden gerangschikt in een rooster waarbij de waarden van A elk een rij vormen en de waarden van B elk een kolom vormen. Vergelijkingen van meer dan 2 variabelen zijn mogelijk met een formule van de vorm $A + B \sim C$. Een vergelijking langs één variabele is mogelijk door een punt te gebruiken in plaats van een variabelenaam, d.w.z. $\cdot \sim A$ of $A \sim \cdot$.

In de volgende plot gebruiken we facetten om onze plot opnieuw te tekenen voor elk van de helderheidsniveaus (clarity). Bovendien is de legende bovenaan geplaatst om meer ruimte te creëren.

```
ggplot(diamonds) +
  geom_point(aes(table, depth, col = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45,75), ylim = c(50, 75)) +
  scale_x_continuous(breaks = seq(45,75,5)) +
  scale_y_continuous(breaks = seq(50,75,5)) +
  scale_color_brewer(palette = "Set1", name = "Diamond Cut Quality") +
  facet_grid(. ~ clarity) +
  theme(legend.position = "top")
```

Een alternatief, meestal geschikt voor vergelijkingen langs één variabele, is `facet_wrap`. In plaats van één rij te maken (zoals `facet_grid` doet), zal het de plots ordenen in een raster met een opgegeven

Comparing table and depth of diamonds
What combination of table and depth gives good quality diamonds?



aantal kolommen of rijen. Hieronder hebben we ze in 3 kolommen gerangschikt.

```
ggplot(diamonds) +
  geom_point(aes(table, depth, col = cut)) +
  labs(title = "Comparing table and depth of diamonds",
       subtitle = "What combination of table and depth gives good quality diamonds?",
       x = "Table",
       y = "Depth") +
  theme_light() +
  coord_cartesian(xlim = c(45,75), ylim = c(50, 75)) +
  scale_x_continuous(breaks = seq(45,75,5)) +
  scale_y_continuous(breaks = seq(50,75,5)) +
  scale_color_brewer(palette = "Set1", name = "Diamond Cut Quality") +
  facet_wrap(~ clarity, ncol = 3) +
  theme(legend.position = "top")
```

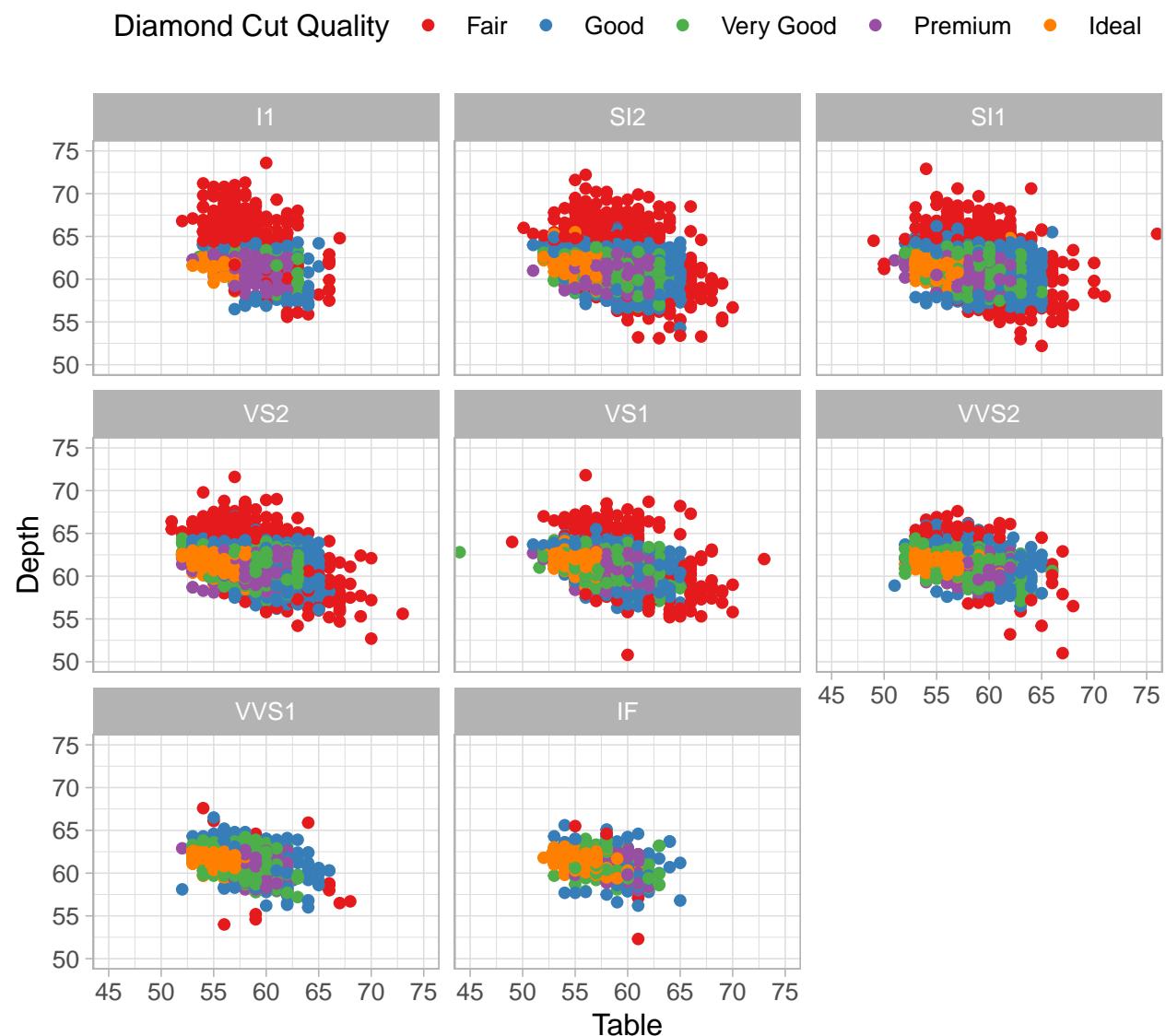
Dit is veel efficienter dan 8 grafieken naast elkaar!

3.5.2 Meerdere layers combineren

Tot nu toe hebben we slechts één geometrische layer tegelijk gebruikt. Het is echter perfect mogelijk om verschillende lagen te combineren. Zo kunnen we bijvoorbeeld het `geom_text` label gebruiken om data labels toe te voegen aan een staafdiagram. `Geom_text` is een geometrische laag die we inderdaad kunnen gebruiken om tekst in een

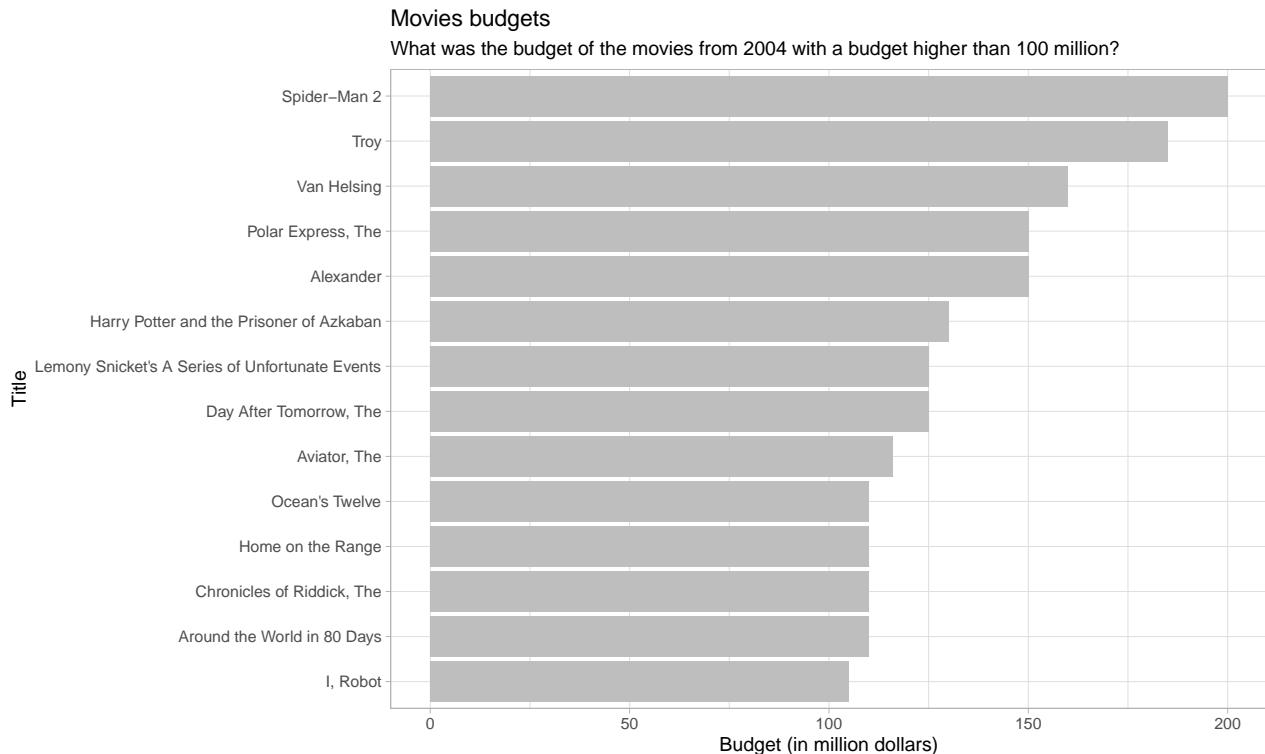
Comparing table and depth of diamonds

What combination of table and depth gives good quality diamonds?



grafiek te zetten. We hebben geom_text nog niet eerder gezien, maar de werking ervan zal eenvoudig zijn, gebaseerd op alles wat we al weten. We bouwen verder op een vorige grafiek, die we een iets fraaier uiterlijk hebben gegeven.

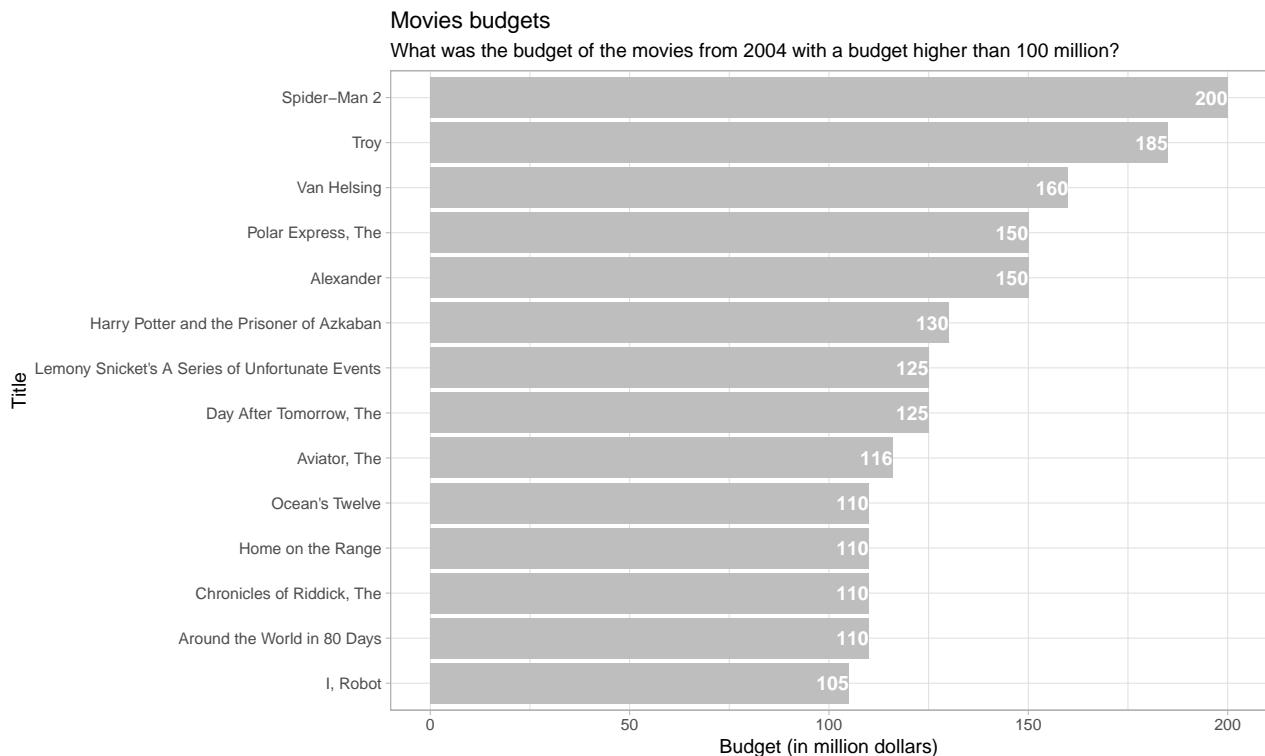
```
filter(movies, year == 2004, budget > 100000000) %>%
  ggplot() +
  geom_col(aes(reorder(title, budget), budget/1000000), fill = "grey") +
  coord_flip() +
  labs(title = "Movies budgets",
       subtitle = "What was the budget of the movies from 2004 with a budget higher than 100 million?",
       x = "Title",
       y = "Budget (in million dollars)") +
  theme_light()
```



Merk op dat we de y-waarde veranderd hebben in budget/1000000, zodat de waarden in miljoenen zijn. Verder is het belangrijk op te merken dat, aangezien we coord_flip hebben gebruikt, onze labels in labs ook zijn verwisseld. Dus verschijnt het x-label op de y-as en het y-label op de x-as. Dit is precies wat we zouden willen, omdat het verwijderen van coord_flip in de toekomst de labels niet in de war zal sturen.

We willen nu het exacte aantal miljoenen boven op de staven zetten. Om dit te doen, voegen we geom_text toe, en geven het dezelfde mapping voor x en y als de geom_col laag. Verder voegen we een mapping toe voor het label, d.w.z. de tekst die moet worden weergegeven. We geven de tekst een vet lettertype en een witte kleur. Tenslotte zorgt de instelling hjust op 1 ervoor dat de tekstlabels horizontaal rechts worden uitgelijnd. Dit zorgt ervoor dat de tekst volledig binnen de balken blijft, en er niet uit valt.

```
filter(movies, year == 2004, budget > 100000000) %>%
  ggplot() +
  geom_col(aes(reorder(title, budget), budget/1000000), fill = "grey") +
  geom_text(aes(reorder(title, budget), budget/1000000,
                label = budget/1000000), color = "white", fontface = "bold", hjust = 1) +
  coord_flip() +
  labs(title = "Movies budgets",
       subtitle = "What was the budget of the movies from 2004 with a budget higher than 100 million?",
       x = "Title",
       y = "Budget (in million dollars)") +
  theme_light()
```



De toevoeging van de tweede geom layer lijkt een beetje omslachtig, omdat we de mapping voor x en y moesten herhalen, wat nog erger

werd omdat er de reorder functie en de deling door een miljoen bij betrokken waren. Er is toch wel een betere manier om dit te doen? We noemen het aesthetics-overerving

3.5.3 Aes-overerving

Aes-erfelijkheid, of overerving van de aes-mapping, betekent dat elk van de geom layers die wordt toegevoegd aan een ggplot functie de mapping *erft* die is gespecificeerd in die ggplot functie-aanroep. Zoals je je misschien nog herinnert van het begin, kan een aes-mapping in zowel geom layers worden geplaatst als in ggplot zelf. We leren nu dat er een klein maar belangrijk verschil is.

Wanneer verschillende layers (een deel van) een mapping gemeen hebben, is het het beste om dit deel te verplaatsen naar de ggplot aanroep. Op die manier hoeft je dit niet te herhalen in de layers die het gebruiken. En, als een van de lagen deze mapping niet gebruikt, kunt je deze eenvoudig overschrijven door een nieuwe mapping op te geven in die laag. Laten we eens kijken naar een voorbeeld.

De vorige plot die we maakten kan eenvoudiger als volgt gemaakt worden:

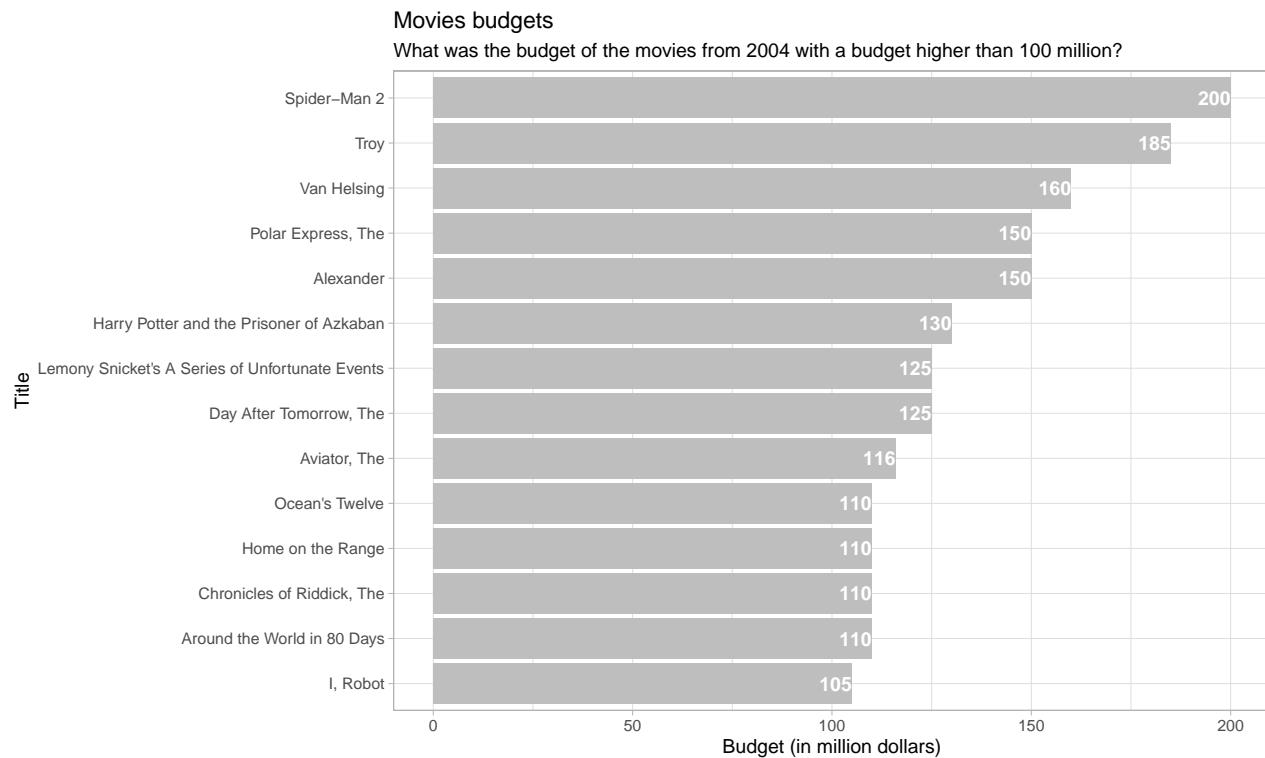
```
filter(movies, year == 2004, budget > 100000000) %>%
  ggplot(aes(reorder(title, budget), budget/1000000)) +
  geom_col(fill = "grey") +
  geom_text(aes(label = budget/1000000), color = "white", fontface = "bold", hjust = 1) +
  coord_flip() +
  labs(title = "Movies budgets",
       subtitle = "What was the budget of the movies from 2004 with a budget higher than 100 million?",
       x = "Title",
       y = "Budget (in million dollars)") +
  theme_light()
```

Door de mapping voor x en y naar ggplot te verplaatsen, is er geen mapping nodig in geom_col, en enkel een mapping voor label in geom_text. Beide geom layers erven het andere deel van de mapping van de ggplot functie-aanroep. Echt, dit is veel efficiënter!

3.6 Background material

Je beheerst nu al heel wat van het plotten met ggplot2. Je hebt beiden geleerd hoe je verschillende geometrische lagen kunt gebruiken om gegevens weer te geven, hoe je ze kunt combineren, hoe je facetten kunt gebruiken, hoe je je code efficiënter kunt maken met aes-overerving, en last but not least, hoe je je plot een mooi uiterlijk kunt geven.

Gefeliciteerd!



Als je graag nog meer wilt weten, kan je het volgende achtergrondmateriaal bekijken:

- The ggplot Cheat Sheet, which provides an overview of all basic functionality in the ggplot2 package.
- The ggplot documentation
- An even more comprehensive ggplot2 tutorial

4

[Lecture notes] Descriptieve statistieken

4.1 Beschrijvende statistieken versus exploratieve plots

- Plots zijn vooral sterk om patronen in de data te visualiseren.
- Plots zijn minder geschikt om de ‘sterkte’ of ‘grootte’ van een patroon uit te drukken.
- Beschrijvende statistieken laten dit wel toe aangezien aspecten van de patronen in een exploratieve plot in exacte getallen worden gegoten.
- Er kunnen hoofdzakelijk 3 soorten beschrijvende statistieken worden onderscheiden:
 - Centrummaten
 - Spreidingsmaten
 - Associatiematen
- Centrummaten en spreidingsmaten zijn univariate statistieken en hebben als doel de verdeling van 1 variabele data samen te vatten in 2 cijfers.
- Associatiematen zijn typisch bivariate statistieken en hebben als doel de samenhang tussen twee variabelen samen te vatten.

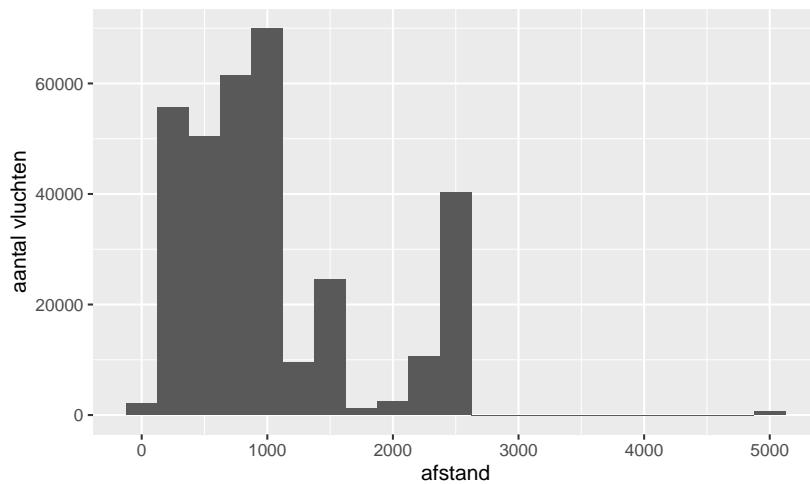
4.2 Notatie

- n : aantal observaties.
- X, Y : variabelen.
- x_i, y_i : de waarden voor variabelen X en Y voor observatie i .
- $x_{(i)}$: de i -de waarde voor X na rangschikking van klein naar groot.

4.3 Data

4.4 Univariate statistieken

4.4.1 Continue variabele



Centrummaten

- Modus
 - Vaak minder bruikbaar bij een continue variabelen omdat iedere waarde zeer weinig voorkomt. Bijgevolg zijn er vaak zeer veel modi met telkens slechts enkele observaties.
- Mediaan
 - De middelste waarde na rangschikking van de gegevens.
 - In geval van een oneven aantal observaties, komt dit overeen met $x_{\frac{(n+1)}{2}}$.
 - In geval van een even aantal observaties zijn er twee ‘middelste’ observaties en is de mediaan gelijk aan $\frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1})$
 - De mediaan is robuust tegen uitschieters.
- (Rekenkundig) Gemiddelde
 - $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$
 - Het gemiddelde is gevoelig voor uitschieters.
 - Dit is de centrummaat die mensen intuïtief selecteren indien mogelijk.

Table 4.1: Afstand (centrummaten)

variabele	gemiddelde	mediaan
afstand	1026.98	820

Spreidingsmaten

- Kwantilen

- Bereik

- Dit is het verschil tussen de grootste en kleinste waarde.
- Zeer gevoelig voor uitschieters.
- Is slechts gebaseerd op 2 observaties en bevat dus weinig informatie. Hiermee bedoelen we dat de spreiding van 2 variabelen sterk kan verschillen terwijl ze toch hetzelfde bereik hebben.

- Interkwartielafstand (IQR)

- Dit is het verschil tussen Q₇₅ en Q₂₅.
- Zelfde principe als het bereik, maar minder gevoelig voor uitschieters.
- IQR is ook slechts gebaseerd op 2 observaties.

- Gemiddelde absolute afwijking (average absolute deviation)

- Dit is de gemiddelde afwijking ten opzichte van het gemiddelde over alle observaties.
- $\frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}|$.

- Variantie

- $s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$.
- Vergelijkbaar met gemiddelde absolute afwijking, maar nu wordt het kwadraat gebruikt om te voorkomen dat de verschillen ten opzichte van het gemiddelde elkaar opheffen.
- Vanuit analytisch standpunt is deze spreidingsmaat interessanter (geen absolute waardes, waardoor afgeleiden bijvoorbeeld eenvoudiger worden om te berekenen).
- Wel gevoelig voor uitschieters en door het kwadraat wordt het effect van deze uitschieters ook nog eens vergroot.
- De wortel van de variantie wordt de standaardafwijking genoemd. De standaardafwijking heeft het voordeel dat het dezelfde eenheid uitgedrukt wordt als de oorspronkelijke data.

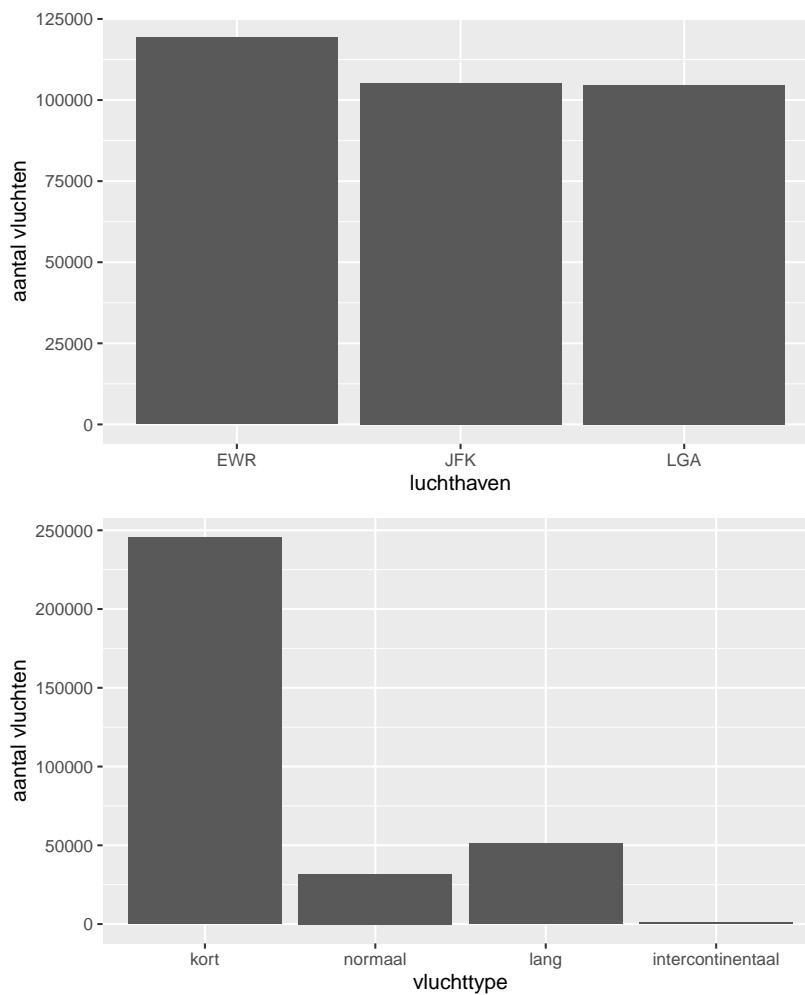
- Median Absolute Deviation (MAD)

- Dit is de middelste afwijking ten opzichte van de mediaan over alle observaties.
- $MAD = \text{median}(|X_i - \text{median}(X)|)$.
- Deze maatstaf is robuster tegen outliers.

Table 4.2: Afstand (spreidingsmaten)

variabele	minimum	Q25	Q50	Q75	maximum	bereik	IQR	var	sd
afstand	17	502	820	1372	4983	4966	870	542630.2	736.6344

4.4.2 Categorische variabele



Frequentietabel

- De absolute frequentie f geeft aan hoe vaak een waarde voorkomt.
- De relatieve frequentie f/n geeft aan welk aandeel deze frequentie heeft in het totaal aantal elementen n .
- De cumulatieve frequentie $F_n(x)$ van een bepaalde waarde x geeft aan hoeveel observaties kleiner zijn dan of gelijk zijn aan x .
- De cumulatieve relatieve frequentie $F_n(x)/n$ van een bepaalde waarde x geeft aan hoeveel percent van de observaties kleiner zijn dan of gelijk zijn aan x .
- Een frequentietabel laat voor alle mogelijke waarden van een categorische variabele de absolute en relatieve frequentie zien (zowel normaal als cumulatief).
- Een frequentietabel laat zien waar een bepaalde waarde zich precies in de verdeling bevindt en hoe uitzonderlijk het is een specifieke

waarde in de data te zien (of een waarde groter/kleiner dan) .

luchthaven	freq	rel_freq	cum_freq	cum_rel_freq
EWR	119282	0.36	119282	0.36
JFK	105230	0.32	224512	0.68
LGA	104662	0.32	329174	1.00

Table 4.3: Aantal vluchten per luchthaven

Centrummaten

- Modus
 - Meest voorkomende waarde.
 - Enige centrummaat voor nominale variabele.
 - Ook bruikbaar voor ordinale variabele.
 - Een variabele kan meerdere modi hebben.
 - De modus is robuust tegen uitschieters.
 - De modus kan je aflezen als de eerste rij in een frequentietabel als je deze ordent van de meest voorkomende tot de minst voorkomende waarde.
- Mediaan
 - De middelste waarde na rangschikken van de gegevens.
 - Voor ordinale variabelen definiëren we de mediaan aan de hand van de relatieve cumulatieve frequentie. De mediaan is de kleinste waarde waar 50% van de observaties kleiner dan of gelijk aan is.
 - De mediaan is robuust tegen uitschieters.

vluchtype	freq	rel_freq	cum_freq	cum_rel_freq
kort	245666	0.75	245666	0.75
normaal	31813	0.10	277479	0.85
lang	50980	0.15	328459	1.00
intercontinentaal	715	0.00	329174	1.00

Table 4.4: Aantal vluchten per vluchtype

variabele	mediaan
vluchtype	kort

Table 4.5: Centrummaten voor vluchtype

Spreidingsmaten

- Kwantielen.

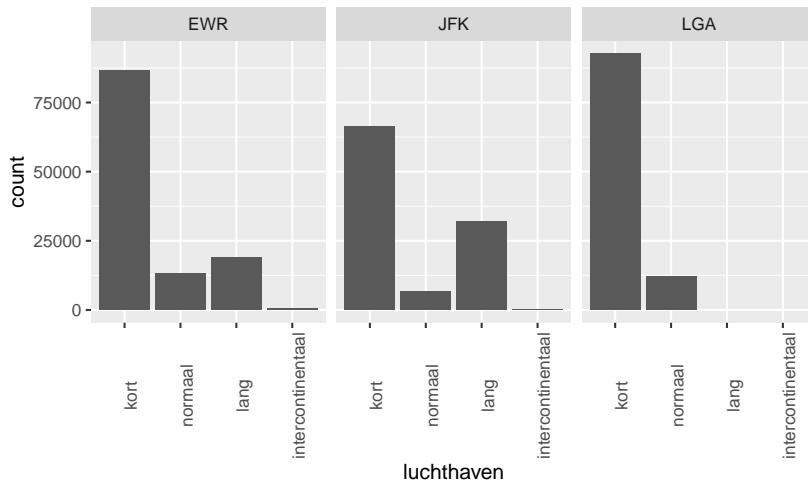
- Kwantielen (of percentielen) zijn gebaseerd op de cumulatieve relatieve frequentie.
- Het $p\%$ kwantiel is de kleinste waarde waar $p\%$ van de observaties kleiner dan of gelijk aan is.
- Het 50% kwantiel komt overeen met de mediaan.
- Veel voorkomende kwantielen om de spreiding van de data weer te geven zijn het 25% en 75% kwantiel.

Table 4.6: Kwantielen voor vluchtype

variabele	Q25	Q50	Q75
vluchtype	kort	kort	normaal

4.5 Bivariate statistieken

4.5.1 Categorisch versus Categorisch



Univariate statistieken per categoriewaarde

- Je toont de relevante centrum- en spreidingsmaten voor de afhankelijke categorische variabele per waarde van de onafhankelijke categorische variabele. Dit is enkel mogelijk indien de afhankelijke categorische variabele ordinaal is, waarbij je minimum, mediaan, maximum en kwantielen kan berekenen.

Contingentietabellen

Een andere mogelijkheid is het maken van 2-dimensionale frequentietabellen, ookal contingentietabellen genoemd. Meer info hierover lees je in de tutorial.

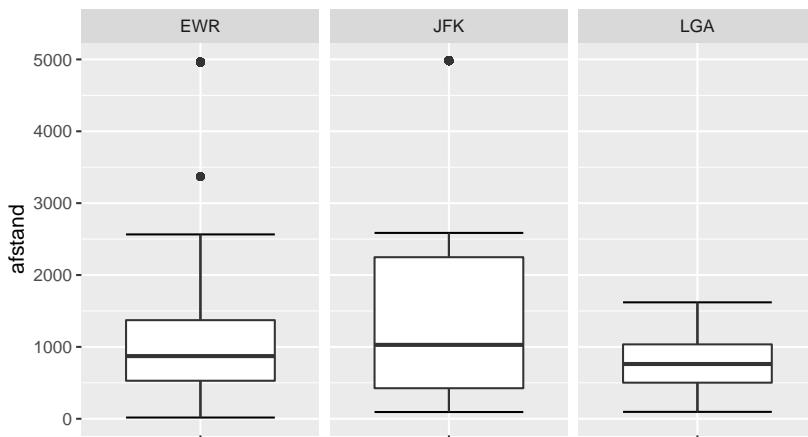
luchthaven	variabele	mediaan
EWR	vluchtttype	kort
JFK	vluchtttype	kort
LGA	vluchtttype	kort

luchthaven	variabele	Q25	Q50	Q75
EWR	vluchtttype	kort	kort	normaal
JFK	vluchtttype	kort	kort	lang
LGA	vluchtttype	kort	kort	kort

Table 4.7: Centrummaten voor vluchtttype-luchthaven

Table 4.8: Kwantielnen voor vluchtttype-luchthaven

4.5.2 Categorisch versus Continu



Univariate statistieken per categoriewaarde

- Je toont de relevante centrum- en spreidingsmaten voor de afhankelijke continue variabele per waarde van de onafhankelijke categorische variabele.

luchthaven	gemiddelde	mediaan
EWR	1049.58	872
JFK	1247.16	1028
LGA	779.84	762

Table 4.9: Afstand-Luchthaven (centrummaten)

Correlatie

- Enkel toepasbaar als de categorische variabele ordinaal is.
- Pearson's correlatiecoëfficiënt kan je NIET toepassen.

luchthaven	var	min	Q25	Q50	Q75	max	bereik	IQR	sd
EWR	536177.0	17	529	872	1372	4963	4946	843	732.2411
JFK	842460.4	94	425	1028	2248	4983	4889	1823	917.8564
LGA	138132.3	96	502	762	1035	1620	1524	533	371.6615

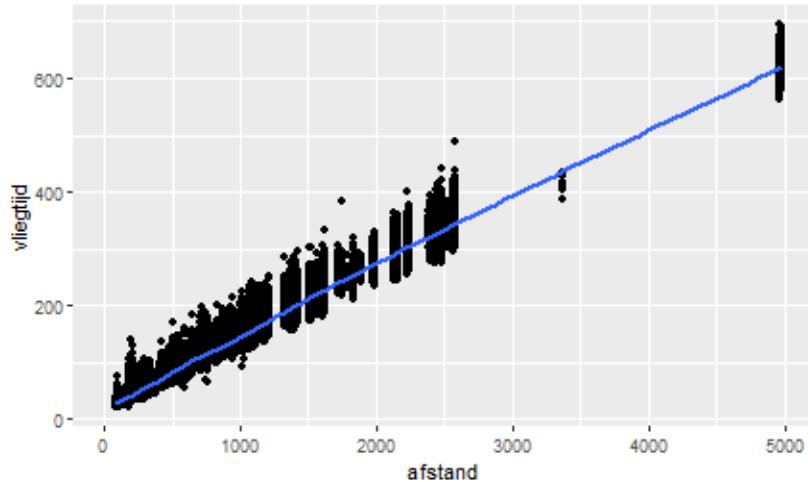
- Spearman rangcorrelatiecoëfficiënt (ρ).
- Kendall's rangcorrelatiecoëfficiënt (τ) kan theoretisch wel toegepast worden, maar is in de praktijk vaak niet haalbaar.

variabelenpaar	spearman
vluchtype-vliegtijd	0.76

Table 4.10: Afstand-Luchthaven (spreidingsmaten)

Table 4.11: Correlatie tussen vluchtype en vliegtijd

4.5.3 Continu versus Continu



Correlatie

- Covariantie
 - $cov(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$.
 - Bij een positieve associatie tussen twee variabelen zal de covariante positief zijn.
 - Bij een negatieve associatie tussen twee variabelen zal de covariante negatief zijn.
 - De covariantie is echter afhankelijk van de maateenheid van de variabelen, waardoor ze weinig bruikbaar is om de sterkte van de associatie weer te geven.
- Pearson correlatiecoëfficiënt
 - Herschaalt de covariantie naar de schaal $[-1, 1]$

- Laat toe om de sterkte van een associatie te evalueren.
- $r(x, y) = \frac{\text{cov}(x, y)}{s_x s_y}$
- $r(x, y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$
- Meet **lineaire** associatie tussen 2 variabelen.
- Twee variabelen kunnen positief geassocieerd zijn, maar in een niet-lineaire wijze, waardoor de correlatiecoëfficiënt naar nul gaat.
- Meest gebruikelijke correlatiecoëfficiënt voor continue variabelen.
- Daarom best altijd samen met een puntenwolk bekijken.

- Spearman's rangcorrelatiecoëfficiënt.
 - Zelfde principe als Pearson's, maar dan gebaseerd op de rangorde van de waarden in plaats van de waarden zelf.
 - r_i : rangorde van waarde x_i . Bijvoorbeeld $r_i = 4$ betekent dat de waarde x_i de vierde kleinste waarde is.
 - s_i : rangorde van waarde y_i .
 - $\rho(x, y) = \frac{\sum_{i=1}^n (r_i - \bar{r})(s_i - \bar{s})}{\sqrt{\sum_{i=1}^n (r_i - \bar{r})^2} \sqrt{\sum_{i=1}^n (s_i - \bar{s})^2}}$
 - Meet associatie tussen 2 variabelen, dus niet specifiek lineaire associatie.

- Kendall's correlatiecoëfficiënt
 - Ook wel Kendall's tau genoemd.
 - De methode is gebaseerd door alle mogelijke observatieparen (x_i, y_i) en (x_j, y_j) te bestuderen.
 - Net als Spearman's aanpak gebaseerd op rangorde (r_i, s_i) en niet de feitelijke waarden.
 - Indien $r_i > r_j$ en $s_i > s_j$ (of $r_i < r_j$ en $s_i < s_j$) dan zijn observaties i en j concordant.
 - Indien $r_i > r_j$ en $s_i < s_j$ (of $r_i < r_j$ en $s_i > s_j$) dan zijn observaties i en j discordant.
 - Notatie: C en D zijn respectievelijk het aantal concordante en discordante paren.
 - $\tau = \frac{C-D}{\frac{1}{2}n(n-1)}$
 - Net als Spearman's correlatiecoëfficiënt, focust Kendall's tau op de associatie (positief of negatief) en niet specifiek op lineaire associatie.
 - Het nadeel van Kendall's tau is dat je alle observatieparen moet bestuderen en het aantal kan snel exploderen bij veel observaties. Immers het aantal paren is $\frac{n!}{2!(n-2)!}$. Hierdoor kan je Kendall in de praktijk niet gebruiken als je veel observaties hebt.

variabelenpaar	pearson	spearman
afstand-vliegtijd	0.99	0.98

Table 4.12: Correlatie tussen afstand en vliegtijd

Vergelijking correlatiecoëfficiënten

- Rangcorrelatiecoëfficiënten meten associatie, terwijl Pearson correlatiecoëfficiënt **lineaire** associatie meet!

x	y
1	0.0
2	4.0
3	5.0
4	5.5
5	7.0
6	15.0
7	15.6
8	16.0
9	50.0
10	1000.0

Table 4.13: Fictieve dataset

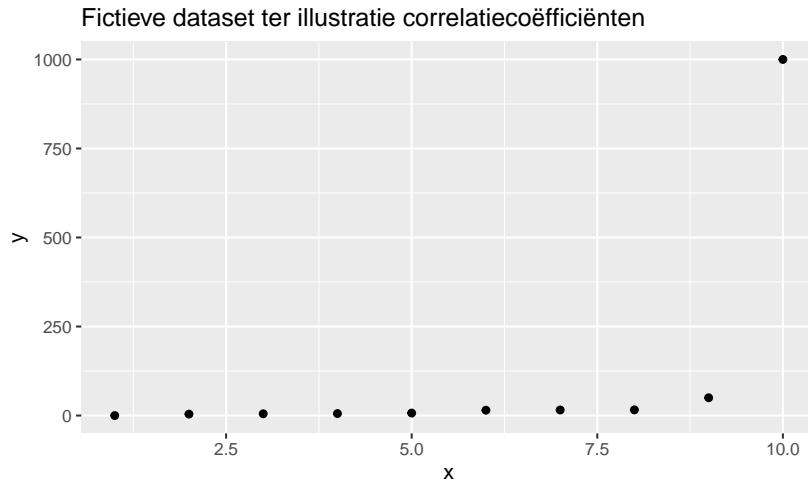


Table 4.14: Correlatiecoëfficiënten fictieve dataset

variabelenpaar	pearson	spearman	kendall
x-y	0.55	1	1

Referenties

- Tekst Beleidsstatistiek: Hoofdstukken 1 en 2 en secties 4.2 en 4.3 (Blackboard)

2. Spearman's rangcorrelatiecoëfficiënt
3. Kendall's rangcorrelatiecoëfficiënt
4. Spearman versus Kendall's correlatiecoëfficiënt

5

[Tutorial] Descriptieve statistieken

5.1 Voor je begint

Voordat je aan deze tutorial begint, moet je eerst de pakketten `dplyr`, `tidyr`, `ggcorrplot`, en `ggplot2` geïnstalleerd hebben, **als je dat nog niet gedaan hebt**. Deze kunt u laden met de `library` functie. Als je sommige nog moet installeren, gebruik dan eerst `install.packages`.

```
library(dplyr)
library(tidyr)
library(ggplot2)
```

Daarnaast wil je misschien ook de volgende packages gebruiken. Ze zullen nuttig blijken op een bepaald punt in deze tutorial, maar zijn niet strikt noodzakelijk.

```
library(pander)
library(forcats)
```

Deze tutorial maakt gebruik van de `mpg` dataset, die informatie bevat over 234 verschillende auto's. Het `mpg.RDS` bestand wordt met deze tutorial meegeleverd.

```
mpg <- readRDS("mpg.RDS")
```

De dataset bevat de volgende variabelen:¹

Variabele	Beschrijving
manufacturer	De fabrikant
model	De naam van het model
displ	Motorinhoud, in liters
year	bouwjaar
cyl	Aantal cilinders
trans	Type transmissie (<i>koppeling</i>)

¹ Hoewel het bepaalde analyses interessanter maakt als je bekend bent met de betekenis van de verschillende variabelen, is geen specifieke kennis van auto's vereist om deze tutorial te voltooien.

Variabele	Beschrijving
drv	f = voorwielaandrijving, r = achterwielaandrijving, 4 = 4wd
cty	Aantal mijl per gallon brandstof in stadsomgeving
hwy	Aantal mijl per gallon brandstof op snelweg
fl	Brandstoftype (c,d,e,p,r)
class	Type auto (2seater, compact, midsize, minivan, pickup, subcompact, suv)

5.2 Introductie

Het doel van deze tutorial is om zowel univariate als bivariate analyses uit te voeren met het `dplyr` package. Verder zullen in een beperkt aantal gevallen de `tidyverse` en `ggcorrplot` packages worden gebruikt voor extra ondersteuning.

De univariate analyse wordt uitgevoerd voor zowel categorische als continue variabelen. De bivariate analyse wordt uitgevoerd voor elk van de volgende paren: continu-categorisch, continu-continu en categorisch-categorisch. Naast een *numerieke* analyse met behulp van functies uit de eerder genoemde pakketten, zullen de analyses vergezeld gaan van passende grafieken gemaakt met `ggplot2`. Een basiskennis van `ggplot2` is vereist.

De tutorial is als volgt opgebouwd:

- Enkele algemene, nuttige `dplyr` functies
- Univariate analyse van een continue variabele
- Bivariate analyse van een continue variabele vs. een categorische variabele
- Univariate analyse van een categorische variabele
- Bivariate analyse van een continue variabele vs. een andere continue variabele
- Bivariate analyse van een categorische variabele vs. een andere categorische variabele

5.3 Nuttige `dplyr` functies

Voordat we overgaan tot het analyseren van onze dataset, zijn er een paar handige hulpfuncties in `dplyr` die we kunnen gebruiken om onze gegevens te verwerken en te analyseren. De eerste is `as_tibble`.

5.3.1 Tibbles

Gegeven een `data.frame` `df`, zal `as_tibble(df)` het in een *tibble* veranderen. Een tibble is een speciaal soort `data.frame` dat gebruikt wordt door `dplyr` en andere pakketten van het tidyverse.² Wanneer een `data.frame` wordt omgezet in een tibble zal de klasse veranderen.

```
class(mpg)

## [1] "data.frame"

mpg <- as_tibble(mpg)
class(mpg)

## [1] "tbl_df"     "tbl"        "data.frame"
```

Je kunt zien dat het `mpg` object nu drie verschillende `class` labels heeft. Het is nog steeds een `data.frame`, maar een speciaal soort `data.frame`, namelijk een tibble `data.frame`.

Het verschil tussen een gewoon `data.frame` en een tibble `data.frame` is het meest merkbaar bij het afdrukken van (grote) `data.frames`. Probeer maar eens de dataset met de volgende twee regels in de console af te drukken.

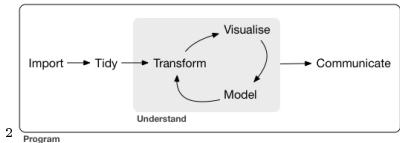
```
#Deze regels worden hier niet uitgevoerd
#je kunt ze proberen in de console.
as.data.frame(mpg)

mpg
```

Merk je het verschil? Bij het afdrukken van een gewoon `data.frame`, zal een overvloed aan observaties worden afgedrukt in de console. Het resultaat is dat je terug naar boven moet scrollen om de variabelenamen te zien. Erger nog, wanneer de kolommen niet op een enkele pagina passen, zal elke observatie verspreid worden over verschillende lijnen, waardoor de uitvoer onleesbaar wordt.³

Wanneer echter een tibble `data.frame` wordt afgedrukt, worden standaard alleen de eerste 10 rijen afgedrukt, en variabelen die niet binnen de breedte van de pagina of console passen worden verborgen. Dit maakt de afgedrukte dataset veel leesbaarder en onze console minder rommelig.

Het nadeel van deze aanpak is dat je soms meer waarnemingen of meer variabelen wil zien, zonder je tibble terug in een `data.frame` te veranderen. Je kunt dit oplossen door expliciet de `print` functie te gebruiken en de argumenten `n` en `width` in te stellen. De volgende regel drukt alle rijen en kolommen af, door beide argumenten op `Inf` te zetten, wat staat voor oneindig.⁴ Je kunt ook andere waarden gebruiken om ander aantal kolommen en rijen te printen. Later



Het *tidyverse* is een set van pakketten voor data science die in harmonie werken omdat ze gemeenschappelijke data representaties en API ontwerp delen. Het *tidyverse*-pakket is ontworpen om het gemakkelijk te maken om kernpakketten van de tidyverse te installeren en te laden met een enkel commando. De tidyverse bevat pakketten zoals: `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `hms`, `stringr`, `lubridate`, `forcats`, `jsonlite`, `readxl`, `broom`, en anderen. Hadley Wickham kan beschouwd worden als de grondlegger van het tidyverse. De beste plaats om deze pakketten te leren kennen is door dit boek te lezen: *R for Data Science*, geschreven door Hadley en Garret Grolemund.

³ Omdat we `mpg` daarnet in een tibble `data.frame` hebben veranderd, kunnen we het alleen afdrukken alsof het een normaal `data.frame` is door de `as.data.frame` functie te gebruiken. We updaten het object echter niet, omdat we het niet opslaan met de `<-`.

⁴ Let op de hoofdletter I

zullen we zien hoe we specifieke rijen en kolommen kunnen printen. Merk daarnaast op dat je altijd `View()` kan gebruiken om de volledige dataset te tonen in een apart Rstudio panel.

```
#Deze regel wordt hier niet uitgevoerd - je kunt het in de console proberen.
print(mpg, n = Inf, width = Inf)
```

Behalve voor het printen, laten tibbles enkele eenvoudiger manipulaties toe vergeleken met een normaal `data.frame` als je het pakket `tibble` installeert. Dit valt echter buiten het bereik van deze tutorial. Verder zal je zien dat de uitvoer van `dplyr` functies die in de rest van deze tutorial worden besproken automatisch tibbles zijn. Dit betekent dat je vaak niet expliciet `as_tibble` hoeft te gebruiken. Voor nu is het dus voldoende om het verschil te begrijpen.

5.3.2 Glimpse

Een tweede handige functie is de `glimpse` functie. Deze functie is het `dplyr`-alternatief voor de bekende `str`-functie voor base-R, en is dus nuttig voor een eerste inspectie van de dataset die vorhanden is. Het zal een iets andere output geven dan `str`, en wordt door sommigen als netter ervaren.

```
str(mpg)

## # tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
## $ manufacturer: Factor w/ 15 levels "audi","chevrolet",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ model      : Factor w/ 38 levels "4runner 4wd",...: 2 2 2 2 2 2 3 3 3 ...
## $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
## $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
## $ cyl         : Ord.factor w/ 4 levels "4"<"5"<"6"<"8": 1 1 1 1 3 3 3 1 1 1 ...
## $ trans       : Factor w/ 10 levels "auto(av)","auto(13)",...: 4 9 10 1 4 9 1 9 4 10 ...
## $ drv         : Factor w/ 3 levels "4","f","r": 2 2 2 2 2 2 2 1 1 1 ...
## $ cty         : int [1:234] 18 21 20 21 16 18 18 16 20 ...
## $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
## $ fl          : Factor w/ 5 levels "c","d","e","p",...: 4 4 4 4 4 4 4 4 4 4 ...
## $ class       : Factor w/ 7 levels "2seater","compact",...: 2 2 2 2 2 2 2 2 2 2 ...
```

```
glimpse(mpg)

## #> #> Rows: 234
## #> #> Columns: 11
## #> #> $ manufacturer <fct> audi, audi, audi, audi, audi, audi, audi, audi, audi, ...
## #> #> $ model      <fct> a4, a4, a4, a4, a4, a4 quattro, a4 quattro, a4...
## #> #> $ displ       <dbl> 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, ...
## #> #> $ year        <int> 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, ...
## #> #> $ cyl         <ord> 4, 4, 4, 4, 6, 6, 4, 4, 4, 6, 6, 6, 6, 8, ...
```

```
## $ trans      <fct> auto(15), manual(m5), manual(m6), auto(av), auto(15), ...
## $ drv        <fct> f, f, f, f, f, f, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, ...
## $ cty        <int> 18, 21, 20, 21, 16, 18, 18, 16, 20, 19, 15, 17, 17...
## $ hwy        <int> 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25...
## $ fl         <fct> p, ...
## $ class      <fct> compact, compact, compact, compact, compact, ...
```

5.3.3 %>%

Een andere handige functie is een heel speciale, en heet *het piping-symbool*.⁵ Het piping-symbool bestaat uit een groter-dan-teken, voorafgegaan en gevolgd door een %-teken.

Het piping-symbool kan worden gebruikt om verschillende commandos *met elkaar te verbinden*. Je kunt het vergelijken met leidingen in een waterleidingnet. Binnen een waterzuiveringsstation vervoeren leidingen het water van het ene zuiveringsstation naar het volgende en uiteindelijk naar de huishoudens. Hier gebruiken we het piping-symbool om onze gegevens van de ene manipulatie naar de volgende te brengen. Beschouw dit zeer eenvoudige voorbeeld:

```
#Deze regel wordt hier niet uitgevoerd
#je kunt het in de console proberen.

head(mpg)
```

We kunnen deze regel parafraseren als: “Neem het hoofd (d.w.z. de eerste 6 rijen) van de dataset mpg.”

Wanneer we het piping-symbool gebruiken, kan het eerste argument van een functie *voor* de functie-aanroep worden geplaatst in plaats van erbinnen. De laatste regel code is daarom gelijk aan de volgende regel.⁶

```
#Deze regel wordt hier niet uitgevoerd
#je kunt het in de console proberen.

mpg %>% head()
```

We kunnen nu zeggen: “We nemen de dataset mpg en nemen dan het hoofd ervan.” Deze zin lijkt natuurlijker, want hij is heel gemakkelijk uit te breiden met de verdere stappen die je gaat nemen.

Zoals leidingen water van de ene plaats naar de volgende brengen in een waterleidingnet, brengt het piping-symbool onze gegevens van de ene plaats naar de volgende, in dit geval de head-functie. Op dit moment lijkt het misschien belachelijk om dit te doen, maar zoals we heel snel zullen zien, komt dit symbool heel goed van pas.

Als ander voorbeeld: merk op dat de volgende twee hypothetische verklaringen zijn gelijkwaardig



⁵

Het piping-symbool werd voor het eerst geïntroduceerd in het pakket `magrittr`, genoemd naar de Belgische surrealistische kunstenaar René Magritte, bekend van zijn schilderij *The Treachery of Images*, oftewel *Ceci n'est pas un pipe*.

⁶ Je kunt zelfs de haakjes van de functies weglaten als er geen argumenten meer zijn. Zo zou `mpg %>% head` prima werken. Verwar dit echter niet met het toevoegen van lagen aan een `ggplot` aanroep. Hier zou je de haakjes van lege functie-aanroepen moeten behouden. Voeg bijvoorbeeld niet `coord_flip` toe aan een `ggplot` object, maar voeg `coord_flip()` toe. Wees je bewust van dit verschil. Uit veiligheidsoverwegingen is het aan te raden consistent de haakjes te laten staan.

```
f(x,y,z)
x %>% f(y,z)
```

En als we twee functies hebben, bv. `f` en `g` die we genest hebben aangeroepen, d.w.z. de een binnen de ander, dan kunnen we het volgende doen.

```
f(g(x,y),z)
# Breng het eerste argument van f naar voren
g(x,y) %>% f(z)
# Breng het eerste argument van g naar voren
x %>% g(y) %>% f(z)
```

Dus, dit commando neemt `x`, voert functie `g` uit met argument `y` en dan wordt het resultaat gegeven aan functie `f` met tweede argument `z`. Dit is veel gemakkelijker te lezen dan de originele regel, waar we functie `f` uitvoerden op het resultaat van functie `g` op `x` en `y`, en `z` gebruikten als tweede argument voor `f`.

Echter, dat is genoeg wat betreft abstracte concepten. Het is tijd om iets met onze dataset te doen en dit symbool nuttig te gebruiken.

5.4 Univariate analyse van een continue variabele

We beginnen met een uniforme analyse van continue variabelen. Voorbeelden van continue variabelen zijn leeftijd, afstand, snelheid, gewicht, enz. Voor dit soort variabelen kunnen we het centrum en de spreiding meten. Maatstaven voor centrum zijn gemiddelde en mediaan. Maatstaven voor spreiding zijn standaardafwijking, kwantielen, min en max, interkwartielafstand en bereik.

Al deze maatstaven hebben één ding gemeen: zij *vatten* een continue variabele *samen* door middel van één getal, één waarde. Dit kan worden uitgevoerd met behulp van de `summarise` functie van `dplyr`.⁷

Als voorbeeld, stel dat we het gemiddelde en de standaardafwijking van de `cty` willen berekenen.

```
summarize(mpg, mean_cty = mean(cty), st_dev_cty = sd(cty))

## # A tibble: 1 x 2
##   mean_cty    st_dev_cty
##       <dbl>      <dbl>
## 1     16.9       4.26
```

Het eerste argument van `summarize` is het data argument. Alle volgende argumenten worden nieuwe kolommen in de resulterende tabellen. `mean_cty` en `st_dev_cty` zijn de namen van de nieuwe kolommen. Deze namen kan je volledig zelf beslissen. De delen na het

⁷ Zowel het Brits-Engelse *summarise* als het Amerikaans-Engelse *summarize* kan worden gebruikt.

=-teken worden de inhoud van de kolommen, d.w.z. het gemiddelde (berekend met de `mean` functie) en de standaardafwijking (berekend met de `sd` functie).

In plaats van namen zonder aanhalingstekens te geven (bijv. `mean_cty`), kan je ook aanhalingstekens toevoegen. Hierdoor kunt u spaties gebruiken. Wanneer je uitvoer definitief is en niet bedoeld is voor verdere manipulatie, zou u dit kunnen doen, om mooie kolomnamen te hebben in je rapport. Bijvoorbeeld `mpg %>% summarize("Gemiddelde cty" = mean(cty))`. In het algemeen kan je spaties echter beter vermijden.

We kunnen de bovenstaande regel herschrijven door het `%>%` symbool te gebruiken. Merk op dat na dit symbool, we *enter* hebben gebruikt, zodat `summarize` op een nieuwe regel begint, met een inspringing. De inspringing wijst erop dat deze regels eigenlijk één statement vormen (d.w.z. de ene regel kan niet worden uitgevoerd zonder de andere).⁸

```
mpg %>%
  summarize(mean_cty = mean(cty), st_dev_cty = sd(cty)) %>%
  pander()
```

mean_cty	st_dev_cty
16.86	4.256

Alle functies die gebruikt worden binnen de `summarize` functie moeten *één* en slechts *één* waarde teruggeven, d.w.z. het gemiddelde, de mediaan, de interkwartielafstand, enz. We noemen deze functies *summary-functies*. Een (onvolledige) lijst van samenvattende functies is hier opgenomen:

- `min`
- `max`
- `mean`
- `median`
- `first` (eerste element van vector)
- `last` (laatste element van vector)
- `n` (aantal waarden in een dataset)⁹
- `nth` (n-de waarde van vector)
- `n_distinct` (aantal verschillende waarden in vecotor)
- `IQR` (interkwartielafstand)
- `var` (variantie)
- `sd` (standaardafwijking)
- `quantile`
- `sum`

⁸ Merk op dat we ook de `pander` functie aan dit statement hebben toegevoegd. Deze functie komt uit het `pander` pakket en is afgeleid van de naam *pandoc*, een gratis, open-source software document converter. Merk op dat de resulterende tabel mooier geformatteerd is dan het resultaat van het vorige statement zonder het gebruik van `pander`. Het dient echter alleen om te gebruiken in een mark-down document als opmaak belangrijk is. Het heeft geen zin te gebruiken worden bij het programmeren in de console of in r-scripts. Je kunt dit statement gewoon negeren voor de rest van deze tutorial.

⁹ WAARSCHUWING: gebruik `n` alleen binnен `summarize`, en telt het aantal observaties in de dataset. In tegenstelling tot alle andere functies hier neemt deze functie geen input aan. Deze functie kan dus ook niet gebruikt worden op normale vectoren, wat voor de andere functies wel geldt.

Merk op dat (behalve `n`) al deze functies ook gebruikt kunnen worden op normale vectoren, dus niet binnen de `summarize` functie. Omgekeerd zijn dit niet de enige functies die binnen `summarize` gebruikt kunnen worden. In het algemeen kunnen alle functies die één enkele waarde teruggeven worden gebruikt. Bovendien kunnen we ook kolommen opnemen waarvan we de waarde handmatig hebben ingesteld. Bijvoorbeeld, we kunnen het laatste voorbeeld bijwerken en opnemen dat we de `cty` variabele als kolom hebben gebruikt. In dit geval hebben we geen summary-functie nodig, maar alleen een waarde, bijvoorbeeld “`cty`”.

```
mpg %>%
  summarize(variable = "cty", mean = mean(cty), st_dev = sd(cty)) %>%
  pander
```

variable	mean	st_dev
cty	16.86	4.256

Sommige overzichtsfuncties hebben extra argumenten nodig, zoals de `nth` functie, die een waarde voor `n` nodig heeft, en eventueel een ordening kan opgeven. De `quantile` functie heeft een `probs` argument nodig, dat staat voor het percentiel. Bijvoorbeeld, om het eerste 10% percentiel te berekenen, stellen we `probs = 0.1` in. Hieronder berekenen we bijvoorbeeld de kwintielen van de `cty` variabele.

```
mpg %>%
  summarize(variable = "cty",
            q0.2 = quantile(cty, 0.2),
            q0.4 = quantile(cty, 0.4),
            q0.6 = quantile(cty, 0.6),
            q0.8 = quantile(cty, 0.8)) %>%
  pander
```

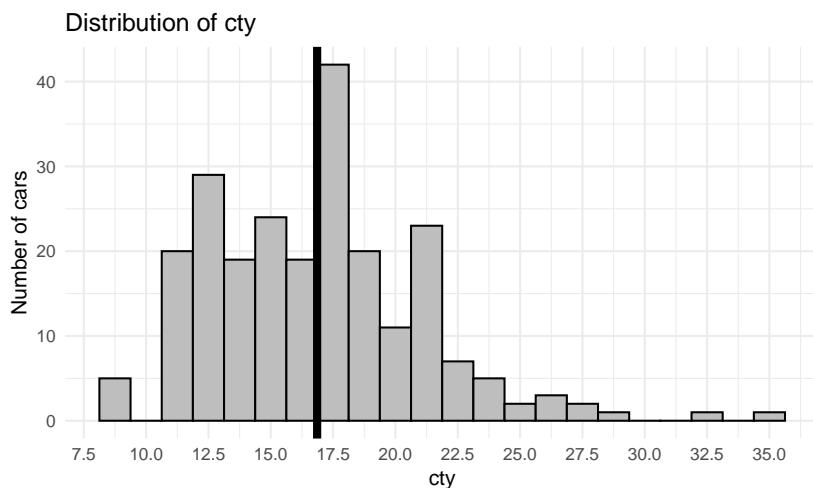
variable	q0.2	q0.4	q0.6	q0.8
cty	13	15	18	20

We zien dat 20% van de `cty` waarden kleiner is dan of gelijk is aan 13, en 20% van de `cty` waarden groter is dan of gelijk is aan 20. In het laatste blok code hebben we elke variabele op een nieuwe regel gezet. Rstudio zal alle regels automatisch laten inspringen, zodat het duidelijk is dat het om argumenten van de `summarize` functie gaat. De `pander` functie komt weer op een nieuwe regel, met dezelfde

inspringing als `summarize`. Geef aandacht aan je opmaak van je code, zodat deze voor anderen alsook jezelf leesbaar is.

Als we onze resultaten willen ondersteunen met een grafiek, kunnen we een box plot of histogram plotten met `ggplot2`. Voor een introductie in `ggplot2`, verwijzen we naar de `ggplot2` tutorial. Hieronder hebben we een histogram uitgezet. We hebben ook een verticale lijn toegevoegd om het gemiddelde van `cty` aan te geven, met `geom_vline`.

```
mpg %>%
  ggplot(aes(cty)) +
  geom_histogram(binwidth = 1.25, color = "black", fill = "grey") +
  geom_vline(xintercept = mean(mpg$cty), lwd = 2) +
  labs(title = "Distribution of cty",
       x = "cty",
       y = "Number of cars") +
  theme_minimal() +
  scale_x_continuous(breaks = seq(7.5, 35, 2.5))
```



Merk op hoe we ook hier het `%>%`-symbool gebruiken om de dataset uit de `ggplot()` functie te halen. In `ggplot()` hoeven we dan enkel nog de aesthetics te definiëren. Maar let op, na `ggplot` gebruik je `+` om nieuwe layers toe te voegen, niet `%>%`!

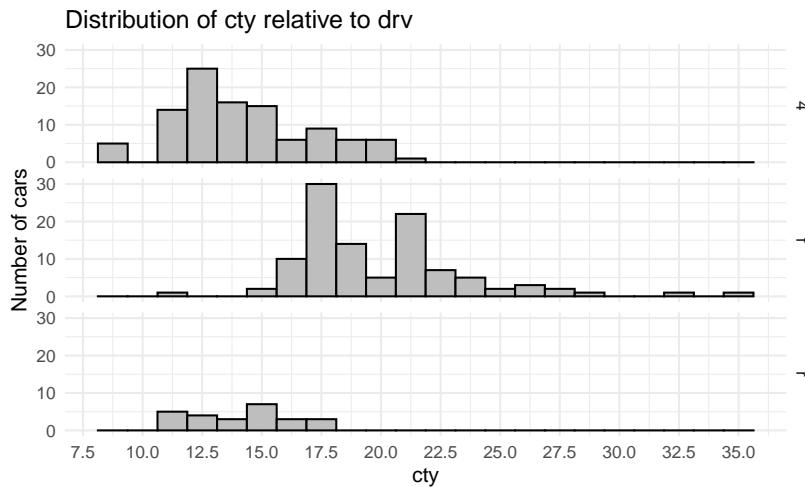
Met de `summarize` functie kunnen we een univariate analyse uitvoeren van de spreiding en centraliteit van elke continue variabele.

Geweldig! Nu is het tijd om een stap verder te gaan en te beginnen met bivariate analyse van continue variabelen in combinatie met categorische variabelen.

5.5 Bivariate analysis of a continuous variable with respect to a categorical variable

Laten we, voor we met onze berekeningen beginnen, eerst grafisch verder gaan. Voordien hebben wij de verdeling van de cty geanalyseerd, zoals blijkt uit het histogram hierboven. Nu willen we deze analyseren voor verschillende soorten aandrijving (b.v. vierwielaandrijving, voorwielaandrijving of achterwielaandrijving), zoals geregistreerd door de variabele `drv`. Grafisch kunnen we verschillende histogrammen plotten voor elk van deze drie categorieën met behulp van facetten.

```
mpg %>%
  ggplot(aes(cty)) +
  geom_histogram(binwidth = 1.25, color = "black", fill = "grey") +
  labs(title = "Distribution of cty relative to drv",
       x = "cty",
       y = "Number of cars") +
  theme_minimal() +
  scale_x_continuous(breaks = seq(7.5, 35, 2.5)) +
  facet_grid(drv ~ .)
```



Het is duidelijk dat er een aantal verschillen zijn tussen deze categorieën. Laten we proberen deze in cijfers uit te drukken. Wat we in feite willen doen is de centrum- en spreidingsmaten berekenen voor elk van deze categorieën, d.w.z. voor elke *groep* van auto's. Daarom introduceren we een nieuwe dplyr-functie, genaamd `group_by`. Het eerste argument van deze functie is nodig voor de gegevens, alle andere argumenten (die normaal altijd categorisch zijn) zullen worden gebruikt om de gegevens te groeperen.

```
group_by(mpg, drv)
```

```

## # A tibble: 234 x 11
## # Groups:   drv [3]
##   manufacturer model     displ  year cyl trans   drv     cty   hwy fl   class
##   <fct>        <fct>     <dbl> <int> <ord> <fct>   <fct> <int> <int> <fct> <fct>
##   1 audi         a4       1.8  1999    4 auto(l~ f      18    29 p   comp~
##   2 audi         a4       1.8  1999    4 manual~ f     21    29 p   comp~
##   3 audi         a4       2    2008    4 manual~ f     20    31 p   comp~
##   4 audi         a4       2    2008    4 auto(a~ f      21    30 p   comp~
##   5 audi         a4       2.8  1999    6 auto(l~ f      16    26 p   comp~
##   6 audi         a4       2.8  1999    6 manual~ f     18    26 p   comp~
##   7 audi         a4       3.1  2008    6 auto(a~ f      18    27 p   comp~
##   8 audi         a4 quat~ 1.8  1999    4 manual~ 4     18    26 p   comp~
##   9 audi         a4 quat~ 1.8  1999    4 auto(l~ 4      16    25 p   comp~
##  10 audi        a4 quat~  2   2008    4 manual~ 4     20    28 p   comp~
## # ... with 224 more rows

```

De uitvoer van deze regel geeft ons gewoon een tibble, zonder opmerkelijke wijzigingen. Maar laat je niet misleiden, want die zijn er wel! Op de tweede geprinte regel lezen we “Groups: *drv* [3]”. Deze tibble is dus gegroepeerd op de variabele *drv*, en er zijn drie verschillende groepen. Laten we, om een voorbeeld te geven, ook *trans* als groep toevoegen.

De variabelen die je gebruikt om een dataset te groeperen zijn meestal - al zijn er uitzonderingen - categorisch. Vaak heeft het geen zin om continue variabelen te gebruiken als groep. Bijvoorbeeld, stel dat je personen groepeert op lengte, in meter. Afhankelijk van het aantal personen in je dataset, ga je maar weinig observaties per groep hebben, aangezien het gaat om een continue variabele, die heel veel waarden kan hebben. Zoals altijd zijn er uitzonderingen, zoals bv jaartallen. Het gaat dan steeds over continue variabelen waarvan het aantal verschillende waarden enigszins beperkt is - wat vaak neer komt op gehele getallen en een beperkte domein van mogelijke waarden.

```

group_by(mpg, drv, trans)

## # A tibble: 234 x 11
## # Groups:   drv, trans [24]
##   manufacturer model     displ  year cyl trans   drv     cty   hwy fl   class
##   <fct>        <fct>     <dbl> <int> <ord> <fct>   <fct> <int> <int> <fct> <fct>
##   1 audi         a4       1.8  1999    4 auto(l~ f      18    29 p   comp~
##   2 audi         a4       1.8  1999    4 manual~ f     21    29 p   comp~
##   3 audi         a4       2    2008    4 manual~ f     20    31 p   comp~
##   4 audi         a4       2    2008    4 auto(a~ f      21    30 p   comp~
##   5 audi         a4       2.8  1999    6 auto(l~ f      16    26 p   comp~
##   6 audi         a4       2.8  1999    6 manual~ f     18    26 p   comp~

```

```

## 7 audi      a4        3.1 2008 6    auto(a~ f      18   27 p    comp~
## 8 audi      a4 quat~  1.8 1999 4    manual~ 4     18   26 p    comp~
## 9 audi      a4 quat~  1.8 1999 4    auto(l~ 4     16   25 p    comp~
## 10 audi     a4 quat~  2    2008 4    manual~ 4    20   28 p    comp~
## # ... with 224 more rows

```

We kunnen gewoon *trans* achter *drv* zetten in de `group_by` functie. Nu kunnen we zien dat de tibble gegroepeerd is op deze twee variabelen, en er zijn 24 verschillende groepen¹⁰. We kunnen de groepering van `data.frame` ook controleren door de `groups` functie te gebruiken. Dit bespaart ons het afdrukken van de hele data om de groepering te controleren. Merk op dat we hieronder weer effectief gebruik maken van het piping symbool.

```

mpg %>%
  group_by(drv, trans) %>%
  groups()

## [[1]]
## drv
##
## [[2]]
## trans

```

Oké, maar wacht eens even. Als er verder niets verandert, waarom doen we dit dan? Wel, zodra een `data.frame` is gegroepeerd, worden de volgende bewerkingen voor elke groep apart uitgevoerd. Is dat niet precies wat we nodig hadden? Inderdaad. Voor elke *drv* groep, wilden we de centrum en spreidingsmaten berekenen. Laten we het eens proberen.

```

mpg %>%
  group_by(drv) %>%
  summarize(mean_cty = mean(cty), sd_cty = sd(cty)) %>%
  pander()

summarise() ungrouping output (override with .groups argument)

```

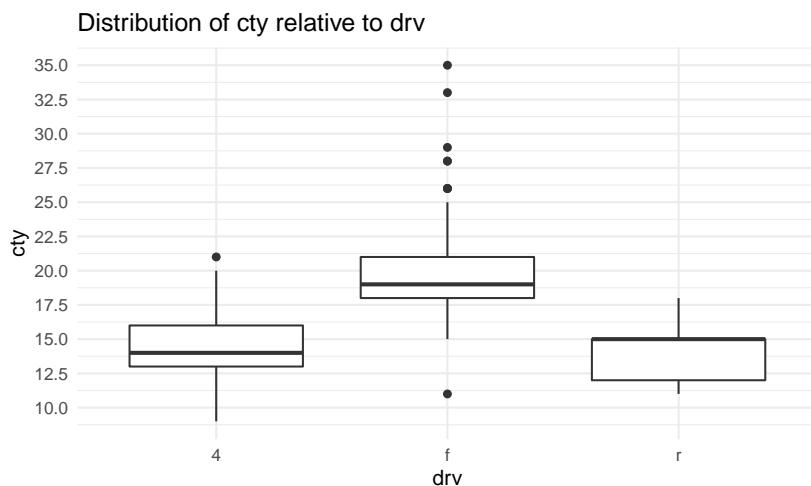
drv	mean_cty	sd_cty
4	14.33	2.874
f	19.97	3.627
r	14.08	2.216

Geweldig. Het gebruiken van `summarize` zal ons niet langer 1 rij met waarden geven. In plaats daarvan zal het 1 rij voor elke groep

¹⁰ In het bijzonder zijn er 3 *drv* waarden en 10 *trans* waarden. Er zouden dus maximaal 30 verschillende groepen kunnen zijn. Het feit dat er slechts 24 groepen worden aangegeven, betekent dat niet alle mogelijke combinaties van *drv* en *trans* waarden in de gegevens voorkomen.

teruggeven. We kunnen zien dat de gemiddelde cty veel groter is voor auto's met voorwielaandrijving dan voor andere auto's, zoals reeds werd vermoed op basis van het histogram. Een andere manier om dit te visualiseren, zonder facets te gebruiken, is het gebruik van boxplots.

```
mpg %>%
  ggplot(aes(drv, cty)) +
  geom_boxplot() +
  labs(title = "Distribution of cty relative to drv",
       x = "drv",
       y = "cty") +
  theme_minimal() +
  scale_y_continuous(breaks = seq(7.5, 35, 2.5))
```



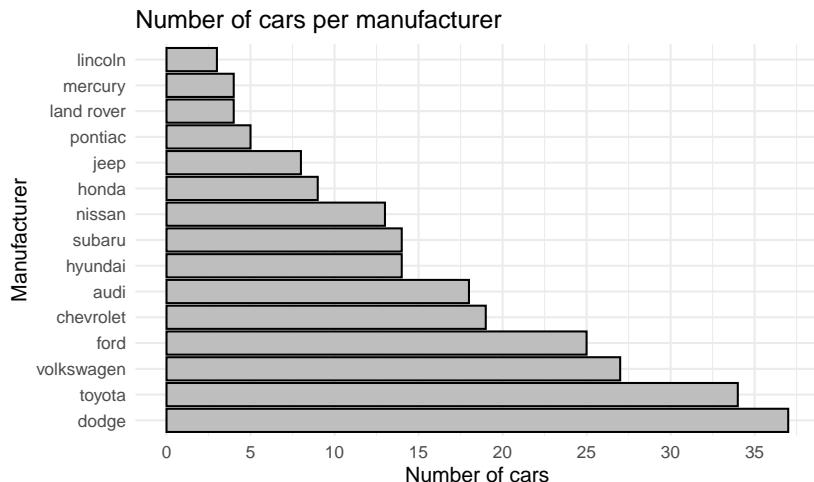
Wij kunnen nu de centraliteit en de spreiding van een continue variabele analyseren, zowel univariaat als bivariaat, ten opzichte van een categorische variabele. Om dit te doen, hebben we geleerd om twee nieuwe functies te gebruiken: `summarize` en `group_by`. Het samenvatten van een niet gegroepeerd data.frame levert een data.frame op met 1 rij. Het samenvatten van een gegroepeerd data.frame zal een data.frame teruggeven waarvan het aantal rijen gelijk is aan het aantal groepen.

Het volgende op onze lijst is de univariate analyse van een categorische variabele. Hiervoor zullen we frequentietabellen berekenen en nog een paar nieuwe interessante functies leren: `mutate`, `arrange` en `slice`. Laten we aan de slag gaan!

5.6 Univariate analyse van een categorische variabele

In onze gegevens zijn er 15 fabrikanten. Stel dat we daar meer over willen weten, d.w.z. welke fabrikant veel auto's in onze gegevens heeft, en welke minder. Grafisch kunnen we dit al doen met behulp van een staafdiagram.¹¹

```
mpg %>%
  ggplot() +
  geom_bar(aes(fct_infreq(manufacturer)),
           color = "black", fill = "grey") +
  coord_flip() +
  labs(title = "Number of cars per manufacturer",
       x = "Manufacturer",
       y = "Number of cars") +
  scale_y_continuous(breaks = seq(0, 40, 5)) +
  theme_minimal()
```

¹¹

Heb je de functie `fct_infreq` gezien die rond de `manufacturer` variabele is gewikkeld in de `geom_bar` mapping? Deze functie zorgt ervoor dat de staven worden gerangschikt van infrequent naar frequent. Deze functie is een van de handige functies voor factoren uit het pakket `forcats`, dat een anagram is van `factors` en het symboliseert het feit dat veel mensen in de R-gemeenschap kattenliefhebbers zijn (serieus). Als je wilt, kan je het installeren en de `fct_infreq` functie gebruiken (afkorting voor “reorder this factor based on (in)frequency”). Een volledige kennismaking met dit pakket wordt (voorlopig) niet verwacht.

Nu willen we dit numeriek analyseren. De meest voor de hand liggende manier om dit te doen is met een frequentietabel. Hieronder zien we het eindproduct van deze analyse. Vervolgens zullen we deze stap voor stap gaan construeren.

`summarise()` ungrouping output (override with `.groups` argument)

nr	manufacturer	frequency	relative_frequency	cumulative_relative_frequency
1	dodge	37	15.81	15.81
2	toyota	34	14.53	30.34
3	volkswagen	27	11.54	41.88
4	ford	25	10.68	52.56
5	chevrolet	19	8.12	60.68
6	audi	18	7.69	68.38

nr	manufacturer	frequency	relative_frequency	cumulative_relative_frequency
7	hyundai	14	5.98	74.36
8	subaru	14	5.98	80.34
9	nissan	13	5.56	85.9
10	honda	9	3.85	89.74
11	jeep	8	3.42	93.16
12	pontiac	5	2.14	95.3
13	land rover	4	1.71	97.01
14	mercury	4	1.71	98.72
15	lincoln	3	1.28	100

Laten we beginnen. Het eerste wat we moeten doen is het aantal waarnemingen tellen voor elke waarde van de categorische variabele. Dus, hoeveel auto's zijn er voor elke fabrikant? We kunnen dit doen door de gegevens te groeperen op *manufacturer* en dan het aantal rijen te tellen met `n`.

```
mpg %>%
  group_by(manufacturer) %>%
  summarise(frequency = n()) %>%
  pander

summarise() ungrouping output (override with .groups argument)
```

manufacturer	frequency
audi	18
chevrolet	19
dodge	37
ford	25
honda	9
hyundai	14
jeep	8
land rover	4
lincoln	3
mercury	4
nissan	13
pontiac	5
subaru	14
toyota	34
volkswagen	27

De functie `n()` heeft geen enkel argument nodig. Het berekent gewoon het aantal rijen in een `data.frame`, of, zoals in dit geval, in

elke groep. De volgende stap is om deze fabrikanten te rangschikken op basis van het aantal auto's. Om de rijen te sorteren gebruiken we de functie `arrange` van `dplyr`. Omdat we de fabrikanten willen rangschikken volgens afnemende frequentie, gebruiken we de functie `desc` (aflopend). Het volgende zal de truc doen:

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  pander

summarise() ungrouping output (override with .groups argument)
```

manufacturer	frequency
dodge	37
toyota	34
volkswagen	27
ford	25
chevrolet	19
audi	18
hyundai	14
subaru	14
nissan	13
honda	9
jeep	8
pontiac	5
land rover	4
mercury	4
lincoln	3

Het eerste argument van `arrange` is weer de data, die aan wordt gegeven met het piping-symbool. De andere argumenten worden dan gebruikt om de gegevens te ordenen. Merk op dat meer dan één variabele kan worden opgegeven. De gegevens worden dan eerst gerangschikt met behulp van de eerste variabele. Daarna zullen de volgende variabelen worden gebruikt om ex aequo's te verbreken. Wanneer `desc` rond een variabele naam wordt geplaatst, wordt deze variabele in aflopende volgorde gerangschikt. Probeer zelf maar eens wat verschillende rangschikkingen uit!

Nu moeten we een nieuwe kolom toevoegen voor de relatieve frequenties. Hiervoor gebruiken we de functie `mutate` van `dplyr`, wat letterlijk *veranderen* (mutteren) betekent. De `mutate` functie-aanroep is vergelijkbaar met die van `summarize`: het eerste argument is de data,

de andere zijn van de vorm `variabele_naam = waarde`. De variabele naam is de naam die als naam van de nieuwe kolom zal verschijnen. De waarde is de waarde voor de nieuwe kolom. Dit kan een functie zijn die een vector teruggeeft met een lengte gelijk aan die van het `data.frame`, of het kan een eenvoudige berekening zijn waarbij gebruik wordt gemaakt van andere variabelen in het `data.frame`. Het volgende statement berekent de relatieve frequenties, door elke frequentie te delen door de som van alle frequenties.

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  pander
```

`summarise()` ungrouping output (override with `.groups` argument)

manufacturer	frequency	relative_frequency
dodge	37	0.1581
toyota	34	0.1453
volkswagen	27	0.1154
ford	25	0.1068
chevrolet	19	0.0812
audi	18	0.07692
hyundai	14	0.05983
subaru	14	0.05983
nissan	13	0.05556
honda	9	0.03846
jeep	8	0.03419
pontiac	5	0.02137
land rover	4	0.01709
mercury	4	0.01709
lincoln	3	0.01282

Hier verwijst `sum(frequency)` naar de som van de frequentiekolom, terwijl `frequency` verwijst naar de specifieke waarden in elke rij. Vervolgens zullen we de cumulatieve relatieve frequentie toevoegen. We kunnen dit toevoegen aan dezelfde `mutate` aanroep.

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
```

```

relative_cumulative_frequency = cumsum(relative_frequency)) %>%
pander
summarise() ungrouping output (override with .groups argument)

```

manufacturer	frequency	relative_frequency	relative_cumulative_frequency
dodge	37	0.1581	0.1581
toyota	34	0.1453	0.3034
volkswagen	27	0.1154	0.4188
ford	25	0.1068	0.5256
chevrolet	19	0.0812	0.6068
audi	18	0.07692	0.6838
hyundai	14	0.05983	0.7436
subaru	14	0.05983	0.8034
nissan	13	0.05556	0.859
honda	9	0.03846	0.8974
jeep	8	0.03419	0.9316
pontiac	5	0.02137	0.953
land rover	4	0.01709	0.9701
mercury	4	0.01709	0.9872
lincoln	3	0.01282	1

Terwijl de berekening van de relatieve frequentie een eenvoudige formule was, wordt de cumulatieve frequentie berekend met behulp van de `cumsum`. Dit is wat men noemt een *windowfunctie*. Terwijl summary-functies altijd één waarde teruggeven, geven windowfuncties hetzelfde aantal waarden terug als de vector die als invoer wordt gebruikt. Dus, onze 15 relatieve frequenties zullen resulteren in 15 cumulatieve frequenties. Andere windowfuncties worden hieronder opgesomd.

Cumulative functions

- `cumsum`: De cumulatieve som van een vector
- `cummax`: Het cumulatieve maximum van een vector
- `cummin`: Het cumulatieve minimum van een vector
- `cumprod`: Het cumulatieve product van een vector
- `cummean`: Een cumulatief, of voortschrijdend, gemiddelde
- `cumany`: Voor logische waarden, een cumulatieve “of”
- `cumall`: Voor logische waarden, een cumulatief “en”
- `cume_dist`: Cumulatieve verdeling

Element-wise function of more than one variable

- `pmax`: Element/paar-gewijs maximum van een vector
- `pmin`: Element/paar-gewijs minimum van een vector

Ranking function

- percent_rank: rangen geschaald naar [0,1]
- row_number: Rang die banden breekt door elementen te nemen door eerste voorkomen (meestal alfabetisch)
- min_rank: Rang die groepen breekt door ze dezelfde rang te geven en de volgende rang weg te laten
- dense_rank: Zelfde als min_rank, maar zonder weglating

Shifting functions

- lead(n): verschuif waarden n plaatsen naar voren, voeg n NA's toe als laatste waarden
- lag(n): verschuif waarden n plaatsen naar achteren, voeg n NA's toe als eerste waarden

Other

- between(a,b): Liggen de waarden van een vector tussen a en b?
Geeft logisch
- ntile(x,n): Gebruik variabele x om de waarnemingen te rangschikken en ze in n even grote groepen te plaatsen.

Dat is een lange lijst, en ze zijn niet allemaal belangrijk. Sommige zul je veel gebruiken (zoals cumsum), en andere alleen in zeldzame gevallen (of nooit). In geval van nood, kom dan terug naar deze lijst.

De pmin functie geeft het paarsgewijze minimum van x en y, terwijl de pmax functie het paarsgewijze maximum van x en y geeft. Hoewel de p in deze functies wijst naar *paar* kan het gebruikt worden op meer dan twee vectoren. Ze verschillen van min en max aangezien deze laatste sumaary functies werken, die op basis van een getal het minimum/maximum teruggeven.

Een voorbeeld:

Beschouw de vectoren x en y met 10 willekeurige waarden

```
x <- runif(10)
y <- runif(10)
x

##  [1] 0.02509755 0.52685544 0.74447147 0.41505208 0.18001737 0.89297414
##  [7] 0.92517795 0.65886502 0.25179933 0.26237555

y

##  [1] 0.09193593 0.46589865 0.36637979 0.48577091 0.40897611 0.18151983
##  [7] 0.33992111 0.58584940 0.60069579 0.53975707
```

Gebruik van summary-functies

```

min(x)
## [1] 0.02509755

min(y)
## [1] 0.09193593

max(x)
## [1] 0.925178

max(y)
## [1] 0.6006958

```

Gebruik van de pair-wise window functies.

```

pmin(x,y)
## [1] 0.02509755 0.46589865 0.36637979 0.41505208 0.18001737 0.18151983
## [7] 0.33992111 0.58584940 0.25179933 0.26237555

pmax(x,y)
## [1] 0.09193593 0.52685544 0.74447147 0.48577091 0.40897611 0.89297414
## [7] 0.92517795 0.65886502 0.60069579 0.53975707

```

Merk op dat max(x,y) het maximum geeft uit beide vectoren, waar pmax(x,y) het maximum geeft op elke positie in de vectoren.

```

max(x,y)
## [1] 0.925178

data.frame(x = sample(1:10)) %>%
  mutate(lead = lead(x),
        lag = lag(x,2),
        between(x,4,8),
        ntile(x,5)) %>%
pander

```

x	lead	lag	between(x, 4, 8)	ntile(x, 5)
10	3	NA	FALSE	5
3	2	NA	FALSE	2
2	7	10	FALSE	1
7	1	3	TRUE	4
1	9	2	FALSE	1
9	8	7	FALSE	5
8	4	1	TRUE	4
4	6	9	TRUE	2
6	5	8	TRUE	3
5	NA	4	TRUE	3

Een voorbeeld van de vensterfuncties lead, lag, between en ntile wordt hierboven getoond. De lead en lag functies verschuiven waarden naar boven en beneden. Standaard worden ze met één plaats verschoven. De between functie test of waarden tussen bepaalde grenzen liggen (inclusief de grenswaarden). Tenslotte creëert de ntile-functie een variabele die waarnemingen groepeert in gelijke bins, in dit geval 5, volgens een bepaalde variabele. Zo worden waarden 1 en 2 van x gegroepeerd in bin 1, waarden 3 en 4 in bin 2, enz. Merk op dat, hoewel de ntile functie getallen teruggeeft, het eigenlijk een categorische variabele is!

Ze zien er misschien niet allemaal even logisch uit, maar soms kunnen deze windowfuncties heel krachtig en handig zijn tijdens een analyse. Zorg ervoor dat je hun bestaan kent. Maar, laten we nu teruggaan naar onze frequentietabel, want we zijn een beetje van het gebaande pad afgeraakt. Hieronder tonen we nogmaals ons laatste resultaat.

```
mpg %>%
  group_by(manufacturer) %>%
  summarise(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency)) %>%
  pander
summarise() ungrouping output (override with .groups argument)
```

manufacturer	frequency	relative_frequency	relative_cumulative_frequency
dodge	37	0.1581	0.1581
toyota	34	0.1453	0.3034
volkswagen	27	0.1154	0.4188
ford	25	0.1068	0.5256
chevrolet	19	0.0812	0.6068
audi	18	0.07692	0.6838
hyundai	14	0.05983	0.7436
subaru	14	0.05983	0.8034
nissan	13	0.05556	0.859
honda	9	0.03846	0.8974
jeep	8	0.03419	0.9316
pontiac	5	0.02137	0.953
land rover	4	0.01709	0.9701
mercury	4	0.01709	0.9872
lincoln	3	0.01282	1

We waren al klaar, zo lijkt het, maar we kunnen de tabel nog verbeteren. Een manier om dat te doen is om de relatieve (cumulatieve) frequenties om te zetten in waarden tussen 0 en 100, en om ze af te ronden op 2 decimalen. Dat laatste kan met de functie `round` en met opgave van het aantal decimalen.

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency),
         relative_frequency = round(100*relative_frequency,2),
         relative_cumulative_frequency = round(100*relative_cumulative_frequency,2)) %>%
  pander
summarise() ungrouping output (override with .groups argument)
```

manufacturer	frequency	relative_frequency	relative_cumulative_frequency
dodge	37	15.81	15.81
toyota	34	14.53	30.34
volkswagen	27	11.54	41.88
ford	25	10.68	52.56
chevrolet	19	8.12	60.68
audi	18	7.69	68.38
hyundai	14	5.98	74.36
subaru	14	5.98	80.34
nissan	13	5.56	85.9
honda	9	3.85	89.74
jeep	8	3.42	93.16
pontiac	5	2.14	95.3
land rover	4	1.71	97.01
mercury	4	1.71	98.72
lincoln	3	1.28	100

Wat hier belangrijk is om op te merken, is dat we 4 verschillende statements in `mutate` hebben gezet. In deze statements is het mogelijk om variabelen te gebruiken die eerder **binnen dezelfde `mutate` call** zijn aangemaakt: het tweede statement gebruikt de variabele in het eerste. Ook is het mogelijk om variabelen te overschrijven, d.w.z. statement 3 en 4 overschrijven de variabelen aangemaakt in statement 1 en 2.

Je vraag je misschien af waarom we 4 opgaven gebruiken in plaats van 2, en waarom we niet onmiddellijk hebben afgerond? Dat hadden

we kunnen doen voor de cumulatieve frequentie, maar niet voor de relatieve frequentie. Kan je achterhalen waarom?

Tenslotte willen we soms een rangorde, d.w.z. een rijnummer, toevoegen aan de frequentietabel. Hier gebruiken we de `row_number` functie die we hierboven hebben geïntroduceerd.

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency),
         relative_frequency = round(100*relative_frequency,2),
         relative_cumulative_frequency = round(100*relative_cumulative_frequency,2),
         nr = row_number(-frequency)) %>%
  pander(split.table = 120)

summarise() ungrouping output (override with .groups argument)
```

manufacturer	frequency	relative_frequency	relative_cumulative_frequency	nr
dodge	37	15.81	15.81	1
toyota	34	14.53	30.34	2
volkswagen	27	11.54	41.88	3
ford	25	10.68	52.56	4
chevrolet	19	8.12	60.68	5
audi	18	7.69	68.38	6
hyundai	14	5.98	74.36	7
subaru	14	5.98	80.34	8
nissan	13	5.56	85.9	9
honda	9	3.85	89.74	10
jeep	8	3.42	93.16	11
pontiac	5	2.14	95.3	12
land rover	4	1.71	97.01	13
mercury	4	1.71	98.72	14
lincoln	3	1.28	100	15

Eerder gebruikten we `desc` om aflopend te rangschikken in `arrange`, en nu gebruiken we het minteken om de rijnummers toe te wijzen volgens afnemende frequentie. Is `dplyr` echt zo inconsistent? Gelukkig is het antwoord nee. Het gebruik van het minteken is een trucje om je code te verkorten, maar kan alleen gebruikt worden voor numerieke variabelen. Dit komt omdat het afnemend sorteren van een numerieke variabele hetzelfde is als het toenemend sorteren van zijn negatieve spiegelbeeld (Overtuig jezelf!). Als zodanig, hadden we

ook het minteken kunnen gebruiken binnen arrange, en we hadden `desc` kunnen gebruiken binnen `row_number`. **Maar**, gebruik nooit het minteken om categorische waarden te sorteren (ze hebben geen negatieve tegenhanger, typisch). Het is best practice om consistent te zijn in het gebruik van - of `desc`, maar omdat dit een tutorial is, laten we je verschillende manieren zien om het te doen.

Mutate heeft het nummer aan het eind van de tabel toegevoegd. Dat is wat we in de meeste gevallen willen, maar niet in dit geval. Om de volgorde van de variabelen te veranderen, kunnen we de `select` functie van `dplyr` gebruiken. Deze functie kan worden gebruikt om variabelen te *selecteren*, en zal ze in de opgegeven volgorde plaatsen. Dus, wat we kunnen doen is het volgende.

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency),
         relative_frequency = round(100*relative_frequency,2),
         relative_cumulative_frequency = round(100*relative_cumulative_frequency,2),
         nr = row_number(-frequency)) %>%
  select(nr, manufacturer, frequency, relative_frequency, relative_cumulative_frequency) %>%
  pander(split.table = 120)

summarise() ungrouping output (override with .groups argument)
```

nr	manufacturer	frequency	relative_frequency	relative_cumulative_frequency
1	dodge	37	15.81	15.81
2	toyota	34	14.53	30.34
3	volkswagen	27	11.54	41.88
4	ford	25	10.68	52.56
5	chevrolet	19	8.12	60.68
6	audi	18	7.69	68.38
7	hyundai	14	5.98	74.36
8	subaru	14	5.98	80.34
9	nissan	13	5.56	85.9
10	honda	9	3.85	89.74
11	jeep	8	3.42	93.16
12	pontiac	5	2.14	95.3
13	land rover	4	1.71	97.01
14	mercury	4	1.71	98.72
15	lincoln	3	1.28	100

Het gebruik van het `select` statement doet de truc. Maar ik ben nog steeds een beetje lui, en ik wil niet alle variabelen uitschrijven, alleen maar om er één op de eerste positie te zetten. We hadden hier maar 5 variabelen, maar stel je voor wat een verspilling van tijd het zou zijn als we er meer hadden, zoals 6 bijvoorbeeld. Kunnen we niet gewoon zeggen, zet `nr` vooraan, en voeg dan alle andere variabelen toe in hun oorspronkelijke volgorde? Dplyr biedt redding!

Gelukkig kunnen we dat. We kunnen gewoon de functie `everything` toevoege aan `select`. Dit zal alle kolommen na `nr` toevoegen, zonder `nr` voor een tweede keer te herhalen. Geweldig, is het niet?

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency),
         relative_frequency = round(100*relative_frequency,2),
         relative_cumulative_frequency = round(100*relative_cumulative_frequency,2),
         nr = row_number(-frequency)) %>%
  select(nr, everything()) %>%
  pander(split.table = 120)

summarise() ungrouping output (override with .groups argument)
```

nr	manufacturer	frequency	relative_frequency	relative_cumulative_frequency
1	dodge	37	15.81	15.81
2	toyota	34	14.53	30.34
3	volkswagen	27	11.54	41.88
4	ford	25	10.68	52.56
5	chevrolet	19	8.12	60.68
6	audi	18	7.69	68.38
7	hyundai	14	5.98	74.36
8	subaru	14	5.98	80.34
9	nissan	13	5.56	85.9
10	honda	9	3.85	89.74
11	jeep	8	3.42	93.16
12	pontiac	5	2.14	95.3
13	land rover	4	1.71	97.01
14	mercury	4	1.71	98.72
15	lincoln	3	1.28	100

Select, arrange, mutate, summary, group_by. We hebben dplyr al aardig onder de knie. Toch niet zo moeilijk, is het niet? Natuurlijk

zijn er nog meer functies, maar deze 5 behoren echt tot de belangrijkste functies voor data manipulatie. Alleen `filter` ontbreekt in de dplyr hall of fame, maar die laten we voor een andere keer. Het is altijd een goed idee om iets te hebben om naar uit te kijken, vind je niet?

Voordat we verder gaan met bivariate analyse, zijn er nog twee dingen die we moeten doen. Kijk naar de `slice` functie, en nog eens terugkijken naar het `%>%` symbool.

De `slice` functie kan gebruikt worden om rijen uit een `data.frame` te `slicen`. Stel dat we een hele lange frequentietabel hebben, en we zijn alleen geïnteresseerd in de top 10 waarden. We kunnen dan de eerste 10 rijen van deze tabel als volgt slicen.

```
mpg %>%
  group_by(manufacturer) %>%
  summarise(frequency = n()) %>%
  arrange(desc(frequency)) %>%
  mutate(relative_frequency = frequency/sum(frequency),
         relative_cumulative_frequency = cumsum(relative_frequency),
         relative_frequency = round(100*relative_frequency,2),
         relative_cumulative_frequency = round(100*relative_cumulative_frequency,2),
         nr = row_number(-frequency)) %>%
  select(nr, everything()) %>%
  slice(1:10) %>%
  pander(split.table = 120)

summarise() ungrouping output (override with .groups argument)
```

nr	manufacturer	frequency	relative_frequency	relative_cumulative_frequency
1	dodge	37	15.81	15.81
2	toyota	34	14.53	30.34
3	volkswagen	27	11.54	41.88
4	ford	25	10.68	52.56
5	chevrolet	19	8.12	60.68
6	audi	18	7.69	68.38
7	hyundai	14	5.98	74.36
8	subaru	14	5.98	80.34
9	nissan	13	5.56	85.9
10	honda	9	3.85	89.74

Dat is slice. Niets meer, niets minder. We hebben een *groot* blok code opgebouwd, maar ons pipingsymbool rijgt het netjes aan elkaar, nietwaar? Stel je even voor dat we dit symbool niet hadden. In zo'n geval zouden we elk eerste argument opnieuw moeten plaatsen in de

functie waar het door wordt gevuld. Dus, mpg moet in group by. De group_by zou in summarize moeten staan, enz. Het resultaat zou het volgende zijn.

```
pander(
  slice(
    select(
      mutate(
        arrange(
          summarize(
            group_by(mpg,
                      manufacturer),
            frequency = n(),
            desc(frequency)),
            relative_frequency = frequency/sum(frequency),
            relative_cumulative_frequency = cumsum(relative_frequency),
            relative_frequency = round(100*relative_frequency,2),
            relative_cumulative_frequency = round(100*relative_cumulative_frequency,2),
            nr = row_number(-frequency)),
            nr,
            everything()),
        1:10)
  )
)
```

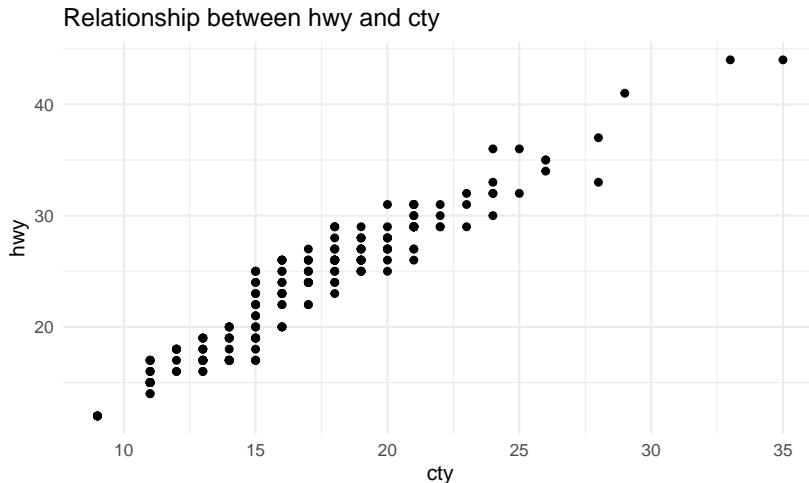
Dat is nogal een puinhoop, is het niet? Alle functies zijn gescheiden van hun argumenten, en we kunnen niet echt opmaken wat we aan het doen waren. Het piping symbool doet zijn werk om onze code leesbaar en begrijpelijk te maken. Zorg ervoor dat je het verstandig gebruikt!

Nu gaan we verder met de bivariate analyse van twee continue variabelen.

5.7 Bivariate analyse van een continue variabele ten opzichte van een andere continue variabele

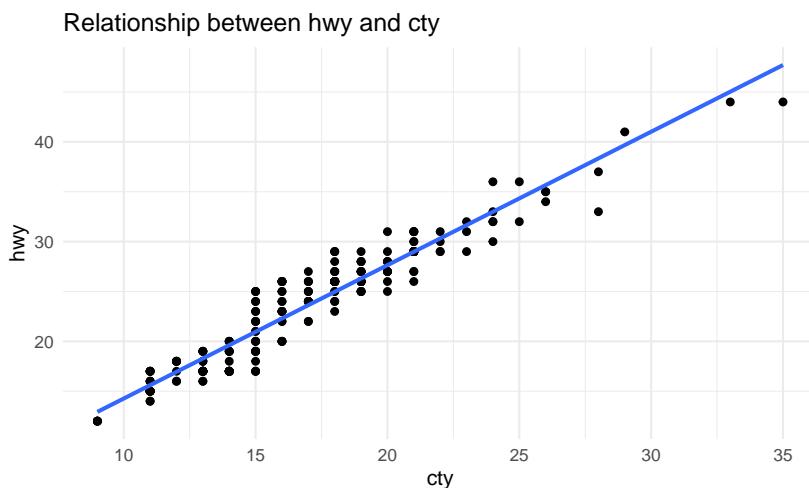
De relatie tussen twee continue variabelen kan gemakkelijk grafisch worden weergegeven met een scatter plot. Laten we eens kijken naar *cty* ten opzichte van *hwy*.

```
mpg %>%
  ggplot(aes(cty, hwy)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Relationship between hwy and cty")
```



Er is een zeer duidelijk verband tussen deze variabelen, en het blijkt vrij lineair te zijn. Laten we een lineaire lijn trekken door de puntenwolk.

```
mpg %>%
  ggplot(aes(cty, hwy)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Relationship between hwy and cty") +
  geom_smooth(method = "lm", se = F)
```



Een lineair verband als dit kan zeer effectief worden gemeten met de correlatiecoëfficiënt. Het berekenen van correlaties kan worden gedaan met de base-R functie `cor`. Deze functie verwacht echter een `data.frame` met alleen continue variabelen, omdat het alle mogelijke correlaties tussen deze zal berekenen. Correlaties tussen categorische variabelen (behalve ordinale) kunnen niet worden berekend. Daarom selecteren we eerst de continue variabelen met de `select` functie van

`dplyr`. De variabelen waarin we geïnteresseerd zijn, zijn *displ*, *year*, *cty* en *hwy*.

```
mpg %>%
  select(displ, year, cty, hwy) %>%
  cor %>%
  pander
```

	displ	year	cty	hwy
displ	1	0.1478	-0.7985	-0.766
year	0.1478	1	-0.03723	0.002158
cty	-0.7985	-0.03723	1	0.9559
hwy	-0.766	0.002158	0.9559	1

De `select` aanroep kan groot worden bij het selecteren van numerieke variabelen in een grote dataset. We kunnen het herschrijven met de `select_if` functie, die geen variabele namen verwacht, maar wel een functie nodig heeft om te testen of een variabele meegenomen moet worden of niet. We kunnen de `is.numeric` functie gebruiken om te testen of een vector numeriek is. Dus,¹²

```
mpg %>% select_if(is.numeric) %>%
  cor %>%
  pander
```

¹² Merk op hoe we de `is.numeric` functie binnen `select_if` gebruiken: zonder aanhalingstekens en zonder haakjes. Dit is belangrijk!

	displ	year	cty	hwy
displ	1	0.1478	-0.7985	-0.766
year	0.1478	1	-0.03723	0.002158
cty	-0.7985	-0.03723	1	0.9559
hwy	-0.766	0.002158	0.9559	1

Hier zien we de zeer sterke correlatie tussen *cty* en *hwy*. Wij zien ook sterke negatieve correlaties tussen *displ* enerzijds en *cty* en *hwy* anderzijds. Kan je proberen deze te visualiseren zoals wij hebben gedaan voor *cty* en *hwy*?

We zouden ook de door `cor` berekende correlaties kunnen visualiseren met `ggplot`, maar er is iets mis. De gegevens die de correlatiefunctie oplevert zijn niet erg netjes. Er zijn variabele namen in de kolommen en in de rijen? Nog erger, het is geen `data.frame`!

```
mpg %>%
  select_if(is.numeric) %>%
```

```
cor %>%
  class
## [1] "matrix" "array"
```

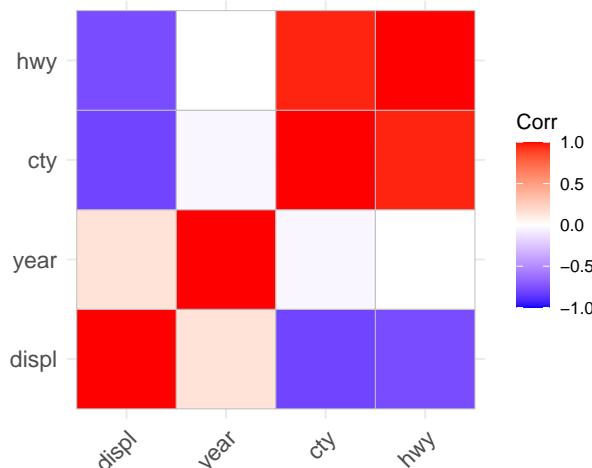
Matrix? De film? Nee, matrix is een ander type object in R waar je misschien nog niet van gehoord hebt. Terwijl een data.frame een verzameling vectoren is, kun je een matrix zien als een tweedimensionale vector. Dit betekent dat, net als in een vector, alle elementen in een matrix hetzelfde type moeten hebben. Zoals een vector namen kan hebben, kunnen ook de rijen en kolommen van een matrix namen hebben, zoals in ons voorbeeld.

Maar, zoals we weten, kan `ggplot2` alleen werken met `data.frames`. We zijn compleet verloren! Maar, we kunnen misschien van een matrix een `data.frame` maken. En dan kunnen we het zo aanpassen dat alle variabelen in kolommen staan. We kunnen het proberen...

Maar, dat lijkt een hoop werk. En we zijn nog steeds lui. Gelukkig zijn de meeste R-gebruikers tot op zekere hoogte lui, en iemand moet dit werk ooit gedaan hebben, en moet zo genereus geweest zijn om het in een pakket te stoppen. `ggcorrplot` is het antwoord. Het `ggcorrplot` pakket heeft één enkele functie die nuttig is voor ons: `ggcorrplot`. Merk op dat zowel het pakket als de functie dezelfde naam hebben gekregen. Hoe genereus! Maar wees voorzichtig: `cor` had één `r`, `ggcorrplot` heeft er twee.

```
library(ggcorrplot)

mpg %>%
  select_if(is.numeric) %>%
  cor %>%
  ggcorrplot()
```



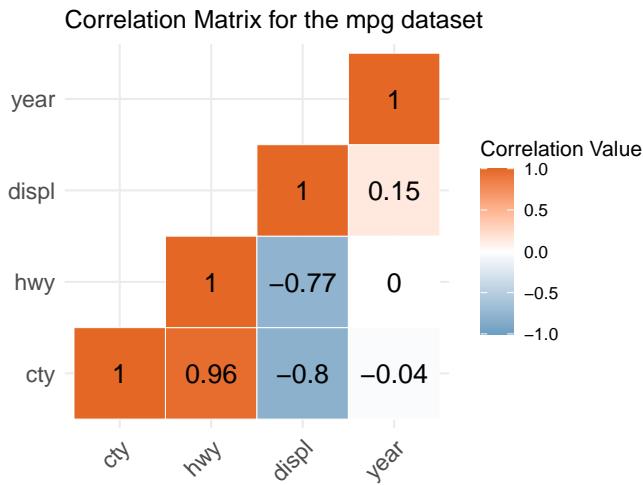
Wat we krijgen is een visuele matrix. De kleur van de vierkantjes geeft de richting van het verband aan (standaard is rood positief en blauw negatief). Er zijn echter veel verschillende opties om `ggcorrplot` te maken zoals we het willen. Laten we eens kijken naar de belangrijkste:

- method: “square” of “circle”: de vorm van de elementen in de matrix
- lab: indien TRUE, zullen de correlatiwaarden boven de vierkanten (of cirkels) worden getoond
- lab_col en lab_size: hiermee kunnen we veranderen hoe de waarden worden afdrukt
- outline_color: de kleur van de randen van de vierkanten
- type: “full”, “lower” of “upper”: een correlatiematrix is symmetrisch, dus we kunnen kiezen om alleen de onderste of de bovenste helft te tonen.
- ggtheme: je kan één van de standaard ggplot2-thema’s gebruiken: theme_grey, theme_minimal, theme_classic. Geef ze op zonder haakjes of aanhalingsstekens, net zoals we een paar minuten geleden deden met `is.numeric`.
- titel
- legende.titel
- colors, een vector van drie kleuren voor de lage, midden en hoge waarden. Standaard: `c("blauw", "wit", "rood")`
- hc.order: indien ingesteld op TRUE kunnen we de variabelen ordenen om te tonen welke variabelen het meest verwant zijn
- show.diag: Toon de diagonaal indien TRUE (in geval type gelijk is aan “lower” of “upper”)

An example of a slightly modified version of our plot is the following:¹³

```
mpg %>%
  select_if(is.numeric) %>%
  cor %>%
  ggcorrplot(type = "lower", ggtheme = theme_minimal, colors = c("#6D9EC1", "white", "#E46726"),
             show.diag = T,
             lab = T, lab_size = 5,
             title = "Correlation Matrix for the mpg dataset",
             legend.title = "Correlation Value",
             outline.color = "white",
             hc.order = T)
```

¹³ Meer informatie en instellingen voor deze plots zijn te vinden in deze handleiding

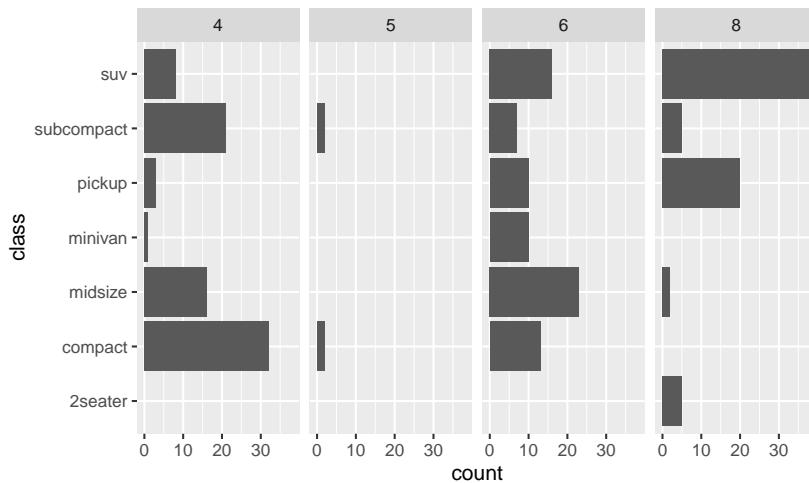


We hebben al een lange en opwindende weg afgelegd door het tidyverse! Het laatste wat we willen kunnen, is een bivariate analyse doen van twee categorische variabelen. Hiervoor zullen we contingentietabellen leren construeren. We hebben bijna alle soorten beschrijvende analyse onder de knie.

5.8 Bivariate analyse van een categorische variabele ten opzichte van een andere categorische variabele

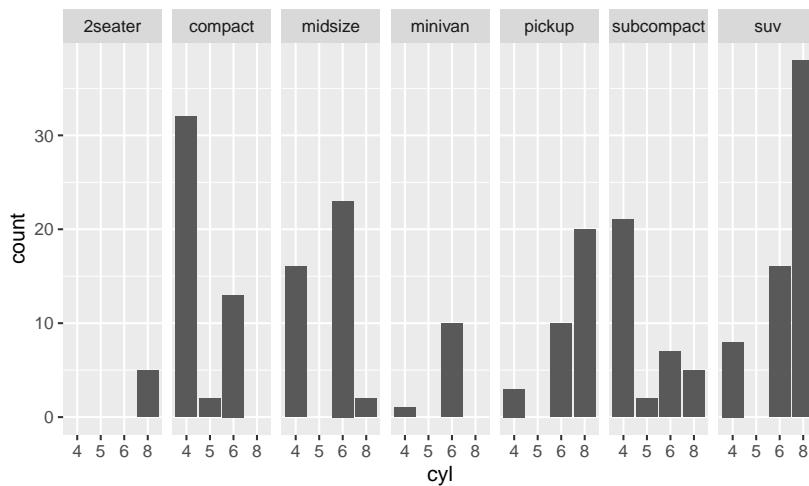
We beginnen weer met het bekijken van een grafiek. Stel dat we de relatie tussen *class* en *cyl* willen bekijken. We zouden faceted staafdiagrammen kunnen maken.

```
mpg %>%
  ggplot(aes(class)) +
  geom_bar() +
  facet_grid(~cyl) +
  coord_flip()
```



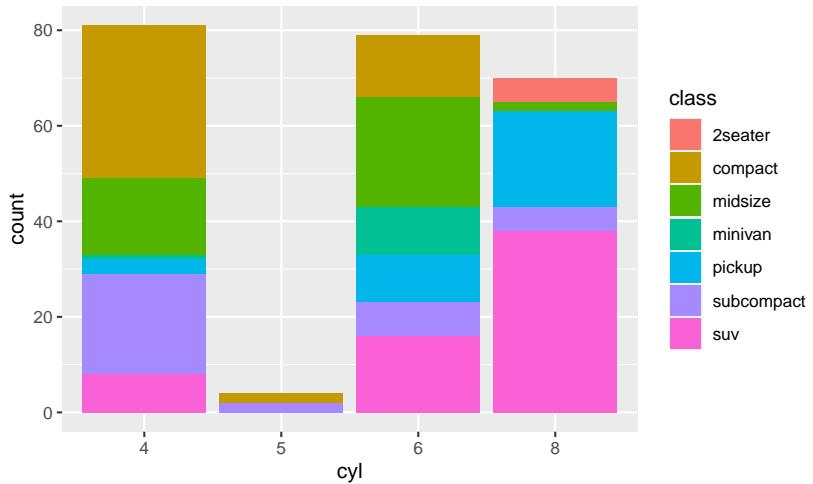
Hier zien we dat de verdeling van de class verschilt wanneer het aantal cilinders verandert. Voor 4 cilinders zijn de meeste auto's compact, terwijl voor 8 cilinders de meeste auto's suv's zijn. We kunnen het echter ook vanuit een ander perspectief bekijken.

```
mpg %>%
  ggplot(aes(cyl)) +
  geom_bar() +
  facet_grid(~class)
```



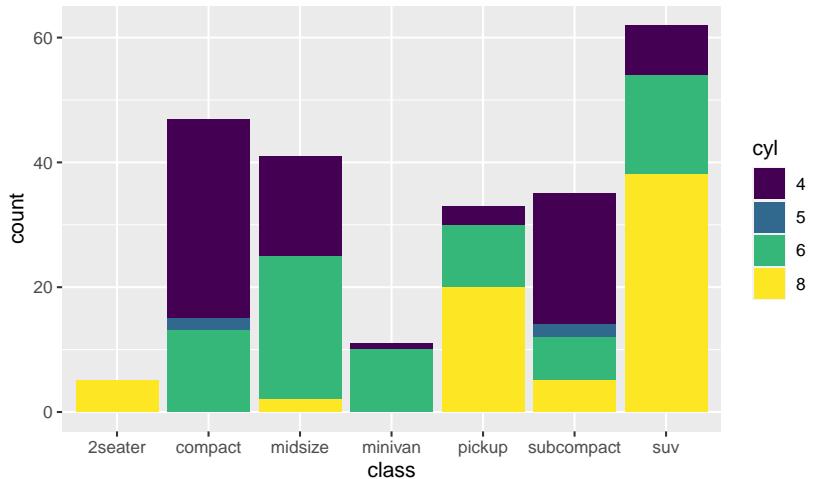
En nog een

```
mpg %>%
  ggplot(aes(cyl)) +
  geom_bar(aes(fill = class))
```



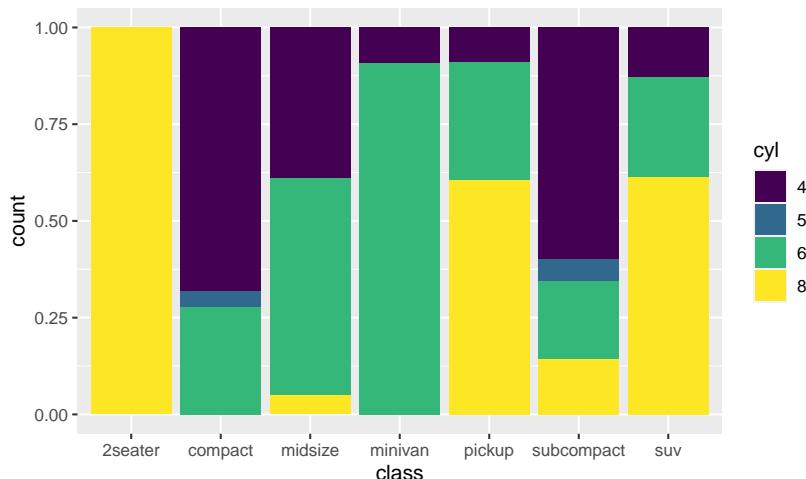
En nog een

```
mpg %>%
  ggplot(aes(class)) +
  geom_bar(aes(fill = cyl))
```



En nog een

```
mpg %>%
  ggplot(aes(class)) +
  geom_bar(aes(fill = cyl), position = "fill")
```



Er zijn inderdaad vele manieren om de relatie tussen twee categorische variabelen te bekijken. De vele grafieken die kunnen worden gemaakt, geven aan dat er ook meerdere contingentietabellen kunnen zijn. Alles hangt echt af van de vraag die je wilt beantwoorden. Gelukkig kennen we inmiddels al een heleboel belangrijke functies, die we goed kunnen gebruiken. We beginnen met te tellen hoeveel auto's er zijn voor elke cilinder-klasse combinatie.

```
mpg %>%
  group_by(cyl, class) %>%
  summarise(frequency = n()) %>%
  pander

summarise() regrouping output by 'cyl' (override with .groups argument)
```

cyl	class	frequency
4	compact	32
4	midsize	16
4	minivan	1
4	pickup	3
4	subcompact	21
4	suv	8
5	compact	2
5	subcompact	2
6	compact	13
6	midsize	23
6	minivan	10
6	pickup	10
6	subcompact	7
6	suv	16

cyl	class	frequency
8	2seater	5
8	midsize	2
8	pickup	20
8	subcompact	5
8	suv	38

Geweldig, dat ging perfect! Om hier een contingentietabel van te maken, willen we een van de twee variabelen op de kolommen zetten. Dit zal een matrix-achtige structuur creëren. Hier hebben we een nieuwe functie voor nodig, die `spread` heet. Deze functie komt uit het `tidyverse` pakket, dat wordt gebruikt om gegevens op te schonen. Echter, de andere functies uit dit pakket zullen hier niet worden besproken.

De functie `spread` heeft 2 argumenten, `key` en `value` (behalve data, natuurlijk). De key verwijst naar de variabele waarvan we de waarden als nieuwe kolommen willen plaatsen, in dit geval `class`. De value verwijst naar de variabele waarvan we de waarden in de nieuwe kolommen willen plaatsen, in dit geval onze frequentie. Geen idee wat er gaat gebeuren? Laten we eens naar een voorbeeld kijken.

```
mpg %>%
  group_by(cyl, class) %>%
  summarise(frequency = n()) %>%
  spread(class, frequency) %>%
  pander
```

`summarise()` regrouping output by ‘cyl’ (override with `.groups` argument)

cyl	2seater	compact	midsize	minivan	pickup	subcompact	suv
4	NA	32	16	1	3	21	8
5	NA	2	NA	NA	NA	2	NA
6	NA	13	23	10	10	7	16
8	5	NA	2	NA	20	5	38

Heb je gezien wat er gebeurd is? Vergelijk gewoon de laatste twee tabellen. Wat gebeurt er als je `cyl` als key gebruikt?

Nu, waarom tonen sommige van de waarden in dit `data.frame` NA, wat staat voor Not Available? Dat komt omdat niet alle `class-cyl` combinaties bestaan, d.w.z., onze lijst voordien toonde ons slechts 19 bestaande combinaties. Er zijn geen auto’s voor de andere 9 combinaties. We houden echter niet zo van NA’s, dus laten we ze veranderen in nullen. Gelukkig is dit een functie van de `spread` functie. Het ar-

argument `fill` wordt gebruikt om de lege cellen te “vullen”. We hoeven het alleen maar op nul te zetten.¹⁴

```
mpg %>%
  group_by(cyl, class) %>%
  summarize(frequency = n()) %>%
  spread(class, frequency, fill = 0) %>%
  pander
```

`summarise()` regrouping output by ‘cyl’ (override with `.groups` argument)

cyl	2seater	compact	midsize	minivan	pickup	subcompact	suv
4	0	32	16	1	3	21	8
5	0	2	0	0	0	2	0
6	0	13	23	10	10	7	16
8	5	0	2	0	20	5	38

Geweldig. Dit is wat ik zou noemen een “Contingentietabel met absolute frequenties”. Het toont ons het absolute aantal auto’s voor elke combinatie van cilinder en klasse, en het vertelt ons dat de combinatie 8 cilinders en SUV het meest prominent is in onze gegevens. Deze contingentietabel past heel goed bij onze eerste twee grafieken.

Een andere vraag die we kunnen stellen is: Als we kijken naar auto’s met 4 cilinders, wat is dan de specifieke verdeling over de klassen? In dit geval zouden we relatieve frequenties willen hebben. Laten we dit eens proberen.

```
mpg %>%
  group_by(cyl, class) %>%
  summarize(frequency = n()) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  pander
```

`summarise()` regrouping output by ‘cyl’ (override with `.groups` argument)

cyl	class	frequency	relative_frequency
4	compact	32	0.3951
4	midsized	16	0.1975
4	minivan	1	0.01235
4	pickup	3	0.03704
4	subcompact	21	0.2593
4	suv	8	0.09877

¹⁴ We zijn fill al in verschillende contexten tegengekomen. Het wordt gebruikt om de vulkleur in plots te specificeren, het kan gebruikt worden als een position in geom_bar om de balken tot 100% te vullen, en nu kunnen we het gebruiken om lege ruimtes in de contingentietabel op te vullen. Verwar ze niet met elkaar!

cyl	class	frequency	relative_frequency
5	compact	2	0.5
5	subcompact	2	0.5
6	compact	13	0.1646
6	midsized	23	0.2911
6	minivan	10	0.1266
6	pickup	10	0.1266
6	subcompact	7	0.08861
6	suv	16	0.2025
8	2seater	5	0.07143
8	midsized	2	0.02857
8	pickup	20	0.2857
8	subcompact	5	0.07143
8	suv	38	0.5429

Dat is precies wat we wilden hebben. Maar, valt je niet iets vreemds op?

Binnen elke cilindergroep tellen de relatieve frequenties op tot één. Kijk bijvoorbeeld naar cyl = 5, daar zijn 50% compacte auto's en 50% subcompact. Dat is wat we wilden, maar... dit is niet wat er eerder gebeurde bij het maken van frequentietabellen. Wat is er veranderd?

Het antwoord is subtiel, lastig en belangrijk.

Telkens wanneer een sommatie wordt gedaan van een gegroepeerd data.frame, verwijdt summary de laatste groeperingsvariabele. In het geval van onze frequentietabel eerder:

```
mpg %>%
  group_by(manufacturer) %>%
  summarize(frequency = n()) %>%
  groups

## list()
```

Na het samenvatten, zijn er geen groeperingsvariabelen meer. Er was er maar één, en die is verwijderd. De functie summarize gaat er impliciet van uit dat de groepering nutteloos is geworden na de summarize. Dit betekent dat, als we **sum(frequency)** in de volgende regel gebruiken om relatieve frequenties te berekenen, het de som van de frequenties in de hele tabel zou berekenen.

Nu, terug naar ons voorbeeld. Nadat de frequenties zijn berekend, zijn de gegevens alleen gegroepeerd per cyl. Dus **sum(frequency)** berekent nu de som van de frequenties voor elke cyl-groep. Inderdaad, vergeet niet, voor een gegroepeerd data.frame, gebeuren alle bewerkingen afzonderlijk voor elke groep. Dat is precies de reden waarom de relatieve frequenties binnen elke cyl-groep bij 1 optellen.

```
mpg %>%
  group_by(cyl, class) %>%
  summarize(frequency = n()) %>%
  groups

## [[1]]
## cyl
```

Als we de volgorde van de groeperingsniveaus in de `group_by` functie veranderen, veranderen we ook de relatieve frequenties die berekend zullen worden. Als we de klasse eerst zetten, zullen de relatieve frequenties voor elk van de klassen opgeteld 1 zijn.

```
mpg %>%
  group_by(class, cyl) %>%
  summarize(frequency = n()) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  pander
```

`summarise()` regrouping output by ‘class’ (override with `.groups` argument)

class	cyl	frequency	relative_frequency
2seater	8	5	1
compact	4	32	0.6809
compact	5	2	0.04255
compact	6	13	0.2766
midsize	4	16	0.3902
midsize	6	23	0.561
midsize	8	2	0.04878
minivan	4	1	0.09091
minivan	6	10	0.9091
pickup	4	3	0.09091
pickup	6	10	0.303
pickup	8	20	0.6061
subcompact	4	21	0.6
subcompact	5	2	0.05714
subcompact	6	7	0.2
subcompact	8	5	0.1429
suv	4	8	0.129
suv	6	16	0.2581
suv	8	38	0.6129

Toegegeven, dit is verwarring, maar tegelijkertijd is het zeer nuttig. Je moet dus wel letten op de volgorde van de variabelen in de

group_by functie, en op het aantal samenvattingen dat na het groeperen wordt gebruikt.

Nu we de relatieve frequenties hebben, kunnen we die met spread opnieuw vormgeven om een matrix-achtige tabel te maken. Deze keer stellen we de relatieve_frequentie in als waarde.

```
mpg %>%
  group_by(class, cyl) %>%
  summarize(frequency = n()) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  spread(cyl, relative_frequency, fill = 0)    %>%
  pander
```

`summarise()` regrouping output by ‘class’ (override with `.groups` argument)

class	frequency	4	5	6	8
2seater	5	0	0	0	1
compact	2	0	0.04255	0	0
compact	13	0	0	0.2766	0
compact	32	0.6809	0	0	0
midsize	2	0	0	0	0.04878
midsize	16	0.3902	0	0	0
midsize	23	0	0	0.561	0
minivan	1	0.09091	0	0	0
minivan	10	0	0	0.9091	0
pickup	3	0.09091	0	0	0
pickup	10	0	0	0.303	0
pickup	20	0	0	0	0.6061
subcompact	2	0	0.05714	0	0
subcompact	5	0	0	0	0.1429
subcompact	7	0	0	0.2	0
subcompact	21	0.6	0	0	0
suv	8	0.129	0	0	0
suv	16	0	0	0.2581	0
suv	38	0	0	0	0.6129

Oh... dat is niet wat we verwacht hadden? We verwachtten één rij voor elke klasse, maar nu hebben we er meer dan één. De reden is eenvoudig. Voor elke combinatie van klasse en cilinder hebben wij twee waarden: frequentie en relatieve frequentie. Wanneer wij de relatieve frequenties over één lijn willen verdelen, is er geen plaats meer voor de frequenties. Het gevolg is dat de lijnen niet meer, zoals voorheen, tot één lijn kunnen *samenvallen*. We kunnen dit echter

eenvoudig oplossen door eerst de frequenties weg te nemen, door select te gebruiken.

```
mpg %>%
  group_by(class, cyl) %>%
  summarize(frequency = n()) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  select(-frequency) %>%
  spread(cyl, relative_frequency, fill = 0) %>%
  pander
```

`summarise()` regrouping output by ‘class’ (override with `.groups` argument)

class	4	5	6	8
2seater	0	0	0	1
compact	0.6809	0.04255	0.2766	0
midsize	0.3902	0	0.561	0.04878
minivan	0.09091	0	0.9091	0
pickup	0.09091	0	0.303	0.6061
subcompact	0.6	0.05714	0.2	0.1429
suv	0.129	0	0.2581	0.6129

Dit lijkt er meer op. Merk op dat in plaats van de variabelen die we willen houden aan te geven met `select`, we de variabele die we willen verwijderen aangeven met een minteken, wat veel korter is. Verwar het niet met het minteken bij het rangschikken van gegevens, want dat is iets heel anders.

Op elke rij kunnen we nu de verdeling van verschillende cilinders zien voor een bepaalde klasse auto’s. Je zou het ook andersom kunnen draaien, en de verdeling van verschillende klassen voor een specifiek aantal cilinders kunnen laten zien. Net zoals we vele grafieken hadden om naar deze twee variabelen te kijken, kunnen we vele contingentietabellen maken.

Stel ten slotte dat we een contingentietabel willen met algemene relatieve frequenties. D.w.z. dat we de vraag willen beantwoorden: welk percentage van de auto’s heeft 5 cilinders en is van de klasse compact. We kunnen het laatste stukje code recyclen, en we hoeven maar één regel toe te voegen.

```
mpg %>%
  group_by(class, cyl) %>%
  summarize(frequency = n()) %>%
  ungroup() %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
```

```
select(-frequency) %>%
spread(cyl, relative_frequency, fill = 0) %>%
pander
```

`summarise()` regrouping output by ‘class’ (override with `.groups` argument)

class	4	5	6	8
2seater	0	0	0	0.02137
compact	0.1368	0.008547	0.05556	0
midsize	0.06838	0	0.09829	0.008547
minivan	0.004274	0	0.04274	0
pickup	0.01282	0	0.04274	0.08547
subcompact	0.08974	0.008547	0.02991	0.02137
suv	0.03419	0	0.06838	0.1624

De functie `ungroup` die we hebben toegevoegd, verwijdert alle groepen. Als gevolg daarvan wordt de som van de frequenties berekend over het volledige data.frame. Dus, alleen de relatieve frequenties van alle combinaties zullen bij elkaar opgeteld één zijn (je kan het zelf narekenen).

5.9 *Background material*

Wij hebben 5 verschillende analyses uitgevoerd en onderweg heel wat nieuwe nuttige functies geleerd. Vergeet zeker niet de belangrijkste: `select`, `arrange`, `group_by`, `summarize`, `mutate` en `spread`!

Als je meer wilt weten, kan je deze materialen raadplegen:

- R for Data Science, chapter 5
- dplyr Introduction

6

[Lecture notes] Structuur van een data analyse project.

Bij dit hoorcollege zijn geen lecture notes beschikbaar. We verwijzen hiervoor naar de slides van het hoorcollege, incl. referenties.

7

[Lecture notes] Data voorbereiden

7.1 Beginnen bij het begin

- Alvorens we aan een exploratieve data analyse kunnen beginnen, moeten we eerst onze data voorbereiden.
- Er kunnen drie grote fases geïdentificeerd worden tijdens de datavoorbereiding.
 - Correct inlezen van de data.
 - Identificeren van problemen in de data en deze corrigeren van de data.
 - Opwaarderen van de data.
- Data kan in diverse formaten aangeleverd worden en de eerste stap is ervoor zorgen dat de data ingeladen is in R. Hierbij zijn er twee specifieke elementen om aandacht aan te schenken:
 - De data analyst moet ervoor zorgen dat de data correct ingeladen wordt en dat de inhoud na het inladen overeenkomt met de inhoud toen deze data de laatste keer werd opgeslagen.
 - De data analyst moet ervoor zorgen dat de verschillende variabelen het juiste data type hebben.
- De volgende fase is het opkuisen van de data. Dit betekent dat men fouten gaat identificeren en deze ‘oplost’ alvorens verder te gaan.
 - Er zijn verschillende soorten fouten die in de data kunnen sluipen. Enkele mogelijke fouten zijn:
 - * Sommige waarden ontbreken (geen waarde voor bepaalde variabele bij bepaalde observaties).
 - * Sommige waarden zijn fout. Bijvoorbeeld: voor een deel observaties is de afstand in km opgeslagen ipv mijl of is er een typfout in de waarde van een categorische variabele.
 - * Sommige observaties staan meerdere keren in de dataset.
 - Het opkuisen van data gebeurt in principe in 2 stappen:

- * Eerst moeten we de data bestuderen en fouten identificeren.
- * Vervolgens moeten we de fouten in de data ‘corrigeren’ (indien mogelijk).
- Het opwaarderen van de data betreft een reeks transformaties met als doel de data bruikbaarder te maken voor exploratieve data analyse. Er zijn verschillende manieren om dit te bereiken:
 - Bestaande variabelen transformeren naar nieuwe variabelen die geschikter zijn om patronen in de data bloot te leggen. Bijvoorbeeld het transformeren van een continue variabele naar een categorische variabele of het creëren van een nieuwe variabele ‘gemiddelde snelheid’ op basis van de variabelen ‘reistijd’ en ‘totaal afgelegde afstand’.
 - Het opsplitsen van de dataset in meerdere datasets die apart bestudeerd worden. Dit is vooral zinvol indien de dataset verschillende soorten observaties bevat of een deel observaties met uitzonderlijke waarden.

7.2 Data inlezen

7.2.1 Uitdagingen bij het correct inlezen van data

- Data kan in verschillende formaten aangeleverd worden. Afhankelijk van het formaat, zal je andere functies moeten gebruiken om de data correct in te lezen
- Maar zelfs als je de juiste functie gebruikt, kan het inlezen fout gaan omdat een computer data altijd als een reeks van 1 en 0’en opslaat en er daarom een soort vertaalsleutel nodig is van 1 en 0’en naar leesbare tekst. Deze vertaalslag wordt gerealiseerd door encoderingschema’s en je moet er voor zorgen dat bij het inlezen van data je het juiste encoderingschema hanteert.
- Eenmaal de data is ingeladen, moet je er voor zorgen dat R de datatypes juist identificeert. De meeste dataformaten houden geen informatie bij van welk datatype een specifieke variabele is en dus moet R dit ‘raden’. Omdat dit wel eens fout kan gaan, moet je als analist dit controleren en corrigeren waar nodig.

7.2.2 Dataformaten

Flat-file databestanden

- Lees de bron over delimited en fixed-width bestanden.
- Flat-file databestanden bevatten data die in een tabelvorm passen:
 - Iedere rij is een observatie.
 - Iedere kolom stelt een variabele voor.

- Alle items in een kolom zijn van dezelfde soort.
- Cellen van de tabel bevatten enkelvoudige gegevens (dus niet een kolom hobby's met hierin meerdere hobby's in 1 cel).
- De volgorde van de kolommen is niet van belang.
- De volgorde van de rijen is niet belang.
- Volgende data kan dus in een flat-file bestand opgeslagen worden:

naam	voornaam
Nelissen	Rob
Franssen	Ann

- De twee meest gebruikte formaten voor flat-file databestanden zijn delimited en fixed-width bestanden.
- Een delimited bestand (vaak ook wel csv-bestand of comma-separated values bestand genoemd):
 - gebruikt voor iedere rij een nieuwe regel,
 - splitst de kolommen op met behulp van een specifiek splitsingsteeken (vaak de komma of de puntkomma),
 - kan het begin en het einde van een karakterstring met behulp van een specifiek quote-teken (vaak ' of ") aanduiden.
- Bovenstaande tabel kan als een delimited databestand opgeslagen worden en ziet er dan als volgt uit:

```
naam;voornaam
Nelissen;Rob
Franssen;Ann
```

- Een fixed-width bestand gebruikt eveneens een aparte regel per rij, maar gebruikt een vast aantal karakters per kolom en heeft dus geen splitsingstekens, noch quote-teken nodig.
- Bovenstaande tabel kan als een fixed-width databestand opgeslagen worden en ziet er dan als volgt uit:

```
naam      voornaam
Nelissen Rob
Franssen Ann
```

- Het nadeel van een delimited bestand is dat je het splitsings- en quote-teken niet kunt gebruiken in je data.
- Het voordeel van een delimited bestand is dat een veld niet meer ruimte in beslag neemt dan nodig.
- Bestudeer hoofdstuk Data Import van het boek 'R for Data Science' om te weten hoe je in R data uit flat-file bestanden kunt inlezen.

Hiërarchische databestanden

- Het nadeel van flat-file data bestanden is dat de data in een tabelvorm moet passen.
- Indien data een complexere (vaak hiërarchische) structuur heeft, dan is dit niet evident om correct in een tabelvorm te gieten.
 - Je hebt bijvoorbeeld data over de studenten en van iedere student heb je naamgegevens en de resultaten van de verschillende afgelegde vakken.
 - Het aantal afgelegde vakken verschilt echter van student tot student.
 - Ook per student kan het aantal opgenomen kansen per vak verschillen van vak tot vak.
 - Het aantal scores per student kan hierdoor sterk variëren.
- Voor hiërarchische databestanden wordt daarom vaak gebruikt gemaakt van XML-bestanden of JSON-bestanden.
- XML- en JSON-bestanden zijn ook zeer populair om gegevens via het web uit te wisselen.

XML-bestanden

- Bestudeer de bron over XML (tot en met XML attributes) om te begrijpen hoe een XML-bestand is opgebouwd.
- Een XML-bestand bestaat uit XML-elementen.
 - De naam van het XML-element wordt bepaald door het openings- en sluitingslabel.
 - Het openingslabel volgt het formaat `<element-naam>`.
 - Het sluitingslabel heeft dezelfde naam en volgt het formaat `</element-naam>`.
 - Tussen het openings- en sluitingslabel plaatsen we de inhoud van het XML-element.
- Voorbeeld: `<student>Rob Nelissen</student>`.
 - Hier wordt het XML-element student gedefinieerd.
 - De inhoud van dit XML-element is Rob Nelissen.
- De inhoud van een XML-element kan ook bestaan uit andere XML-elementen. Op deze manier kan je volledige tabellen in XML opslaan. Onderstaande voorbeeld is de vertaling van voorgaande data in tabelvorm naar XML.
- Voorbeeld:

```
<studenten>
  <student>
    <naam>Nelissen</naam>
```

```

<voornaam>Rob</naam>
</student>
<student>
  <naam>Franssen</naam>
  <voornaam>Ann</voornaam>
</student>
</studenten>

```

- Zoals blijkt uit de vergelijking tussen de XML-representatie en voorgaande flat-file representaties, bevat een XML-bestand redelijk veel overhead om tabelvorm-data op te slaan.
- De kracht van XML ten opzichte van de flat-file bestanden is echter dat je veel complexere datastructuren kunt opslaan. Onderstaand voorbeeld opslaan in een tabelvorm (en dus flat-files) is allesbehalve evident.
- Voorbeeld:

```

<studenten>
  <student>
    <naam>Nelissen</naam>
    <voornaam>Rob</voornaam>
    <vakken>
      <vak>
        <naam>Exploratieve en Descriptieve Data Analyse</naam>
        <academiejaar>20162017</academiejaar>
        <score_kans1>8</score_kans1>
        <score_kans2>12</score_kans2>
      </vak>
      <vak>
        <naam>Macro-economie</naam>
        <academiejaar>20152016</academiejaar>
        <score_kans1>8</score_kans1>
        <score_kans2>7</score_kans2>
      </vak>
      <vak>
        <naam>Macro-economie</naam>
        <academiejaar>20162017</academiejaar>
        <score_kans1>14</score_kans1>
      </vak>
      ...
    </vakken>
  </student>
  <student>
    <naam>Franssen</naam>
    <voornaam>Ann</voornaam>
  </student>
</studenten>

```

```

<vakken>
  <vak>
    <naam>Exploratieve en Descriptieve Data Analyse</naam>
    <academiejaar>20162017</academiejaar>
    <score_kans1>15</score_kans1>
  </vak>
  <vak>
    <naam>Macro-economie</naam>
    <academiejaar>20162017</academiejaar>
    <score_kans1>16</score_kans1>
  </vak>
  ...
</vakken>
</student>
...
</studenten>

```

JSON-bestanden

- Bestudeer de JSON Tutorial om te begrijpen hoe een JSON-bestand is opgebouwd.
- JSON is een ander formaat dat steeds populairder wordt om hiërarchische data op te slaan en uit te wisselen.
 - In vergelijking met XML is JSON korter en eenvoudiger te lezen.
- Een JSON-bestand bestaat voornamelijk uit JSON-objecten en JSON-lijsten.
- JSON-objecten komen typisch overeen met een observatie (rij) in een dataset.
 - Een JSON-object wordt omsloten door accolades.
 - De inhoud van een JSON-object bestaat uit key-value paren.
 - * De key-value paren zijn van elkaar gescheiden door middel van een komma.
 - * De key is een string omsloten door dubbele aanhalingsstekens en geeft aan wat de value voorstelt.
 - * Key en value zijn van elkaar gescheiden door middel van een dubbelpunt.
- Voorbeelden van een JSON-object dat de student Rob Nelissen voorstelt:
 - {"naam": "Nelissen", "voornaam": "Rob"}
- Een JSON-lijst (array genoemd) bestaat uit een lijst van waarden gescheiden door een komma en omgeven door rechte haakjes.

- De waarden van de verschillende elementen in een JSON-lijst moeten van hetzelfde type zijn.
- Toegelaten waarden zijn o.a. strings, getallen, objecten, andere arrays (lijsten).
- Indien je data in tabelvorm wenst voor te stellen, zal je iedere rij als een JSON-object voorstellen en de volledige tabel als een lijst van deze JSON-objecten.
- Onderstaand voorbeeld is de vertaling van voorgaande studentendata in tabelvorm naar JSON.

```
[  
  {"naam": "Nelissen",  
   "voornaam": "Rob"  
  },  
  {"naam": "Franssen",  
   "voornaam": "Ann"  
  }  
]
```

- Net als bij XML kan je met JSON complexe datastructuren voorstellen.
- Voorbeeld:

```
[  
  {"naam": "Nelissen",  
   "voornaam": "Rob",  
   "vakken":  
     [  
       {"naam": "Exploratieve en Descriptieve Data Analyse",  
        "academiejaar": "20162017",  
        "score_kans1": 8,  
        "score_kans2": 12  
       },  
       {"naam": "Macro-economie",  
        "academiejaar": "20152016",  
        "score_kans1": 8,  
        "score_kans2": 7  
       },  
       {"naam": "Macro-economie",  
        "academiejaar": "20162017",  
        "score_kans1": 14  
       }  
     ]  
  },  
  {"naam": "Franssen",  
   "voornaam": "Ann",  
   "vakken":  
     [  
       {"naam": "Exploratieve en Descriptieve Data Analyse",  
        "academiejaar": "20162017",  
        "score_kans1": 10,  
        "score_kans2": 10  
       },  
       {"naam": "Macro-economie",  
        "academiejaar": "20152016",  
        "score_kans1": 10,  
        "score_kans2": 10  
       },  
       {"naam": "Macro-economie",  
        "academiejaar": "20162017",  
        "score_kans1": 14  
       }  
     ]  
  }]
```

```

"vakken": [
    {
        "naam": "Exploratieve en Descriptieve Data Analyse",
        "academiejaar": "20162017",
        "score_kans1": 15
    },
    {
        "naam": "Macro-economie",
        "academiejaar": "20162017",
        "score_kans1": 16
    }
]
}
]

```

Applicatie-specifieke dataformaten

- Naast deze standaard dataformaten, waarbij data als tekstbestanden worden opgeslagen, bestaan er verschillende applicatie-specifieke dataformaten.
- Het voordeel van applicatie-specifieke dataformaten is dat deze extra informatie over de data kunnen opslaan die specifiek is voor de applicatie.
 - Zo zal een Excel databestand ook informatie bevatten over de formules en opmaak van de data (vet, cursief, ...).
- Het nadeel van deze applicatie-specifieke dataformaten is dat ze niet altijd leesbaar zijn door andere applicaties.

R-packages voor het inladen van diverse dataformaten

- Standaard R heeft een aantal functies om delimited bestanden in te lezen, namelijk `read.csv()` en `read.csv2()`. Het verschil tussen beide functies heeft betrekking op de standaardwaarden voor specifieke parameters, zoals welk teken als delimiter gebruikt wordt (`read.csv` veronderstelt het komma-teken als delimiter, terwijl `read.csv2` er van uitgaat dat de puntkomma gebruikt wordt om waarden van elkaar te scheiden).
- Er is ook het R pacakge ‘`readr`’ dat ook twee soortgelijke functies aanbiedt - `read_csv()` en `read_csv2()` - die performanter zijn dan de standaardfuncties.
- Om xml-bestanden in te lezen, wordt typisch het R package ‘`xml`’ gebruikt. Voor JSON-bestanden kan gebruik gemaakt worden van de packages ‘`rjson`’ of ‘`jsonlite`’.
- Voor een aantal applicatie-specifieke formaten zijn ondertussen ook al R-packages ontwikkeld. Zo is er bijvoorbeeld het R-package

‘readxl’ dat het inladen van Excel bestanden relatief eenvoudig maakt.

7.2.3 Data-encoding

Binair, decimaal en hexadecimaal rekenstelsel

- Een computer kan slechts 2 waarden opslaan, typisch voorgesteld als 0 en 1.
- Iedere opslaglocatie op een computer kan dus slechts 2 verschillende waarden opslaan en wordt een bit genoemd.
 - De afkorting van bit is de kleine letter ‘b’.
- Een rekenstelsel waarbij iedere locatie slechts 2 waarden kan voorstellen noemen we een binair rekenstelsel.
 - Indien we 2 opslaglocaties (2 bits) gebruiken, kunnen we 4 verschillende waarden opslaan: 00, 01, 10 en 11.
 - Indien we 3 bits gebruiken zijn er 8 mogelijke waarden, bij 4 bits zijn er 16 mogelijke waarden.
 - Het aantal waarden dat men met n bits kan opslaan is gelijk aan 2^n .
- Merk op dat in het rekenstelsel dat door mensen gebruikt wordt iedere opslaglocatie 10 verschillende waarden gebruikt kunnen worden (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
 - Dit noemen we het decimaal rekenstelsel.
 - Met 2 opslaglocaties kunnen we in het decimaal rekenstelsel 100 ($= 10^2$) waarden opslaan: van 00 tot 99.
- Een ander rekenstelsel dat vaak gebruikt wordt binnen computerwetenschappen is het hexadecimaal rekenstelsel.
 - In dit rekenstelsel kan iedere opslaglocatie 16 waardes opslaan, nl. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E en F.
 - Het hexadecimaal rekenstelsel is interessant omdat 1 locatie overeenstemt met 4 bit (4 locaties in het binair rekenstelsel).
- Om in computerprogramma’s aan te geven dat iets voorgesteld wordt in het hexadecimaal stelsel, laten we het voorafgaan door een 0x. Om aan te geven dat iets voorgesteld wordt in het binair stelsel, gebruiken we prefix 0b. Zonder prefix verwijzen we typisch naar het decimaal rekenstelsel.

decimaal	hexadecimaal	binair
0	0x0	0b0000
1	0x1	0b0001

decimaal	hexadecimaal	binair
2	0x2	0b0010
3	0x3	0b0011
4	0x4	0b0100
5	0x5	0b0101
6	0x6	0b0110
7	0x7	0b0111
8	0x8	0b1000
9	0x9	0b1001
10	0xA	0b1010
11	0xB	0b1011
12	0xC	0b1100
13	0xD	0b1101
14	0xE	0b1110
15	0xF	0b1111

Tabel: conversietabel decimaal, hexadecimaal en binair.

- 8 bits worden ook een byte genoemd wat afgekort wordt met de hoofdletter B. 1B bestaat dus uit 8b en kan dus 256 ($= 2^8$) waarden opslaan.

Tekst opslaan

- Bestudeer de bron over Encoding tot sectie ‘Encodings en PHP’.
- Lees de bron over de geschiedenis van ASCII (enkel sectie ‘A Historical Perspective’).
- Aangezien computers enkel bits kunnen opslaan, hebben we een conversieschema nodig om tekst op te slaan. Ieder letterteken zal moeten omgezet worden naar een string van bits. Dit conversieschema wordt een ‘encoding scheme’ of encoderingsschema genoemd.
- Ieder encoderingsschema voorziet de vertaling van een specifieke set van karakters naar bijhorende bitstrings. Deze sets van karakters noemen we ‘character sets’ of karaktersets.
- Er bestaan zeer veel encoderingsschema’s.
 - ASCII is een van de oudste encoderingsschema’s en is voor- namelijk bruikbaar voor Engelstalige tekst.
 - * De ASCII karakterset bestaat uit 128 lettertekens en bevat o.a. de cijfers 0 tot 9, de letters a-z en A-Z.
 - * ASCII gebruikt 1 byte per letterteken en kan dus in principe 256 verschillende lettertekens opslaan.

- * Aangezien ASCII slechts een karakterset van 128 tekens heeft, gebruikt het dus slechts 7 bit van de beschikbare byte ($2^7 = 128$).
- * Omdat de ASCII tekenset gemaakt was voor de Engelse taal ontbreken er verschillende tekens voor andere talen.
- De ANSI-standaard nam de ASCII tekenset over, maar voegt hier vervolgens 128 tekens aan toe door de volledige byte te gebruiken.
 - * Met welke tekens de karakterset wordt uitgebreid ligt echter niet vast binnen de ANSI-standaard, maar is afhankelijk van de gekozen codepage (of karakterset).
 - * Er bestaan zeer veel ANSI codepages (die eigenlijk Windows codepages genoemd moeten worden). Voor de eerste 128 tekens maakt de specifieke codepage niet uit, maar voor de laatste 128 code pages is dit wel belangrijk.
- Voor talen waar 1 byte per letterteken onvoldoende is, werden dan weer nieuwe encoderingsschema's gebruikt die 2 bytes gebruiken en zo 65536 lettertekens kunnen voorstellen.
- Unicode is een poging om tot 1 karakterset te komen voor alle tekens die gebruikt worden in tekst.
 - * Unicode is een standaard en definieert zelf geen encoding. Ze vertaalt dus zelf geen lettertekens naar bitstrings.
 - * Unicode legt wel codepoints vast, wat een mapping is tussen lettertekens en een hexadecimaal getal. Zo is de letter 'A' gekoppeld aan het codepoint 0x0041.
 - * Het Unicode systeem bevat in totaal meer dan 1 miljoen codepoints en omvat niet enkel cijfers en letters, maar ook emoji's. Zo heeft de 'lachend gezicht'-emoji codepoint 0x1F642.
 - * De feitelijke omzetting van de codepoints naar een bitstring gebeurt door een specifieke encoding, waarbij UTF-8 de meest voorkomende is.
- Het gevolg is dat we een grote waaier aan encoderingsschema's hebben.
 - Als je dus een tekst opslaat volgens het ene encoderingsschema en vervolgens terug inleest volgens een ander encoderingsschema, dan kan het zijn dat delen van de tekst geen steek meer houden.
 - Als je bijvoorbeeld de letter 'i' opslaat volgens de Windows-1252 codepage dan zal dit binair als 11101111 opgeslagen worden (0xEF). Als je deze reeks van 8 bits echter later weer inleest volgens de Windows-1257 (Windows-Baltic), dan zal de binaire reeks 11101111 geïnterpreteerd worden als 'l'.
- Het R-package 'readr' gaat er van uit dat tekst geëncodeerd is in UTF-8.

7.2.4 Datatypes controlleren en corrigeren

- We zullen de datavoorbereidingsfase illustreren aan de hand van de vluchtgegevens van de drie luchthavens in New York City.
- Voor we kunnen beginnen is het altijd verstandig een snel overzicht te maken van de dataset, zodat we weten welke variabelen we voor handen hebben alsook hun datatypes.
- Met de *glimpse* functie krijg je snel een overzicht van de verschillende variabelen en van welk type ze zijn.

```
glimpse(df)
```

```
## Rows: 329,174
## Columns: 7
## $ luchthaven      <fct> EWR, LGA, JFK, LGA, EWR, LGA, JFK, LGA, JF...
## $ maatschappij    <fct> United Air Lines Inc., United Air Lines Inc., A...
## $ vertrek_vertraging <dbl> 2, 4, 2, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2...
## $ aankomst_vertraging <dbl> 11, 20, 33, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## $ afstand          <chr> "1400", "1416", "1089", "762", "719", "1065", "...
## $ vliegtijd         <dbl> 227, 227, 160, 116, 150, 158, 53, 140, 138, 149...
## $ vluchtttype       <ord> normaal, normaal, kort, kort, kort, kort, ...
```

- Deze output bevat al een opmerkelijk resultaat. Zo zien we dat de variabele ‘afstand’ als tekst is opgeslagen in plaats van als een continue (numerieke) variabele.
- Soms gebeurt het dat R niet het juiste variablename herkent. Zo kan het zijn dat een categorische variabele als numerieke variabele wordt beschouwd omdat de categorieën gehele getallen zijn (vb. aantal cylinders: 4, 6 of 8).
- In onze dataset hebben we opgemerkt dat de variabele ‘afstand’ niet als numerieke variabele wordt geïnterpreteerd, maar als een ‘tekst’-variabele. Dit kan verschillende oorzaken hebben.
 - Zo kan het zijn dat er voor 1 van de observaties een waarde geregistreerd is met een niet-numeriek teken (vb. 1OO ipv 100). In dat geval zal je eerst deze waarden moeten corrigeren naar de juiste waarde.
 - Een andere vaak voorkomende oorzaak is dat R een punt als decimaalteken verwacht, terwijl dat een komma is in de dataset. Dit valt vaak op te lossen door de data met andere opties in te lezen in R.
- Indien de fouten zijn gecorrigeerd, dan moeten we nog altijd de data omzetten van een ‘tekst’-variabele naar een numerieke variabele (in ons geval). Dit doen we door middel van de ‘mutate’-functie en de ‘as.numeric’-functie.

```
df <- df %>%
  mutate(afstand = as.numeric(afstand))
```

- Merk op dat als er toch nog een waarde aanwezig is die niet kan omgezet worden naar het nieuwe variabeletype, R een waarschuwing zal geven en de waarde zal vervangen door ‘NA’. In dat geval ga je eerst moeten zoeken naar de oorzaak van de waarschuwing, deze aanpakken en dan de variabele transformeren naar het nieuwe type.
- De belangrijkste datatypes in R en de bijhorende transformatiefuncties zijn:
 - numeric (decimale getallen) - as.numeric()
 - integer (gehele getallen) - as.integer()
 - character (tekst) - as.character()
 - factor (nominale variabele) - as.factor()
 - ordered factor (ordinale variabele) - as.ordered()

7.3 Dataproblemen identificeren en corrigeren

7.3.1 Overzicht

- We onderscheiden drie soorten problemen die kunnen opduiken met data en die best op voorhand gecorrigeerd worden:
 - Foutieve waarden.
 - Ontbrekende waarden.
 - Inconsistente waarden.

7.3.2 Foutieve waarden

Categorische variabele

- Coderingsfouten bij categorische variabelen uiten zich typisch in redundante categorielabels. Dit zijn labels met een typfout die door R als een aparte categorie worden beschouwd, maar dit niet zijn.
- Om dit soort coderingsfouten te detecteren, moet je de verschillende labels van een categorische variabele bestuderen.
 - Omdat deze foute categorielabels meestal uitzonderlijk zijn, kan je best de verschillende categorielabels bekijken volgens stijgende frequentie.
 - Een andere aanpak is de categorielabels alfabetisch te ordenen.
- Eenmaal men deze coderingsfouten gedetecteerd heeft, kan men ze manueel corrigeren door gebruik te maken van de functies *mutate* (dplyr) en *fct_recode* (forcats).

Case: Vluchtdata NYC

- Als we de categorische variabele *maatschappij* analyseren op foutieve labels dan zien we dat 1 vlucht foutief het label ‘Xpress-Jet Airlines Inc.’ heeft gekregen in plaats van ‘ExpressJet Airlines Inc.’.

```
df %>%
  group_by(maatschappij) %>%
  tally() %>%
  arrange(n)
```

maatschappij	n
XpressJet Airlines Inc.	1
Envoi Air	19
SkyWest Airlines Inc.	32
Hawaiian Airlines Inc.	342
Mesa Airlines Inc.	601
Frontier Airlines Inc.	685
Alaska Airlines Inc.	714
AirTran Airways Corporation	3260
Virgin America	5162
Southwest Airlines Co.	12275
Endeavor Air Inc.	18460
US Airways Inc.	20536
Envoy Air	26378
American Airlines Inc.	31327
Delta Air Lines Inc.	46779
JetBlue Airways	50940
ExpressJet Airlines Inc.	54172
United Air Lines Inc.	57491

Table 7.3: Maatschappijen geordend volgens stijgende frequentie.

- Indien we de labels van de categorische variabele *maatschappij* alfabetisch ordenen dan zien we ook dat er een aantal vluchten foutief gecodeerd zijn als ‘Envoi Air’ in plaats van ‘Envoy Air’.

```
df %>%
  group_by(maatschappij) %>%
  tally() %>%
  arrange(maatschappij)
```

- We kunnen deze foutieve labels corrigeren met behulp van de functie `fct_recode` uit het `forcats` package.

Table 7.4: Maatschappij alfabetisch geordend.

maatschappij	n
AirTran Airways Corporation	3260
Alaska Airlines Inc.	714
American Airlines Inc.	31327
Delta Air Lines Inc.	46779
Endeavor Air Inc.	18460
Envoi Air	19
Envoy Air	26378
ExpressJet Airlines Inc.	54172
Frontier Airlines Inc.	685
Hawaiian Airlines Inc.	342
JetBlue Airways	50940
Mesa Airlines Inc.	601
SkyWest Airlines Inc.	32
Southwest Airlines Co.	12275
United Air Lines Inc.	57491
US Airways Inc.	20536
Virgin America	5162
XpressJet Airlines Inc.	1

```
df %>%
  mutate(maatschappij = fct_recode(maatschappij,
                                    "ExpressJet Airlines Inc." = "XpressJet Airlines Inc.",
                                    "Envoy Air" = "Envoi Air")) -> df

df %>%
  group_by(maatschappij) %>%
  tally() %>%
  arrange(maatschappij)
```

Ordinale variabelen

- Bij ordinale variabelen kunnen dezelfde coderingsfouten voorkomen als bij categorische variabelen. Deze worden op dezelfde manier gedetecteerd en gecorrigeerd.
- Er is echter nog een bijkomende coderingsfout voor ordinale variabelen, namelijk wanneer de voorgedefinieerde volgorde tussen de labels fout is.
- Om dit te detecteren, moet je de verschillende labels ('levels') opvragen met behulp van de *unique*-functie.
- Indien we de ordinale variabele *vluchtype* analyseren dan zien we dat de voorgedefinieerde volgorde van de labels foutief is.

```
unique(df$vluchtype)
## [1] normaal      kort        lang       intercontinentaal
```

```
## Levels: lang < kort < normaal < intercontinentaal

• We kunnen de volgorde tussen de labels van een ordinale variabele corrigeren met behulp van de functies mutate (dplyr) en fct_relevel (forcats).

df %>%
  mutate(vluchttype = fct_relevel(df$vluchttype, "lang", after = 2)) -> df

unique(df$vluchttype)

## [1] normaal          kort            lang           intercontinentaal
## Levels: kort < normaal < lang < intercontinentaal
```

Continue variabelen

- Foutieve waarden bij een continue variabelen detecteren is een stuk moeilijker omdat een foutieve waarde nog steeds een geldige waarde kan zijn (nog steeds een getal).
- Ook de aanpak om naar weinig voorkomende waarden te kijken, zoals bij categorische variabelen, werkt niet goed omdat bij een continue variabele vaak veel waarden zijn zeer weinig voorkomen.
- De meest voor de hand liggende aanpak is de waarden te bestuderen die opmerkelijk hoog of laag zijn in vergelijking met de andere waarden van de variabele.
- Het is belangrijk te beseffen dat niet iedere extreme waarde per definitie een foutieve waarde is. Uitzonderlijk hoge of lage waardes zijn natuurlijk altijd mogelijk.
- Daarom moet men altijd voorzichtig te werk gaan bij het bepalen of iets een foutieve waarde is (meetfout, ingavefout) of een uitzonderlijke doch correcte waarde. Domeinkennis kan hierbij helpen.
- Indien je door te kijken naar de uiterste waarden mogelijke problemen hebt gedetecteerd, moet je deze observaties van nabij bestuderen om te achterhalen of het meetfouten kunnen zijn of niet. Ga hiervoor steeds naar de volledige observatie kijken en niet enkel naar de waarde voor de continue variabele.
- Een andere manier om te detecteren of een continue variabele uitzonderlijke waarden bevat, is door middel van een boxplot. Uitzonderlijk grote/kleine waarden vallen buiten de ‘whiskers’ en worden door punten aangeduid in een boxplot. Let wel op, de filosofie achter uitzonderlijke waarden is gebaseerd op een normale verdeling van de data. Indien de data werkelijk normaal verdeeld is, dan is de kans op een uitzonderlijke waarde slechts 0.7%. Dit betekent echter dat men best op voorhand het histogram bekijkt om te controleren of de data enigszins normaal verdeeld is, alvorens de boxplot te hanteren.

- Bij foutieve waarden van een continue variabele is het vaak niet mogelijk om de correcte waarde af te leiden (zoals bij een categorische variabele). Daarom is de enige juiste correctie deze foutieve waarden te vervangen met “missing values”.
- We zullen de variabele *vliegtijd* analyseren op foutieve waarden.
- We zullen eerst de kleinste waarden bestuderen. Hiervoor selecteren we de 10 vluchten met de kortste vliegtijd en rangschikken deze volgens stijgende vliegtijd. We kijken hierbij niet alleen naar de vliegtijd, maar ook naar de luchthaven, de maatschappij en de afgelegde afstand.

```
df %>%
  arrange(vliegtijd) %>%
  select(luchthaven, maatschappij, afstand, vliegtijd) %>%
  filter(row_number()<11)
```

luchthaven	maatschappij	afstand	vliegtijd
JFK	Endeavor Air Inc.	94	0.5833333
EWR	JetBlue Airways	1065	2.7666667
JFK	Delta Air Lines Inc.	2248	5.1500000
EWR	ExpressJet Airlines Inc.	116	20.0000000
EWR	ExpressJet Airlines Inc.	116	20.0000000
EWR	ExpressJet Airlines Inc.	116	21.0000000
EWR	ExpressJet Airlines Inc.	80	21.0000000
EWR	ExpressJet Airlines Inc.	116	21.0000000
EWR	ExpressJet Airlines Inc.	80	21.0000000
LGA	US Airways Inc.	184	21.0000000

Table 7.5: Vluchten met kortste vliegtijd.

- Deze analyse doet vermoeden dat de eerste drie vluchten waarschijnlijk meetfouten zijn. Het betreffen hier drie vluchten van minder dan 6 minuten wat zeer onwaarschijnlijk is, zeker wanneer we zien dat de tweede en derde vlucht lange vluchten zijn.
- De overige vluchten zijn vluchten van 20 minuten of meer, maar aangezien het hier om korte vluchten gaan is dit mogelijk correct. We zullen enkel de eerste drie observaties (met een vliegtijd kleiner dan 6) als foutief beschouwen.
- We bestuderen vervolgens de vluchten met de grootste vliegtijd.

```
df %>%
  arrange(-vliegtijd) %>%
  select(luchthaven, maatschappij, afstand, vliegtijd) %>%
  filter(row_number()<11)
```

luchthaven	maatschappij	afstand	vliegtijd
EWR	United Air Lines Inc.	4963	695
JFK	Hawaiian Airlines Inc.	4983	691
JFK	Hawaiian Airlines Inc.	4983	686
JFK	Hawaiian Airlines Inc.	4983	686
JFK	Hawaiian Airlines Inc.	4983	683
JFK	Hawaiian Airlines Inc.	4983	679
EWR	United Air Lines Inc.	4963	676
JFK	Hawaiian Airlines Inc.	4983	676
JFK	Hawaiian Airlines Inc.	4983	675
EWR	United Air Lines Inc.	4963	671

Table 7.6: Vluchten met langste vliegtijd.

- Deze resultaten doen vermoeden dat het hier NIET om meetfouten gaat. Het gaat hier immers om zeer verre vluchten en de maatschappijnaam doet vermoeden dat het hoofdzakelijk vluchten naar Hawaï zijn. We hebben daarom via Google opgezocht hoe lang een vlucht van New York naar Hawaï duurt en dit komt overeen met de vliegtijden van 11 tot 12u in deze dataset. Daarom besluiten we dat deze waarden geen foutieve waarden zijn.
- Vervolgens zullen we voor de drie vluchten met een vliegtijd van minder dan 6 minuten de waarde van de vliegtijd vervangen door een ontbrekende waarde. In R wordt dit aangegeven door de waarde *NA* dat voor ‘not available’ staat.

```
df <- df %>%
  mutate(vliegtijd = ifelse(vliegtijd<6,NA,vliegtijd))
```

```
df %>%
  arrange(vliegtijd) %>%
  select(luchthaven, maatschappij, afstand, vliegtijd) %>%
  filter(row_number()<11)
```

luchthaven	maatschappij	afstand	vliegtijd
EWR	ExpressJet Airlines Inc.	116	20
EWR	ExpressJet Airlines Inc.	116	20
EWR	ExpressJet Airlines Inc.	116	21
EWR	ExpressJet Airlines Inc.	80	21
EWR	ExpressJet Airlines Inc.	116	21
EWR	ExpressJet Airlines Inc.	80	21
LGA	US Airways Inc.	184	21
JFK	Endeavor Air Inc.	94	21
EWR	ExpressJet Airlines Inc.	116	21
EWR	ExpressJet Airlines Inc.	116	21

Table 7.7: Vluchten met kortste vliegtijd.

7.3.3 Ontbrekende waarden

- Soms gebeurt het dat voor bepaalde observaties waarden ontbreken voor een specifieke variabele. In zulke gevallen spreken we van ontbrekende waarden of missing values.
- Het detecteren van ontbrekende waarden is relatief eenvoudig, omdat deze normaal als *NA* gecodeerd zijn in een dataset (*NA* = ‘not available’).
- We kunnen onderscheid maken tussen drie soorten van ontbrekende waarden.
 - Missing completely at random (MCAR): Indien het ontbreken van waarden voor een specifieke variabele volledig willekeurig is, dan spreekt men over MCAR.
 - Missing at random (MAR): Indien het ontbreken van waarden voor variabele X_1 niet willekeurig is, maar afhankelijk van de waarden van andere variabelen X_2, X_3, \dots , dan spreekt men over MAR.
 - Not missing at random (NMAR): Indien het ontbreken van waarden voor variabele X_1 niet willekeurig is, maar afhankelijk is van de waarde van X_1 of van de waarden van ongeobserveerde variabelen, dan spreekt men over NMAR.
- Om te bepalen of data al dan niet MCAR is, moet men achterhalen of het ontbreken van waarden voor variabele X_1 gecorreleerd is met de waarden van een andere variabele X_2 . Een mogelijkheid is om de dataset in twee te splitsen, i.e. alle observaties met een waarde voor X_1 en alle observaties met een missing value voor X_1 . Vervolgens kijken we naar de verdeling van variabele X_2 . Indien deze hetzelfde is voor beide datasets, dan suggereert dit dat er geen relatie bestaat tussen de waarde van X_2 en het al dan niet ontbreken van de waarde voor X_1 . Indien deze verdeling van X_2 sterkt verschilt tussen beide datasets, dan is er mogelijk wel een relatie en dan is de data niet MCAR.
- Het soort ontbrekende waarde heeft belangrijke implicaties hoe je correct met ontbrekende waarden kan omgaan in het kader van confirmatorische data analyse. Zo zal het ‘weglaten’ van observaties met ontbrekende waarden enkel in het geval van MCAR geen vertrekking geven in de resultaten van een confirmatorische data analyse.
- In het kader van een descriptieve of exploratieve data analyse, zijn de implicaties eerder beperkt, omdat men toch enkel uitspraken wenst te doen voor de beschikbare data.
- Wel kan het identificeren van het type ontbrekende waarden op zich interessante inzichten geven. Zo kan het het patroon dat het jaarsalaris voornamelijk ontbreekt bij mensen die hogere studies

gevolgd hebben op zich ook interessant zijn voor verdere interpretatie.

- In descriptieve en exploratieve analyses zijn er 3 manieren om met ontbrekende waarden om te gaan:
 - We verwijderen de variabele waarvoor we missing values hebben.
 - We verwijderen de observaties met missing values.
 - We beschouwen de missing values als een aparte waarde.
- Het verwijderen van de variabele zelf is een drastische maatregel.
Dit betekent immers dat we de variabele volledig buiten beschouwing laten in onze analyse. Dit is vaak het laatste redmiddel en wordt enkel toegepast als er een te hoog percentage ontbrekende waarden is.
- Bij het verwijderen van observaties moet men met de nodige aandacht te werk gaan. Indien de ontbrekende waarden MAR zijn (en niet MCAR), dan gaat men mogelijk waardevolle patronen tussen andere variabelen ook verwijderen. Een mogelijke manier om dit te omzeilen is de observaties met ontbrekende waarden te negeren bij analyses van de variabelen waarvoor de waarden ontbreken. Dit zorgt ervoor dat deze observaties wel nog beschikbaar zijn voor de analyse van andere variabelen.
- Indien de data suggereert dat de ontbrekende waarden MCAR zijn, dan kan men overwegen deze observaties te verwijderen. Indien dit niet het geval is, dan is het beter deze NA-waarden als een aparte categorie te beschouwen.
 - Omdat R de waarde ‘NA’ anders behandelt dan reguliere waarden, is het vaak aangeraden om deze waarde te transformeren (indien je de ontbrekende waarden als een aparte categorie wenst te beschouwen).
 - In geval van een categorische variabele, kan je de ‘NA’ waarde transformeren naar een aparte categorie (vb ‘waarde ontbreekt’).
 - In geval van een continue variabele, is het aangeraden een nieuwe categorische variabele aan te maken die aangeeft of er wel of niet een waarde aanwezig was voor de continue variabele.
- De eerste stap is na te gaan welke variabelen ontbrekende waarden hebben en hoe vaak deze variabelen ontbrekende waarden hebben.
Dit functie summary() is hiervoor zeker nuttig.

```
summary(df)
```

```
##   luchthaven           maatschappij vertrek_vertraging
##   EWR:119282    United Air Lines Inc. :57491   Min.   : -43.00
##   JFK:105230    ExpressJet Airlines Inc.:54173   1st Qu.:  -5.00
##   LGA:104662    JetBlue Airways       :50940   Median : -2.00
```

```

##          Delta Air Lines Inc.    :46779   Mean    : 12.71
##          American Airlines Inc. :31327   3rd Qu.: 11.00
##          Envoy Air            :26397   Max.    :1301.00
##          (Other)              :62067   NA's     :8214
##  aankomst_vertraging      afstand      vliegtijd                  vluchtype
##  Min.    : -86.000   Min.    : 17   Min.    : 20.0   kort             :245666
##  1st Qu.: -17.000   1st Qu.: 502  1st Qu.: 81.0   normaal        : 31813
##  Median  : -5.000    Median : 820  Median  :127.0   lang            : 50980
##  Mean    :  6.987    Mean   :1027  Mean    :149.6   intercontinentaal:  715
##  3rd Qu.: 14.000    3rd Qu.:1372 3rd Qu.:184.0
##  Max.    :1272.000   Max.   :4983  Max.    :695.0
##  NA's    :9365       NA's    :9368  NA's    :9368

```

- Uit deze analyse blijkt dat de variabelen *vertrek_vertraging*, *aankomst_vertraging* en *vliegtijd* last hebben van ontbrekende waarden.
- Om vervolgens te analyseren of deze ontbrekende waarden MCAR zijn of niet, zullen we voor ieder van de drie continue variabelen een nieuwe categorische variabele maken die aangeeft of de waarde ontbreekt of niet.

```

df <- df %>%
  mutate(vertrek_vertraging_na = is.na(vertrek_vertraging),
        aankomst_vertraging_na = is.na(aankomst_vertraging),
        vliegtijd_na = is.na(vliegtijd))

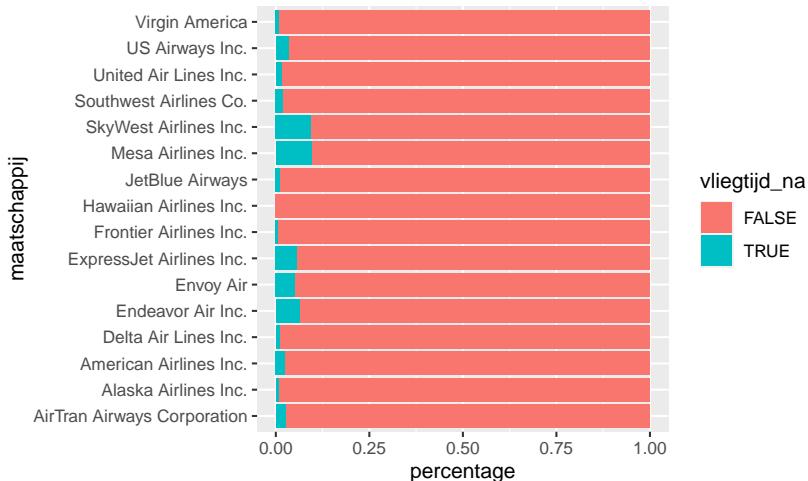
```

- Nu kunnen we met ggplot achterhalen of de andere variabelen zich ‘anders’ gedragen als er voor één van deze drie variabelen een waarde ontbreekt.
- We illustreren voor ‘afstand’ (continu) en voor ‘maatschappij’ (categorisch).
- Indien we dit bestuderen voor ‘maatschappij’ gaan we voor iedere maatschappij laten zien welk percentage cases een ontbrekende waarde voor *vliegtijd* heeft. Indien er geen verband is, dan zouden we geen grote verschillen mogen zien tussen de maatschappijen.

```

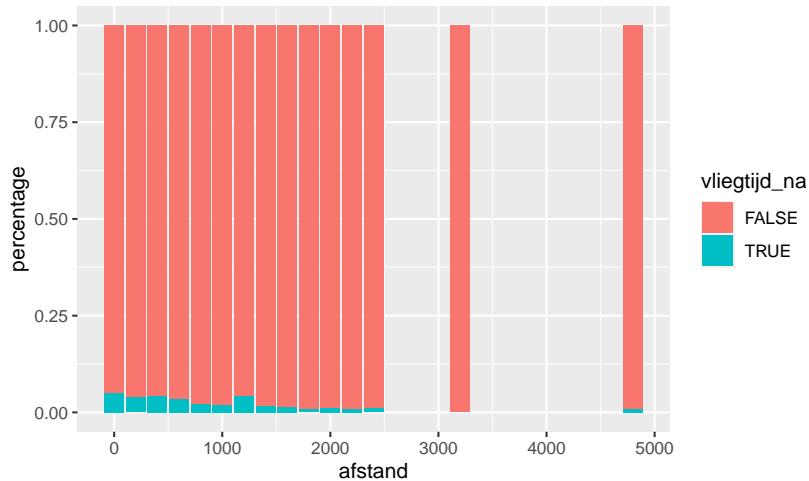
df %>%
  group_by(maatschappij, vliegtijd_na) %>%
  summarise(aantal = n())%>%
  ungroup() %>%
  group_by(maatschappij) %>%
  mutate(totaal = sum(aantal), percentage = aantal/totaal) %>%
  ggplot(aes(x=maatschappij, y=percentage)) +
  coord_flip() +
  geom_col(aes(fill=vliegtijd_na), position = "stack")

```



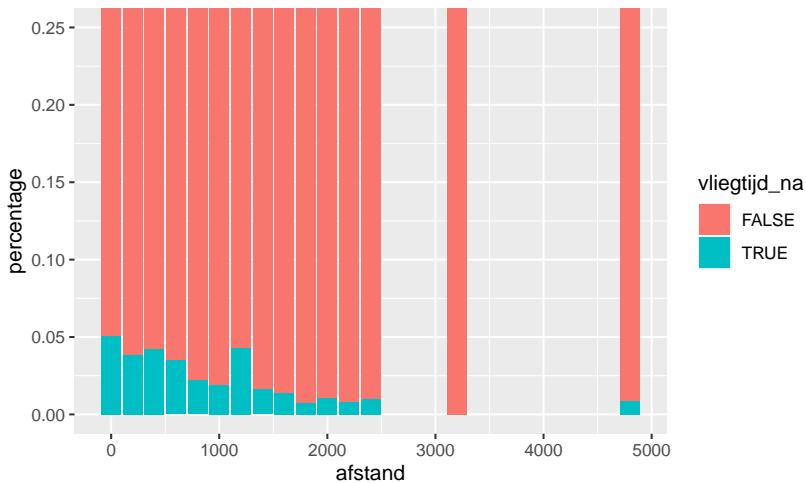
- Uit deze resultaten blijkt dat voor sommige maatschappijen een aanzienlijk hoger percentage ontbrekende waarden bij vliegtijd voorkomt (SkyWest, Mesa en ook ExpressJet, Envoy en Endeavor).
- We kunnen een soortgelijke analyse ook uitvoeren voor de variabele *afstand*. Hiervoor zullen we eerst de variabele *afstand* omvormen tot een categorische variabele.

```
binwidth <- 200
df %>%
  mutate(afstand_cat = afstand %% binwidth) %>%
  group_by(afstand_cat, vliegtijd_na) %>%
  summarise(aantal = n()) %>%
  ungroup() %>%
  group_by(afstand_cat) %>%
  mutate(totaal = sum(aantal), rel_aantal = aantal/totaal) %>%
  ggplot(aes(x=afstand_cat*binwidth, y=rel_aantal)) +
  geom_col(aes(fill=vliegtijd_na), position = "stack") +
  xlab("afstand")+
  ylab("percentage")
```



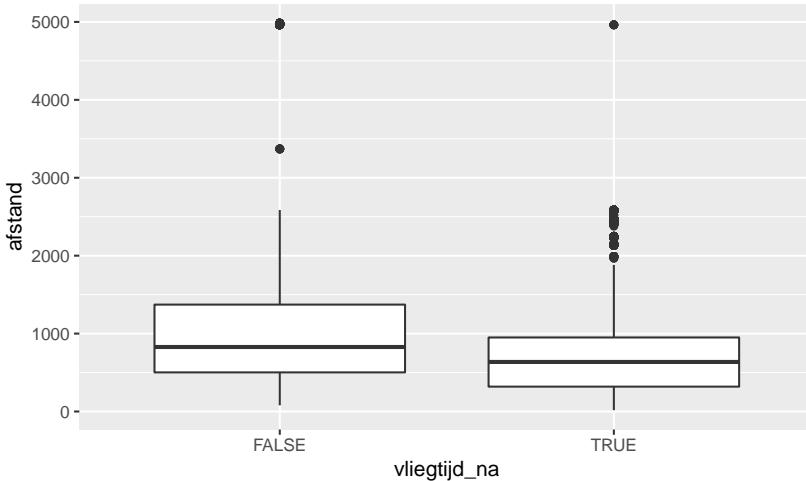
- Deze resultaten lijken te suggereren dat naarmate de vlucht langer wordt, het percentage ontbrekende waarden bij *vliegtijd* afneemt (met een uitzonderlijke piek bij vluchten rond 1200 mijl).
- Omdat het percentage ontbrekende waarden eerder klein is, is het moeilijk om het patroon duidelijk te zien. We kunnen ook dezelfde plot maken, maar de y-as laten stoppen bij een waarde van 0.25. Op deze manier wordt het patroon duidelijker.

```
binwidth <- 200
df %>%
  mutate(afstand_cat = afstand %% binwidth) %>%
  group_by(afstand_cat, vliegtijd_na) %>%
  summarise(aantal = n()) %>%
  ungroup() %>%
  group_by(afstand_cat) %>%
  mutate(totaal = sum(aantal), rel_aantal = aantal/totaal) %>%
  ggplot(aes(x=afstand_cat*binwidth, y=rel_aantal)) +
  geom_col(aes(fill=vliegtijd_na), position = "stack") +
  xlab("afstand")+
  ylab("percentage")+
  coord_cartesian(ylim = c(0, 0.25))
```



- Tenslotte kunnen we ook nog op een andere manier het verband tussen de *afstand* en het voorkomen van ontbrekende waarden bij *vliegtijd* bestuderen, nl. via 2 boxplots.

```
df %>%
  ggplot(aes(x=vliegtijd_na, y=afstand)) +
  geom_boxplot()
```



- Hier zien we dat vluchten waarvoor de vliegtijd ontbreekt vaak kortere vluchten zijn dan waarvoor we de vliegtijd wel hebben. Dit komt overeen met de vorige bevinding.
- Op basis van deze resultaten kunnen we dus stellen dat het ontbreken van de vliegtijd niet willekeurig is, maar vaker voorkomt bij bepaalde maatschappijen en eerder bij kortere dan bij langere vluchten.
- We zouden nog verder kunnen onderzoeken of deze maatschappijen eerder langere of kortere vluchten organiseren.

- Soortgelijke analyses kunnen we ook uitvoeren voor de variabelen *vertrek_vertraging* en *aankomst_vertraging*.

7.3.4 Inconsistente waarden

- Data is inconsistent als het niet voldoet aan een aantal regels/beperkingen die horen te gelden op basis van domeinkennis.
- De vorm van inconsistenties waar we ons op focussen, betreft in-record inconsistenties. Dit zijn tegenstrijdigheden die aanwezig zijn binnen één enkele observatie. Enkele voorbeelden zijn:
 - De gemiddelde snelheid van een vlucht ligt hoger dan de maximale theoretische snelheid van het vliegtuig.
 - Het aankomsttijdstip van een vlucht vindt plaats voor het vertrektijdstip.
 - De aankomsttijdstip komt niet overeen met het vertrektijdstip + vertrekvertraging + vluchtduur.
- Het identificeren van inconsistenties kan door middel van diverse dplyr-functies, waarbij je voor iedere observatie test of deze voldoen aan de opgelegde beperkingsregel.
- Daarnaast is er ook het editrules package dat nuttige functies biedt om op een gestructureerdere manier consistentie te evalueren.

7.4 Data opwaarderen

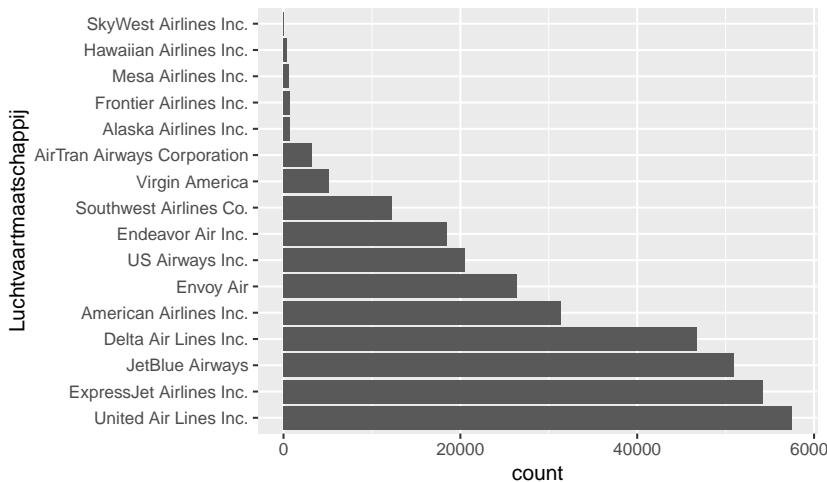
- Van zodra de data geen foutieve en/of ontbrekende waarde meer bevat, kunnen we een aantal technieken toepassen om de data bruikbaarder te maken voor exploratieve analyses. We onderscheiden hierbij 2 technieken:
 - Transformatie van bestaande variabelen.
 - Selectie van observaties.

Categorische variabelen

- Soms is het beter om de categorieën van een categorische variabelen te wijzigen door sommige categorieën samen te nemen. Er zijn verschillende situaties waarbij dit het overwegen waard is, zoals:
 - De labels van een categorische variabele is op een te gedetailleerd niveau gedefinieerd, met als gevolg dat de exploratieve analyse al snel complex wordt door de vele categorieën. In zulke gevallen kan het zinvol zijn om het aantal categorieën te verminderen door categorieën die inhoudelijk bij elkaar horen samen te nemen.

- Een categorische variabele bestaat uit een beperkt aantal categorieën met veel observaties en een groot aantal categorieën met zeer weinig observaties. In zulke gevallen kan het zinvol zijn om de categorieën met weinig observaties samen te nemen in 1 categorie “Overige”.
- Om te bepalen welke categorieën men kan samenvoegen, kan een frequentietabel of barplot gemaakt worden.
- Het herdefiniëren van de labels gebeurt vervolgens met de functie `fct_recode` (forcats). Hierbij heeft men steeds de keuze om de oorspronkelijke variabele te vervangen of een nieuwe variabele aan te maken.
- Laten we eens aan de hand van een barplot naar de variabele ‘luchtvaartmaatschappij’ kijken. We zien hierbij dat er relatief veel luchtvaartmaatschappijen (categorieën) in onze data zijn en dat er een aantal verwaarloosbaar weinig vluchten bevatten.

```
df %>%
  ggplot(aes(x=fct_infreq(maatschappij))) +
  geom_bar() +
  coord_flip() +
  xlab("Luchtvaartmaatschappij")
```



- We kunnen de exacte aantallen achterhalen met behulp van een frequentietabel.

```
df %>%
  group_by(maatschappij) %>%
  summarise(n = n()) %>%
  arrange(n)
```

maatschappij	n
SkyWest Airlines Inc.	32
Hawaiian Airlines Inc.	342
Mesa Airlines Inc.	601
Frontier Airlines Inc.	685
Alaska Airlines Inc.	714
AirTran Airways Corporation	3260
Virgin America	5162
Southwest Airlines Co.	12275
Endeavor Air Inc.	18460
US Airways Inc.	20536
Envoy Air	26397
American Airlines Inc.	31327
Delta Air Lines Inc.	46779
JetBlue Airways	50940
ExpressJet Airlines Inc.	54173
United Air Lines Inc.	57491

Table 7.8: Luchtvaartmaatschappijen geordend volgens stijgend aantal vluchten.

- Op basis van deze analyse beslissen we om de luchtvaartmaatschappijen met minder dan 10000 vluchten samen te voegen in een nieuwe categorie met het label “Overige”. We opteren ervoor de oorspronkelijke variabelen te vervangen.

```
df %>%
  mutate(maatschappij = fct_lump(maatschappij,
                                   n = 9,
                                   other_level = "Overige")) -> df
```

- De nieuwe frequentietabel toont het resultaat.

```
df.complete %>%
  group_by(maatschappij) %>%
  summarise(n = n()) %>%
  arrange(n)
```

Continue variabelen

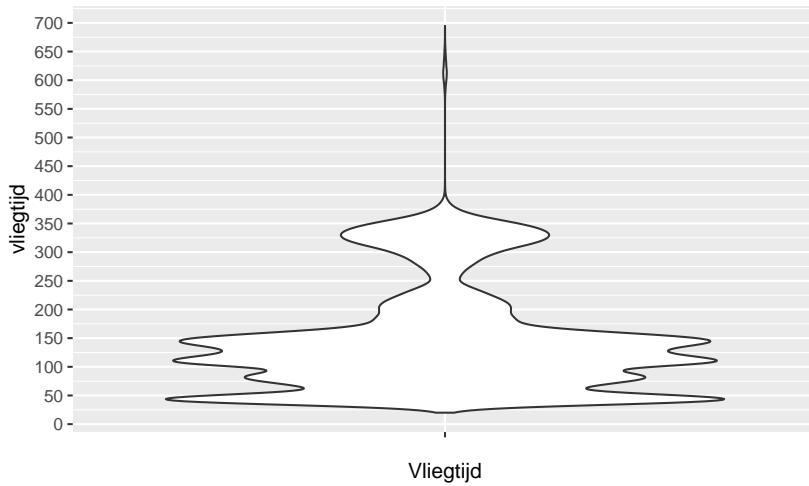
- Bij continue variabelen zijn er verschillende transformaties die regelmatig uitgevoerd worden:
 - De transformatie van een continue variabele naar een categorische variabele.
 - Het herschalen van de continue variabele.
 - De creatie van een nieuwe variabele op basis van bestaande continue variabelen.

maatschappij	n
Overige	10796
Southwest Airlines Co.	12275
Endeavor Air Inc.	18460
US Airways Inc.	20536
Envoy Air	26397
American Airlines Inc.	31327
Delta Air Lines Inc.	46779
JetBlue Airways	50940
ExpressJet Airlines Inc.	54173
United Air Lines Inc.	57491

Table 7.9: Luchtvaartmaatschappijen geordend volgens stijgend aantal vluchten.

- Ook hier hebben we weer steeds de mogelijkheid om de bestaande variabele te vervangen of een nieuwe variabele aan te maken.
- Laten we de variabele vliegtijd eens onder de loep nemen. We beginnen met een visuele analyse aan de hand van een violinplot.

```
df %>%
  ggplot(aes(x="", y=vliegtijd)) +
  geom_violin() +
  xlab("Vliegtijd") +
  scale_y_continuous(breaks=seq(0,800,50))
```



- Op basis van deze plot beslissen we een nieuwe categorische variabele ‘vliegtijd_fct’ aan te maken, waarbij ‘kort’ overeenkomt met een vlucht die minder dan een uur duurt, ‘normaal’ overeenkomt met een vlucht tussen 1 en 4 uur (60-240) en ‘lang’ overeenkomt met een vlucht van meer dan 4 uur. Hiervoor maken we gebruik van de functie *cut*.

```
df %>%
```

```
mutate(vliegtijd_fct = cut(vliegtijd, c(-Inf,60,240,Inf),
                           labels=c('kort','normaal','lang'))) -> df
```

- Aan de hand van een frequentietabel kunnen we nu het resultaat bekijken.

```
df %>%
  group_by(vliegtijd_fct) %>%
  tally()
```

vliegtijd_fct	n
kort	53220
normaal	210758
lang	55828
NA	9368

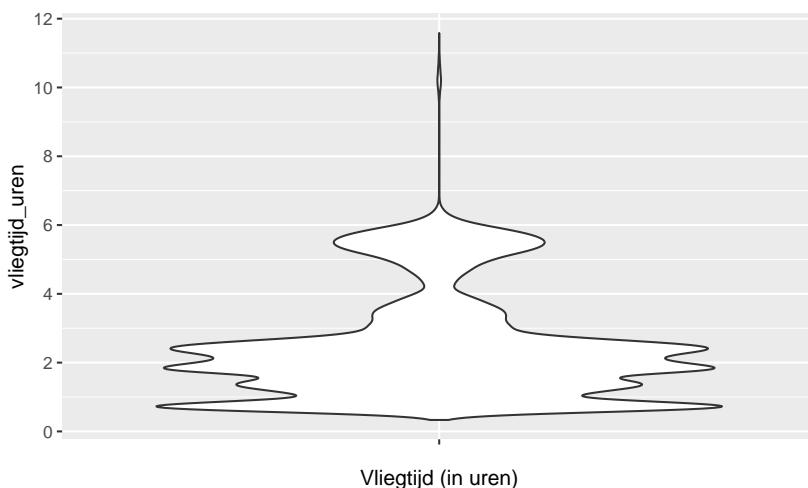
Table 7.10: Frequentietabel vliegtijd (factor)

- Vervolgens beslissen we een nieuwe variabele te maken die de vliegtijd uitdrukt in uren in plaats van minuten.

```
df %>%
  mutate(vliegtijd_uren = vliegtijd/60) -> df
```

- We kunnen het resultaat bekijken met een violinplot.

```
df %>%
  ggplot(aes(x="", y=vliegtijd_uren)) +
  geom_violin() +
  xlab("Vliegtijd (in uren)") +
  scale_y_continuous(breaks=seq(0,12,2))
```

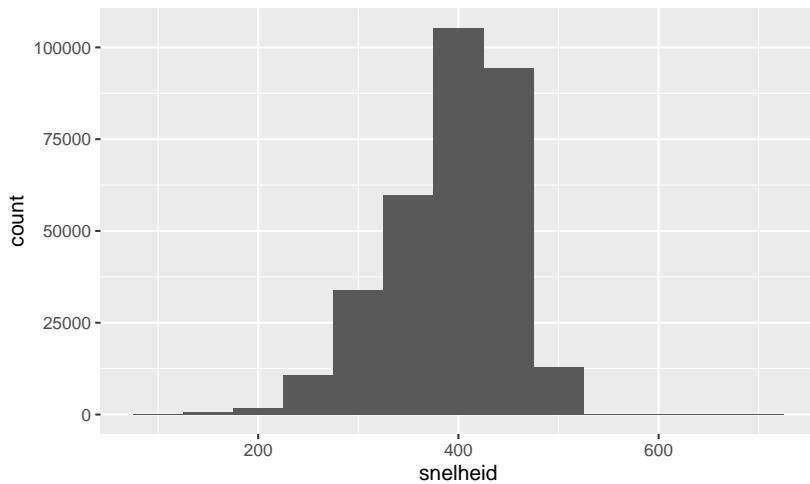


- Tenslotte maken we een nieuwe variabele die de gemiddelde snelheid van het vliegtuig uitdrukt door de afstand te delen door de vliegtijd.

```
df %>%
  mutate(snelheid = afstand / vliegtijd_uren) -> df
```

- Laten we het resultaat aan de hand van een histogram bekijken.

```
df %>%
  ggplot(aes(x=snelheid)) +
  geom_histogram(binwidth = 50) +
  scale_x_continuous(breaks = seq(0,3000, 200))
```



7.4.1 Sampling

- Soms is een dataset zo groot, dat analyses veel tijd in beslag nemen. In zulke gevallen kan het nuttig zijn om een random sample te nemen van de oorspronkelijke data om een eerste exploratieve analyse op uit te voeren.
- Zolang de sample willekeurig getrokken wordt en de nieuwe dataset niet te klein wordt, is de kans dat je patronen ontdekt in de sample die niet voorkomen in de volledige dataset eerder klein.
- Na een eerste exploratieve data analyse op de beperkte sample, kan men vervolgens gerichter de volledige dataset analyseren.
- Laten we een sample van 10000 vluchten nemen uit de oorspronkelijke dataset.

```
df.10000 <- df %>% sample_n(10000)
```

- We kunnen nu een eerste blik op deze sample werpen met behulp van de *summary* functie.

```
summary(df.10000)

##   luchthaven                  maatschappij  vertrek_vertraging
## EWR:3739    United Air Lines Inc.    :1783   Min.   :-19.00
## JFK:3096    ExpressJet Airlines Inc.:1646   1st Qu.: -5.00
## LGA:3165    JetBlue Airways       :1522   Median  : -1.00
##          Delta Air Lines Inc.     :1384   Mean    : 12.89
##          American Airlines Inc.  : 924   3rd Qu.: 10.00
##          Envoy Air             : 831   Max.    :502.00
##          (Other)              :1910   NA's    :253

##   aankomst_vertraging      afstand      vliegtijd           vluchtype
##   Min.   :-65.000      Min.    : 80   Min.   :21.0   kort      :7508
##   1st Qu.:-17.000      1st Qu.: 502  1st Qu.:82.0   normaal   : 936
##   Median : -5.000      Median : 764  Median :126.0   lang      :1527
##   Mean   :  7.322      Mean   :1026  Mean   :149.6  intercontinentaal: 29
##   3rd Qu.: 14.000      3rd Qu.:1215 3rd Qu.:182.0
##   Max.   :497.000      Max.   :4983  Max.   :650.0
##   NA's    :286         NA's    :287   NA's    :287

##   vertrek_vertraging_na aankomst_vertraging_na vliegtijd_na   vliegtijd_fct
##   Mode :logical        Mode :logical        Mode :logical   kort   :1584
##   FALSE:9747          FALSE:9714          FALSE:9713  normaal:6444
##   TRUE :253           TRUE :286           TRUE :287   lang   :1685
##                           NA's    :287

##   vliegtijd_uren      snelheid
##   Min.   : 0.350      Min.   :120.0
##   1st Qu.: 1.367      1st Qu.:356.9
##   Median : 2.100      Median :402.7
##   Mean   : 2.494      Mean   :392.5
##   3rd Qu.: 3.033      3rd Qu.:436.3
##   Max.   :10.833      Max.   :522.9
##   NA's    :287         NA's    :287
```

- Als we dit vergelijken met de volledige dataset, dan zien we relatief weinig verschillen wat betreft de centrummaten en de robuste spreidingsmaten.
- Merk op dat minima's en maxima's wel sterk kunnen verschillen.
Dit is omdat dit geen robuste maatstaven zijn.

```
summary(df)

##   luchthaven                  maatschappij  vertrek_vertraging
## EWR:119282    United Air Lines Inc.    :57491   Min.   :-43.00
## JFK:105230    ExpressJet Airlines Inc.:54173   1st Qu.: -5.00
```

```

##   LGA:104662    JetBlue Airways          :50940  Median : -2.00
##                               Delta Air Lines Inc.   :46779  Mean   : 12.71
##                               American Airlines Inc. :31327  3rd Qu.: 11.00
##                               Envoy Air           :26397  Max.   :1301.00
##                               (Other)            :62067  NA's   :8214
##   aankomst_vertraging    afstand          vliegtijd                  vluchtype
##   Min.    : -86.000    Min.    : 17     Min.    : 20.0  kort             :245666
##   1st Qu.: -17.000    1st Qu.: 502    1st Qu.: 81.0  normaal        : 31813
##   Median  : -5.000    Median  : 820    Median  :127.0  lang            : 50980
##   Mean    :  6.987    Mean    :1027    Mean    :149.6  intercontinentaal:  715
##   3rd Qu.: 14.000    3rd Qu.:1372    3rd Qu.:184.0
##   Max.    :1272.000   Max.    :4983    Max.    :695.0
##   NA's    :9365      NA's    :9368
##   vertrek_vertraging_na aankomst_vertraging_na vliegtijd_na    vliegtijd_fct
##   Mode :logical      Mode :logical      Mode :logical  kort   : 53220
##   FALSE:320960       FALSE:319809       FALSE:319806  normaal:210758
##   TRUE :8214         TRUE :9365        TRUE :9368    lang   : 55828
##   NA's  :9368
##
##
##
##   vliegtijd_uren      snelheid
##   Min.    : 0.333    Min.    : 76.8
##   1st Qu.: 1.350    1st Qu.:356.3
##   Median  : 2.117    Median  :402.6
##   Mean    : 2.493    Mean    :392.1
##   3rd Qu.: 3.067    3rd Qu.:436.2
##   Max.    :11.583    Max.    :703.4
##   NA's    :9368      NA's    :9368

```

Referenties

1. Encoding
2. De geschiedenis van ASCII
3. Windows code pages
4. Data importeren in R
5. Delimited en fixed-width bestanden
6. XML
7. JSON Tutorial
8. Unique-functie
9. Factors
10. Fct_relevel
11. From continuous to categorical

8

[Tutorial] Data voorbereiden

8.1 Voor je begint

Tijdens deze tutorial zullen we verschillende r-pakketten gebru., indien nodig.

```
library(ggplot2)
library(dplyr)
library(forcats)
library(mice)
```

Het **forcats** pakket bevat een serie functies om op eenvoudige wijze factoren te manipuleren (waarvan *forcats* een anagram is). Het **mice** pakket kan worden gebruikt om het voorkomen van ontbrekende waarden te analyseren (*MICE* staat voor *Multiple Imputation by Chained Equations*, verwijzend naar een techniek om ontbrekende waarden te schatten).

We gaan er ook van uit dat je bekend bent met de inhoud van onze **ggplot2** en **dplyr** tutorial. Je kunt de **survey**¹ dataset laden die bij deze tutorial wordt geleverd als je zelf dingen wilt uitproberen.[^try]

Als je het zelf wilt proberen, volg dan stap voor stap de tutorial. Het incrementele karakter van het cleaning- en transformatieproces staat niet toe om delen van deze tutorial geïsoleerd uit te voeren.

```
survey <- readRDS("survey.RDS")
glimpse(survey)

## Rows: 21,483
## Columns: 9
## $ year      <chr> "2000", "2000", "2000", "2000", "2000", "2000", "2000", "20...
## $ marital   <chr> "Never married", "Divorced", "Widowed", "Never married", "D...
## $ age       <chr> "26", "48", "67", "39", "25", "25", "36", "44", "44", "47",...
## $ race      <chr> "White", "White", "White", "White", "White", "White", "Whit...
## $ rincome   <chr> "$8000 to 9999", "$8000 to 9999", "Not applicable", "Not ap...
```

¹ De **survey** data die gebruikt wordt komt van de General Social Survey, en bevat algemene informatie over sociale aspecten van de Amerikaanse burgers. In het bijzonder bevat het **survey** data.frame een steekproef van categorische attributen.

```
## $ partyid <chr> "Ind,near rep", "Not str republican", "Independent", "Ind,n...
## $ relig   <chr> "Protestant", "Protestant", "Protestant", "Orthodox-christi...
## $ denom   <chr> "Southern baptist", "Baptist-dk which", "No denomination", ...
## $ tvhours <chr> "12", NA, "2", "4", "1", NA, "3", NA, "0", "3", "2", NA, "1...
```

8.2 Inleiding

Deze tutorial over schoonmaken en transformeren is verdeeld in drie verschillende secties.

- Lezen
- Schoonmaken
- Transformeren

Data import. De eerste taak is het inlezen van de gegevens in een bepaald formaat in R. Hoewel we dit onderwerp niet uitgebreid zullen behandelen, geven we wel nuttige verwijzingen naar geschikte R-pakketten om dit te doen.

Data cleaning. We *cleanen* de gegevens door: fouten, duplicaten, enz. te verwijderen.

Data Transformation. Hier richten we ons niet op het verwijderen van fouten en inconsistenties, maar proberen we de gegevens gemakkelijker analyseerbaar te maken: discrete representaties maken van continue variabelen, berekende variabelen toevoegen, of de labels van categorische variabelen hercoderen.

Deze stappen vinden in het algemeen plaats voordat we enige analyse of visualisatie doen die we in de `ggplot2` en `dplyr` tutorials hebben gezien, hoewel vaak meerdere iteraties nodig zijn. Tot nu toe ontvingen we onze gegevens altijd in een vrij schone staat, maar dat is zelden het geval in de werkelijkheid. Nu is het tijd om onze eigen cleaning te doen. Laten we aan de slag gaan!

8.3 Gegevens lezen

Voor dit vak zullen we de verschillende data formaten en hoe ze te lezen niet uitvoerig bespreken.² Meestal zullen we de `readRDS` functie gebruiken, die je waarschijnlijk al eerder hebt gezien. Bijvoorbeeld,

```
survey <- readRDS("survey.RDS")
```

Een .RDS-bestand slaat een **enkel** R-object op een geserialiseerde manier op. (RDS kan worden opgevat als R Data Serialized). We kunnen een .RDS-bestand maken met `saveRDS` en er een lezen met `readRDS`.

Informatiesystemen zullen **nooit** gegevens exporteren als .RDS-bestanden – alle .RDS-bestanden worden gemaakt binnen R. Alle

² Het gedeelte over het lezen van gegevens moet worden gezien als achtergrondmateriaal voor als je het nodig hebt. Er wordt alleen van je verwacht dat je bekend bent met de functies en formaten die in de lessen worden besproken. Als je echter in je toekomstige (studenten)loopbaan een bepaald gegevensbestand moet importeren, kan je dit als uitgangspunt gebruiken. De inhoud van dit deel kan dus gedeeltelijk worden overgeslagen, **uitgezonderd** voor het deel over het converteren van variabelen.

.RDS-bestanden die je hebt gebruikt in de oefeningen en tutorials zijn door ons gemaakt. Dus, in welk type bestanden kunnen gegevens in het wild dan gevonden worden? We geven je een korte rondleiding langs veel voorkomende bestandsformaten.

8.3.1 CSV and TSV

CSV-bestanden zijn waarschijnlijk het meest voorkomende type gegevensbestanden. CSV staat voor Comma Separated Values (komma gescheiden waarden). Deze bestanden kunnen worden gezien als gewone tekstbestanden, waarbij elke regel een waarneming is, d.w.z. een rij, en de kolommen worden gescheiden door komma's (vandaar de naam). De eerste rij kan de namen van de kolommen bevatten, hoewel dit niet noodzakelijk is. TSV is een veel minder gebruikelijke variant, die staat voor Tab Separated Values. Zoals de naam al zegt, worden de waarden in dit bestand niet gescheiden door komma's, maar door tabs.

Voor CSV-bestanden zijn er twee verschillende importfuncties: `read.csv` en `read.csv2`. De eerste is voor normale door komma's gescheiden bestanden, terwijl de tweede voor door puntkomma's gescheiden bestanden is. Verder is het gebruik vergelijkbaar met `readRDS`. De functies voor TSV zijn vergelijkbaar – alleen met een T in plaats van een C.

```
data <- read.csv("path/to/data/file.csv")
```

```
data <- read.csv2("path/to/data/file.csv")
```

Beide functies zijn base-R functies, en hebben veel extra argumenten om het resulterende `data.frame` te verfijnen op basis van eigendigheden in het databestand. Ze worden echter minder gebruikt sinds het `readr` pakket uit de tidyverse snellere functies introduceerde met betere defaults. Deze functies zijn `read_csv` en `read_csv2`, d.w.z. met een underscore in plaats van een punt.

8.3.2 Excel

Hoewel wij niet erg van Excel houden, doen veel mensen dat helaas nog steeds. Het is dan ook waarschijnlijk dat je vroeg of laat een Excel bestand moet inlezen. Het inlezen van Excel bestanden kan met behulp van het speciale `readxl` package. Dit pakket bevat de `read_excel` functie.

```
library(readxl)
data <- read_excel("path/to/excel/file.xlsx")
```

Nogmaals, net als voor csv, zijn er veel extra argumenten in `read_excel`. Je kan bijvoorbeeld het blad in het excel-bestand dat je wilt lezen instellen, je kunt de types van de variabelen configureren, en je kunt zelfs een bereik in het excel-bestand opgeven dat je wilt lezen, bijv. B3:G8.

8.3.3 JSON en XML

JSON - of JavaScript Object Notation - en XML - eXtensible Markup Language - zijn veel complexere gegevensnotaties in vergelijking met CSV. We zullen deze formaten hier niet bespreken, maar in plaats daarvan alleen de pakketten noemen die je kunt gebruiken als je deze types tegenkomt.

- Voor JSON is het meest gebruikte R-pakket `jsonlite`, dat de `fromJSON` functie bevat.
- Voor XML zijn er meerdere opties, maar wij adviseren het `xml2` pakket. Voor `xml` bestanden is er niet een enkele functie, maar zul je typisch veel functies moeten combineren om de data in het juiste formaat te krijgen.

8.3.4 Andere statistische pakketten

Soms moet je gegevens lezen die afkomstig zijn van andere commerciële data-analyse en statistische software die door minder R-vaardige medewerkers wordt gebruikt. (Vaak is dit nodig omdat de betreffende analyse niet door de commerciële pakketten kan worden gedaan en R je moet redden.) In het bijzonder kunnen bestanden afkomstig zijn van SPSS, STATA en SAS. Voor elk van deze bestanden bevat het `haven` pakket een lees-functie.

```
# SPSS
read_spps("file")

# Stata
read_dta("file")

#sas
read_sas("file")
```

8.3.5 Databases

Tenslotte is het ook mogelijk gegevens te analyseren die in een database zijn opgeslagen. De manier waarop zal afhangen van het type database. Een van de handige pakketten is `DBI`, maar je hebt dan wel een specifieke database backend nodig, zoals `RMySQL`, `RSQlite`, `RPostgreSQL`).

Ook handig is `dbplyr`, waarmee veel `dplyr` functies direct op een databank gebruikt kunnen worden, zodat zware berekeningen niet door je pc gedaan hoeven te worden.

8.3.6 Achtergrond materiaal

Meer informatie over het importeren van gegevens vindt u in hoofdstuk 11 van het R for Data Science boek, en op de helppagina's van genoemde pakketten en functies.

Dit is het einde van de optionele sectie over data import

8.3.7 Variabelen omzetten

Vaak is een integraal deel van het lezen van gegevens uit bestanden, ervoor te zorgen dat alle variabelen in onze gegevens correct zijn opgeslagen. Laat factoren factoren zijn, en getallen getallen. Laten we eens kijken naar de dataset.

```
glimpse(survey)
```

```
## Rows: 21,483
## Columns: 9
## $ year      <chr> "2000", "2000", "2000", "2000", "2000", "2000", "2000", "20...
## $ marital   <chr> "Never married", "Divorced", "Widowed", "Never married", "D...
## $ age       <chr> "26", "48", "67", "39", "25", "25", "36", "44", "44", "47",...
## $ race      <chr> "White", "White", "White", "White", "White", "White", "White", ...
## $ rincome   <chr> "$8000 to 9999", "$8000 to 9999", "Not applicable", "Not ap...
## $ partyid   <chr> "Ind,near rep", "Not str republican", "Independent", "Ind,n...
## $ relig     <chr> "Protestant", "Protestant", "Protestant", "Orthodox-christi...
## $ denom     <chr> "Southern baptist", "Baptist-dk which", "No denomination", ...
## $ tvhours   <chr> "12", NA, "2", "4", "1", NA, "3", NA, "0", "3", "2", NA, "1..."
```

Dat lijkt niet erg juist. Door een of andere duivelse kracht zijn alle variabelen als characters opgeslagen, wat niet echt correct is. De variabelen `jaar` en `leeftijd` moeten zeker numeriek zijn, terwijl huwelijk bijvoorbeeld duidelijk een nominale variabele is, en deze als factor moet worden opgeslagen.

Het type van variabelen kan veranderd worden met de volgende functies:³

- `as.numeric` -> voor numerieke variabelen
- `as.integer` -> voor gehele getallen variabelen
- `(as.)factor` -> voor nominale variabelen
- `(as.)ordered` -> voor ordinale variabelen
- `as.character` -> voor karaktervariabelen

³ Merk op dat deze conversies niet standaard zonder gevaar zijn. Een variabele kan bijvoorbeeld alleen numeriek worden gemaakt als alle waarden ervan als numerieke waarden kunnen worden behandeld. Indien het waarden vindt die niet correct kunnen worden omgezet, zoals tekst, zal het in plaats daarvan een ontbrekende waarde invoegen (NA, voor Not Available, zoals we hieronder zullen zien). Het invoegen van NA's zal altijd leiden tot een waarschuwing. Zo'n waarschuwing zal je meestal waarschuwen dat je iets verkeerd deed (een verkeerde variabele geconverteerd?) of dat er fouten in de gegevens zitten.

Om dit op te lossen, gebruiken we een oude bekende uit `dplyr`: `mutate`. We hebben al geleerd dat `mutate` gebruikt kan worden om nieuwe variabelen aan een dataset toe te voegen, maar we kunnen het net zo goed gebruiken om bestaande variabelen te *overschrijven*.

```
survey %>%
  mutate(year = as.integer(year),
        marital = as.factor(marital),
        age = as.integer(age),
        race = as.factor(race),
        rincome = as.factor(rincome),
        partyid = as.factor(partyid),
        relig = as.factor(relig),
        denom = as.factor(denom),
        tvhours = as.numeric(tvhours)) %>%
  glimpse

## #> #> Rows: 21,483
## #> #> Columns: 9
## #> #>   $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, ...
## #> #>   $ marital    <fct> Never married, Divorced, Widowed, Never married, Divorced, ...
## #> #>   $ age        <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, 51, 52, ...
## #> #>   $ race       <fct> White, White, White, White, White, White, White, White, Whi...
## #> #>   $ rincome    <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not applicabl...
## #> #>   $ partyid    <fct> "Ind,near rep", "Not str republican", "Independent", "Ind,n...
## #> #>   $ relig       <fct> Protestant, Protestant, Protestant, Orthodox-christian, Non...
## #> #>   $ denom       <fct> Southern baptist, Baptist-dk which, No denomination, Not ap...
## #> #>   $ tvhours     <dbl> 12, NA, 2, 4, 1, NA, 3, 0, 3, 2, NA, 1, 7, NA, 3...
```

Dat ziet er al beter uit! Merk echter op dat we het resultaat van onze inspanningen nog niet hebben opgeslagen. Eigenlijk willen we van de gelegenheid gebruik maken om alle variabelen een makkelijke en begrijpelijke naam te geven. Hiervoor kunnen we de `rename` functie gebruiken. `rename` is een `dplyr` functie met een heel duidelijke taak: variabelen hernoemen. Je kunt hem gebruiken door hem een lijst met nieuwe namen te geven, gekoppeld aan oude namen: `new_name = old_name`.

```
survey %>%
  mutate(year = as.integer(year),
        marital = as.factor(marital),
        age = as.integer(age),
        race = as.factor(race),
        rincome = as.factor(rincome),
        partyid = as.factor(partyid),
        relig = as.factor(relig),
```

```

denom = as.factor(denom),
tvhours = as.numeric(tvhours)) %>%
rename(reported_income = rincome,
party = partyid,
religion = relig,
denomination = denom,
tv_hours = tvhours) %>%
glimpse

## Rows: 21,483
## Columns: 9
## $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 200...
## $ marital    <fct> Never married, Divorced, Widowed, Never married, Di...
## $ age        <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, ...
## $ race       <fct> White, White, White, White, White, White, White, Wh...
## $ reported_income <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not a...
## $ party      <fct> "Ind,near rep", "Not str republican", "Independent"...
## $ religion   <fct> Protestant, Protestant, Protestant, Orthodox-christ...
## $ denomination <fct> Southern baptist, Baptist-dk which, No denomination...
## $ tv_hours    <dbl> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, ...

```

Er is zeker geen juist antwoord voor de naamgeving van variabelen. Zorg er gewoon voor dat hun namen begrijpelijk, gemakkelijk te gebruiken en enigszins uniform getypt zijn.

Merk verder op dat wat we net deden niet de enige mogelijke manier is. We zouden bijvoorbeeld ook direct de nieuwe variabelennamen kunnen maken met mutate, hoewel we daarna de oude namen wel nog zullen moeten verwijderen.

```

survey %>%
  mutate(year = as.integer(year),
marital = as.factor(marital),
age = as.integer(age),
race = as.factor(race),
reported_income = as.factor(rincome),
party = as.factor(partyid),
religion = as.factor(relig),
denomination = as.factor(denom),
tv_hours = as.numeric(tvhours)) %>%
select(-rincome:-tvhours) %>%
glimpse

## Rows: 21,483
## Columns: 9
## $ year      <int> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 200...
## $ marital    <fct> Never married, Divorced, Widowed, Never married, Di...

```

```
## $ age           <int> 26, 48, 67, 39, 25, 25, 36, 44, 44, 47, 53, 52, 52, ...
## $ race          <fct> White, White, White, White, White, White, Wh...
## $ reported_income <fct> $8000 to 9999, $8000 to 9999, Not applicable, Not a...
## $ party          <fct> "Ind,near rep", "Not str republican", "Independent"...
## $ religion        <fct> Protestant, Protestant, Protestant, Orthodox-christ...
## $ denomination    <fct> Southern baptist, Baptist-dk which, No denomination...
## $ tv_hours        <dbl> 12, NA, 2, 4, 1, NA, 3, NA, 0, 3, 2, NA, 1, NA, 1, ...
```

Het resultaat is hetzelfde, maar de code is iets korter. Als je dit echt onder de knie wilt krijgen, is het misschien interessant om te weten dat er veel varianten op mutate zijn die je leven misschien nog eenvoudiger (of verwarrender) maken.

- transmute: dit zal **alleen** de *nieuwe* variabelen in je lijst houden
- mutate_if: dit werkt op dezelfde manier als select_if, b.v. een functie toepassen op een bepaald type kolommen
- mutate_at: dit zal een functie toepassen op een bepaalde set van kolommen die je opgeeft.

Maak je geen zorgen. Je zult een heel eind komen als je select, mutate en rename kunt gebruiken. Maar wees niet bang om jezelf uit te dagen en de meer geavanceerde dingen uit te proberen.

Nu, laten we verder gaan. Voordat we dat doen, kopiëren we het laatste stukje code, waarbij we dit keer het resultaat weer opslaan als **survey**, waarmee we de oude versie overschrijven. U kunt dit op twee manieren doen: of je zet **survey** `<-` voor het stukje code, of je zet `->` **survey** na het stukje code. Nogmaals, er is geen foute of goede manier. Persoonlijk geef ik de voorkeur aan de laatste optie, omdat het mooi aansluit bij ons verhaal dat we hebben gemaakt met het `%>%` symbol: we nemen een dataset, we voeren een aantal stappen uit, en dan slaan we het op.

```
survey %>%
  mutate(year = as.integer(year),
         marital = as.factor(marital),
         age = as.integer(age),
         race = as.factor(race),
         reported_income = as.factor(rincome),
         party = as.factor(partyid),
         religion = as.factor(relig),
         denomination = as.factor(denom),
         tv_hours = as.numeric(tvhours)) %>%
  select(-rincome:-tvhours) -> survey
```

Dit is een goede plaats om aandacht te besteden aan work-flow aspecten. Vroeger, tijdens de analyse van gegevens, hingen verschillende stukken code zelden van elkaar af. Als we bijvoorbeeld grafiek

A maakten en daarna tabel B, konden beide onafhankelijk van elkaar worden gemaakt. We sloegen de resultaten die we maakten nooit op om later te gebruiken (afgezien van een steekproef van gegevens die we soms namen). Nu we echter de gegevens gaan opruimen en transformeren, zullen we het data.frame altijd bijwerken, meestal onder dezelfde naam. We willen immers niet eindigen met een lijst van survey, survey2, survey3, survey4, zonder hun verschillen te onthouden. Dus, bij elke stap, updaten we de vorige versie van de survey dataset.

Er is echter een risico. Als wij een fout maken, kunnen onze gegevens worden beschadigd. Als wij bijvoorbeeld race per ongeluk hebben geconverteerd naar numeriek, zal de functie as.numeric hier niet in slagen en in plaats daarvan een kolom vol met NA's creëren. We kunnen dan snel onze fout in de code rechtzetten, maar dit zal de originele race variabele niet terugbrengen – die was weg op het moment dat we ze per vergissing converteerden.

Om onze fouten recht te zetten, zullen we de gegevens opnieuw moeten laden, en ook alle transformaties die we al eerder hebben toegepast. Alleen de code corrigeren zal niet meer voldoende zijn, **we moeten onze data corrigeren**. Wanneer je in R Markdown werkt, kun je dit het beste doen met de centrale knop in een R-chunk, omdat je dan alle vorige R-chunks opnieuw uitvoert, waardoor onze gegevens weer in de staat komen waarin ze eerder waren.

Deze afhankelijkheden in onze workflow betekenen ook dat oefeningen meer van elkaar afhankelijk zullen zijn, en we moeten er altijd voor zorgen dat we ons bijgewerkte data.frame opslaan. Het niet bijwerken van de gegevens (of het niet uitvoeren van de code) zal later een frequente bron van fouten zijn. Let op. (U bent gewaarschuwd)

8.4 Cleaning Data

Nu we de gegevens hebben geïmporteerd en er zeker van zijn dat alle variabelen ten minste het juiste type hebben, is het tijd om de gegevens op te schonen. In het bijzonder zullen we de volgende onderwerpen behandelen

- Dubbele observaties
- Cleanen van categorische variabelen
- Cleanen van continue variabelen
- Controleren van inconsistencies in de gegevens

Verder zullen we ook enige tijd besteden aan het bespreken van ontbrekende waarden. Dat zijn een heleboel concepten om te behandelen, dus laten we beginnen!

8.4.1 Dubbele observaties

Soms kan het gebeuren dat sommige rijen per ongeluk meerdere keren in de dataset zijn opgenomen. Er is een gemakkelijke manier om deze te vinden, en te verwijderen. De `duplicated` functie (een base-R functie), geeft een logische vector terug die dubbele rijen in een dataset aangeeft. De vector heeft dezelfde lengte als het aantal rijen in de dataset, en is `TRUE` voor rijen die niet uniek zijn, en `FALSE` anders.

```
survey %>%
  duplicated %>%
  summary

##      Mode    FALSE     TRUE
## logical  21220     263
```

Het lijkt erop dat er 263 in onze gegevens zitten die niet uniek zijn. We kunnen deze bekijken door de uitvoer van `duplicated` te gebruiken als invoer van filter.

```
survey %>%
  filter(duplicated(.))

## # A tibble: 263 x 9
##       year marital   age race reported_income party religion denomination
##       <int> <fct>   <int> <fct> <fct>        <fct> <fct>    <fct>
## 1  2000 Married     48 White $25000 or more Ind,~ Catholic Not applica~
## 2  2000 Never ~     39 Other $25000 or more Not ~ Catholic Not applica~
## 3  2000 Married     36 Other $25000 or more Not ~ Catholic Not applica~
## 4  2000 Never ~     30 White $25000 or more Not ~ Catholic Not applica~
## 5  2000 Never ~     19 White $1000 to 2999 Not ~ Catholic Not applica~
## 6  2000 Married     47 White $25000 or more Not ~ Catholic Not applica~
## 7  2000 Married     29 White $25000 or more Inde~ Catholic Not applica~
## 8  2000 Married     39 White $10000 - 14999 Not ~ Catholic Not applica~
## 9  2000 Widowed    80 White Not applicable Stro~ Protest~ Southern ba~
## 10 2002 Married     43 White Not applicable Inde~ Catholic Not applica~
## # ... with 253 more rows, and 1 more variable: tv_hours <dbl>
```

Zie je de `.` in de `duplicated` functie? De punt heeft een speciale betekenis als hij samen met het piping-symbool wordt gebruikt. Intern zal het vervangen worden door de invoer die door het piping symbool komt. Als zodanig is `survey %>% filter(duplicated(.))` gelijk aan `filter(survey, duplicated(survey))`. Dit is erg handig als je meerdere keren naar de piping-invoer moet verwijzen, niet alleen als eerste argument van de functie.

Op dit moment hebben we de dubbele rijen geselecteerd en kunnen we ze bekijken. Als we alleen de unieke rijen willen behouden, kunnen we een `!`-symbool toevoegen om de selectie te negeren.

```

survey %>%
  filter(!duplicated(.))

## # A tibble: 21,220 x 9
##   year marital age race reported_income party religion denomination
##   <int> <fct>  <int> <fct> <fct>      <fct> <fct>    <fct>
## 1 2000 Never ~    26 White $8000 to 9999 Ind,~ Protest~ Southern ba~
## 2 2000 Divorc~ 48 White $8000 to 9999 Not ~ Protest~ Baptist-dk ~
## 3 2000 Widowed 67 White Not applicable Inde~ Protest~ No denomina~
## 4 2000 Never ~    39 White Not applicable Ind,~ Orthodo~ Not applica~
## 5 2000 Divorc~ 25 White Not applicable Not ~ None     Not applica~
## 6 2000 Married   25 White $20000 - 24999 Stro~ Protest~ Southern ba~
## 7 2000 Never ~    36 White $25000 or more Not ~ Christi~ Not applica~
## 8 2000 Divorc~ 44 White $7000 to 7999 Ind,~ Protest~ Lutheran-mo~
## 9 2000 Married   44 White $25000 or more Not ~ Protest~ Other
## 10 2000 Married  47 White $25000 or more Stro~ Protest~ Southern ba~
## # ... with 21,210 more rows, and 1 more variable: tv_hours <dbl>

```

Dit is echter een beetje omslachtig. Daarom bevat dplyr een heel handige short cut: de distinct functie.

```

survey %>%
  distinct()

## # A tibble: 21,220 x 9
##   year marital age race reported_income party religion denomination
##   <int> <fct>  <int> <fct> <fct>      <fct> <fct>    <fct>
## 1 2000 Never ~    26 White $8000 to 9999 Ind,~ Protest~ Southern ba~
## 2 2000 Divorc~ 48 White $8000 to 9999 Not ~ Protest~ Baptist-dk ~
## 3 2000 Widowed 67 White Not applicable Inde~ Protest~ No denomina~
## 4 2000 Never ~    39 White Not applicable Ind,~ Orthodo~ Not applica~
## 5 2000 Divorc~ 25 White Not applicable Not ~ None     Not applica~
## 6 2000 Married   25 White $20000 - 24999 Stro~ Protest~ Southern ba~
## 7 2000 Never ~    36 White $25000 or more Not ~ Christi~ Not applica~
## 8 2000 Divorc~ 44 White $7000 to 7999 Ind,~ Protest~ Lutheran-mo~
## 9 2000 Married   44 White $25000 or more Not ~ Protest~ Other
## 10 2000 Married  47 White $25000 or more Stro~ Protest~ Southern ba~
## # ... with 21,210 more rows, and 1 more variable: tv_hours <dbl>

```

Of we dubbele rijen willen verwijderen of niet hangt echt af van de gegevens en context. In ons geval is het helemaal niet verwonderlijk dat sommige van deze rijen dezelfde zijn. Het is gewoon zo dat sommige mensen erg op elkaar lijken: dezelfde leeftijd, inkomen, religie enz.

Maar in andere gevallen zouden dergelijke dubbele rijen onmogelijk zijn. Bijvoorbeeld, als er variabelen zijn die elke rij per definitie uniek

zouden maken, zoals een nationaal identificatienummer. In dergelijke gevallen moeten dubbele rijen duidelijk verder bekeken worden en kan het verwijderen ervan de juiste oplossing zijn.

8.4.2 Cleaning categorische variabelen

Voor het opruimen van categorische variabelen bespreken wij de volgende wijzigingen

- Waarden hercoderen
- Herordenen van waarden

Hercoderen Categorische Variabelen

Soms hebben categorische variabelen, d.w.z. factoren, vreemde of zelfs verkeerde labels. In dat geval zouden we deze waarden willen *hercoderen*. Het vinden van verkeerde labels is niet altijd gemakkelijk, en vaak komen deze fouten later tijdens de analyse aan het licht, in welk geval je een stap terug moet doen en ze achteraf moet corrigeren. Niettemin kan het bekijken van frequentietabellen in alfabetische volgorde, of gerangschikt van minst naar meest frequent, wijzen op enkele fouten.⁴ Laten we de partijvariabele als voorbeeld nemen.

```
survey %>%
  count(party)

## # A tibble: 10 x 2
##   party             n
##   <fct>           <int>
## 1 Don't know       1
## 2 Ind,near dem    2499
## 3 Ind,near rep    1791
## 4 Independent      4119
## 5 No answer        154
## 6 Not str democrat 3690
## 7 Not str republican 3032
## 8 Other party      393
## 9 Strong democrat  3490
## 10 Strong republican 2314
```

Hoewel de waarden voor partij niet echt fout zijn, zijn ze niet allemaal uniform: Str en Strong, Ind en Independent... Laten we ze veranderen. We kunnen de factor labels hercoderen met de `fct_recode` functie van `forcats`. Als argumenten moeten we opgeven welke variabele we willen hercoderen, en welke levels we willen veranderen.

⁴ De `count` functie die hier gebruikt wordt is een `dplyr` afkorting voor `group_by(party) %>% summarise(n = n())`. Voel je vrij om het te gebruiken om je een hoop typewerk te besparen. Het heeft ook een `sort` argument om te sorteren op aflopende frequenties. Als zodanig is `count(party, sort = T)` gelijk aan `group_by(party) %>% summarise(n = n()) %>% arrange(-n)`. Let er echter op dat `count` de hele groepering van een `data.frame` achteraf verwijdert, in tegenstelling tot `summarize`.

```
data %>%
  mutate(<factor_name> = fct_recode(<factor_name>,
                                     "<new_level1>" = "<old_level1>",
                                     "<new_level2>" = "<old_level2>")
```

We kunnen zoveel oude velden hercoderen in nieuwe velden als we willen. Bovendien kun je meerdere oude velden vervangen door hetzelfde nieuwe veld. Elk niveau dat niet genoemd wordt, zal ongeldig blijven. Laten we wat uniformiteit brengen in de politieke voorkeuren.

```
survey %>%
  mutate(party = fct_recode(party,
                            "Republican, strong" = "Strong republican",
                            "Republican, weak" = "Not str republican",
                            "Independent, near rep" = "Ind,near rep",
                            "Independent, near dem" = "Ind,near dem",
                            "Democrat, weak" = "Not str democrat",
                            "Democrat, strong" = "Strong democrat"
  )) -> survey
```

Vergeet niet de dataset op te slaan!

Ordenen Categorische Variabelen

Een andere mogelijkheid, vooral voor ordinale factoren, is dat de waarden niet echt fout zijn, maar dat ze in de verkeerde volgorde staan. Normaal is dit iets waar we opletten bij het lezen van de data, maar soms wel eens vergeten. Kijk bijvoorbeeld eens naar het gerapporteerde inkomen.

```
survey %>%
  count(reported_income)

## # A tibble: 16 x 2
##   reported_income     n
##   <fct>           <int>
## 1 $1000 to 2999     395
## 2 $10000 - 14999    1168
## 3 $15000 - 19999    1048
## 4 $20000 - 24999    1283
## 5 $25000 or more    7363
## 6 $3000 to 3999     276
## 7 $4000 to 4999     226
## 8 $5000 to 5999     227
## 9 $6000 to 6999     215
```

```
## 10 $7000 to 7999      188
## 11 $8000 to 9999      340
## 12 Don't know         267
## 13 Lt $1000           286
## 14 No answer          183
## 15 Not applicable     7043
## 16 Refused            975
```

De waarde “Lt \$1000” - wat betekent Limited, of minder dan \$1000 - zou eerst moeten worden getoond, maar in plaats daarvan staat het op de verkeerde plaats. Hier hebben we een andereforcats functie nodig, namelijk `fct_relevel`. Deze functie kan op twee verschillende manieren gebruikt worden om een level op een andere plaats te zetten.

Optie 1: Verplaats één (of meer) niveau(s) naar voren

```
data %>%
  mutate(factor_name = fct_relevel(factor_name,
                                    "level1_to_move", "level2_to_move", "..."))
```

Optie 2: Een (of meer) niveau(s) invoegen na een aantal N van niveaus

```
data %>%
  mutate(factor_name = fct_relevel(factor_name,
                                    "level1_to_move", "level2_to_move", "...", after = N))
```

Dus, laten we het LT \$1000 niveau naar de eerste plaats verplaatsen.

```
survey %>%
  mutate(reported_income = fct_relevel(reported_income,
                                         "Lt $1000")) -> survey
```

We kunnen de resultaten controleren door telling te gebruiken op het bijgewerkte `survey` data.frame. (Want je hebt je wijziging opgeslaan, toch?)

```
survey %>%
  count(reported_income)

## # A tibble: 16 x 2
##       reported_income     n
##       <fct>             <int>
## 1 Lt $1000           286
## 2 $1000 to 2999       395
## 3 $10000 - 14999     1168
## 4 $15000 - 19999     1048
```

```

## 5 $20000 - 24999    1283
## 6 $25000 or more   7363
## 7 $3000 to 3999    276
## 8 $4000 to 4999    226
## 9 $5000 to 5999    227
## 10 $6000 to 6999   215
## 11 $7000 to 7999   188
## 12 $8000 to 9999   340
## 13 Don't know       267
## 14 No answer        183
## 15 Not applicable   7043
## 16 Refused          975

```

Het wijzigen van de volgorde van de niveaus van een categorische variabele is nuttig voor zowel nominale als ordinale gegevens. Voor ordinale gegevens is het logisch dat we willen dat de volgorde van de niveaus de juiste is. Maar ook voor nominale gegevens kan het nodig zijn de volgorde te veranderen. Er zijn bijvoorbeeld vaak “catch-all”-waarden zoals “Andere” of “Diverse”. Het is een goede gewoonte om deze waarden anders te behandelen dan de gewone waarden in een nominale variabele, door ze als laatste te zetten. Ze komen dan aan de ene kant van een grafiek of tabel te staan, en niet tussen de andere waarden. Laten we eens kijken naar partij.

```

survey %>%
  count(party)

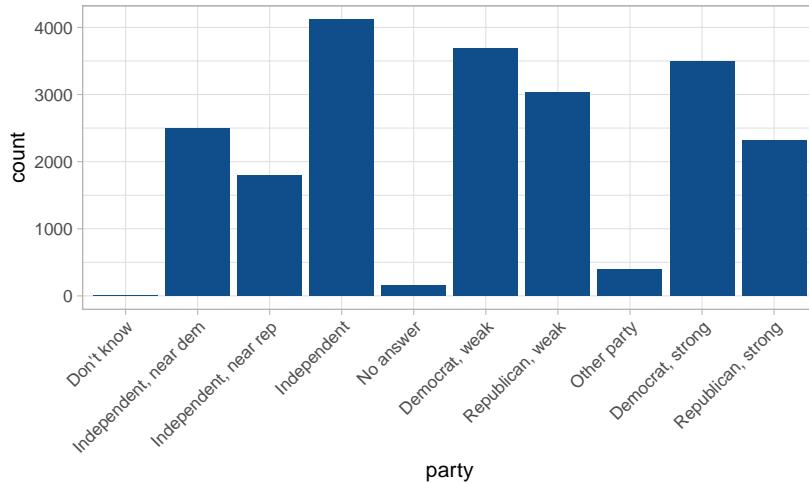
## # A tibble: 10 x 2
##   party                 n
##   <fct>                <int>
## 1 Don't know            1
## 2 Independent, near dem 2499
## 3 Independent, near rep 1791
## 4 Independent            4119
## 5 No answer              154
## 6 Democrat, weak        3690
## 7 Republican, weak      3032
## 8 Other party            393
## 9 Democrat, strong       3490
## 10 Republican, strong    2314

```

Beslissen of een factor ordinaal is of niet, is niet altijd zo eenvoudig. Als we naar het gerapporteerde inkomen kijken, is het duidelijk dat er een rangorde is. Maar we hebben **partij** niet gedefinieerd als een geordende factor. Er is geen “beste” of “superieure” politieke partij, dus het expliciet programmeren van deze variabele als een ordinale

factor zou een brug te ver zijn – we zouden dan moeten beslissen welke kant van het politieke spectrum de “laagste” en welke de “hoogste” is. Dit is echter enigszins ongewenst als we grafieken maken.

```
survey %>%
  ggplot(aes(party)) +
  geom_bar(fill = "dodgerblue4") +
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Wanneer we geen ordinale factor hebben, zal de volgorde van de labels vaak alfabetisch zijn. In dit geval, omdat we de labels eerder hebben gehercodeerd, staan ze niet eens meer in alfabetische volgorde⁵. De resulterende grafiek is moeilijk te lezen, omdat de x-as door elkaar is geschud. Een natuurlijke reflex zou zijn om het staafdiagram te ordenen volgens frequentie, maar dat zou de leesbaarheid in dit speciale geval niet echt verbeteren. In plaats daarvan kunnen we een logischer volgorde toepassen, zonder dat we de partij als een ordinale variabele hoeven te beschouwen. Een dergelijke logische volgorde is gemakkelijk beschikbaar voor de huidige variabele, aangezien we vaak spreken van linkse en rechtse politici. We kunnen de alternatieve antwoorden (Weet niet, Geen Antwoord, Andere Partij) laten staan, hetzij aan het begin of aan het einde van de volgorde. Door ze niet te vermelden in de code hieronder, zal het laatste gebeuren.

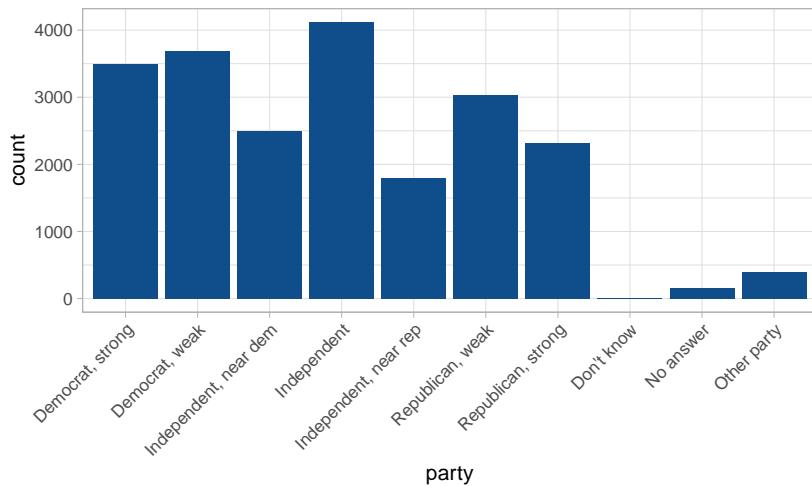
```
survey %>%
  mutate(party = fct_relevel(party,
    "Democrat, strong",
    "Democrat, weak",
    "Independent, near dem",
    "Independent",
    "Independent, near rep",
```

⁵ Toen we voor het eerst as.factor(party) deden, werden de niveaus in alfabetische volgorde geplaatst. Daarna hebben we echter sommige niveaus gehercodeerd, maar dit heeft de volgorde niet bijgewerkt. Zo werd “Strong Republican” nu “Republican, Strong” en “Strong Democrat” werd “Democrat, Strong”, maar beide bleven de laatste niveaus omdat ze oorspronkelijk begonnen met S. Het is niet te verwachten dat je bekend bent met alle neveneffecten van sommige transformaties, maar dit toont je de complexiteit wanneer onze acties beginnen af te hangen van eerdere acties, en hoe we echt werken in een iteratieve context.

```
"Republican, weak",
"Republican, strong")) -> survey
```

Onze grafiek ziet er nu als volgt uit. Beter, is het niet? [^beter]

```
survey %>%
  ggplot(aes(party)) +
  geom_bar(fill = "dodgerblue4") +
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



In conclusie: gebruiken we

- `fct_recode` voor het hercoderen van waarden van een categorische variabele, en
- `fct_relevel` voor het handmatig herordenen van waarden van een categorische variabele.

Later zullen we meer specifieke functies zien voor het hercoderen en herordenen van categorische variabelen bij het transformeren van gegevens (niet omdat er fouten zijn, maar omdat ze onze analyse daarmee kunnen verbeteren). Zorg ervoor dat je het overzicht niet verliest! Eerst kijken we naar het opruimen van continue gegevens.

8.4.3 Cleaning continue variabelen

Voor continue gegevens is het bereik van mogelijke waarden oneindig, en is het dus moeilijker *foute* waarden te vinden. Zonder informatie over de context van de gegevens is het vinden van foute continue waarden uiterst moeilijk.

Fouten

In het `survey` data.frame, zijn er drie continue variabelen: jaar, leeftijd en (dagelijkse) `tv_hours`. Voor elk van deze variabelen hebben we een idee over het mogelijke bereik van waarden. De leeftijd zal waarschijnlijk ergens tussen 20 en 100 liggen (wetende dat de dataset informatie over volwassenen bevat). Het aantal tv-uren moet tussen 0 en 24 liggen, aangezien er 24 uur in een dag zitten. Ook voor het jaar weten we min of meer wat we kunnen verwachten. Laten we eens kijken naar elk.⁶

```
summary(survey$year)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	2000	2002	2006	2007	2010	2014

Voor het jaar, lijkt alles in orde. We hebben het al eerder omgezet in een gehele variabele, dus we hoeven niet te controleren op foutieve decimale jaren. Ook het minimum en maximum lijken in orde. Als het jaar fouten bevat, zien we dat waarschijnlijk in deze extremen: 2102 in plaats van 2012, 1099 in plaats van 1999, of bv. 9999 wat aangeeft dat het eigenlijk ontbreekt.

Laten we dan eens kijken naar leeftijd.

```
summary(survey$age)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	18.00	33.00	46.00	47.18	59.00	89.00	76

Op het eerste gezicht lijken er ook geen problemen te zijn met de leeftijd. Er zijn 76 ontbrekende waarden, maar de aanwezige waarden liggen tussen 18 en 89 jaar, wat ook weer een logisch bereik is. Ook leeftijd werd eerder zonder problemen omgezet in een gehele variabele, zodat alle waarden gehele getallen zijn.

Als we naar de tv-uren kijken, zien we iets eigenaardigs.

```
summary(survey$tv_hours)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	0.000	1.000	2.000	3.004	4.000	84.000	10146

We zien dat er 10146 ontbrekende waarden zijn, wat hoog is maar niet noodzakelijk verkeerd (tenzij we per ongeluk enkele waarden hebben verwijderd, wat we niet hebben gedaan). Er zijn geen negatieve waarden, aangezien het minimumaantal uren dat iemand tv heeft gekeken nul is. Aan de andere kant zien we echter dat het aantal tv-uren oploopt tot 84 uur per dag - dit is duidelijk verkeerd. We hebben allemaal maar 24 uur in een dag.

⁶ Wanneer we categorische gegevens inspecteren, kunnen we een snelle telling doen. Hoewel dit zou kunnen werken voor sommige continue variabelen, zoals jaar, is het niet geschikt voor de meeste continue variabelen, omdat ze vaak uniek zijn voor elke observatie. In plaats daarvan kijken we naar de samenvatting van de variabelen.

We kunnen verder kijken naar de records waarvoor de tv-uren meer dan 24 zijn.

```
survey %>%
  filter(tv_hours > 24)

## # A tibble: 5 x 9
##   year marital age race reported_income party religion denomination tv_hours
##   <int> <fct>  <int> <fct>      <fct> <fct>    <fct>      <dbl>
## 1 2000 Married    84 White Not applicable Inde~ Protest~ No denomina~     28
## 2 2000 Widowed    68 White Not applicable Inde~ Protest~ Other luthe~     84
## 3 2010 Married    69 White Not applicable Demo~ Protest~ Baptist-dk ~    35
## 4 2010 Divorc~    57 White Not applicable Demo~ Catholic Not applica~    56
## 5 2014 Separa~    30 White Not applicable Inde~ None      Not applica~    28
```

Er blijken 5 waarnemingen te zijn waarvoor het aantal tv-uren duidelijk fout is, en die moeten we corrigeren. We hebben echter geen idee hoe we dat in ons geval moeten doen. Het enige wat we kunnen doen, is deze waarden schrappen, en ze missing maken.⁷ Let op, we verwijderen niet de volledige waarnemingen, alleen de tvhours variabele voor deze waarnemingen. De andere variabelen kunnen nog steeds worden gebruikt voor deze 5 rijen.

In sommige gevallen kan je proberen te achterhalen wat de fout is. Als het gaat over de lengte van personen zijn er mogelijke enkele waarden in meter uitgedrukt, en de rest in centimeter? Misschien is er sprake van verschillende eenheden (meter vs feet, km vs mile). Anders kan je in werkelijkheid ook proberen teruggaan naar de bron van de data, of de domein expert, om te achterhalen wat er mis is. Als al deze acties geen opheldering bieden, zit er niets anders op dan de waarden te verwijderen.

We kunnen dit doen door gebruik te maken van de `ifelse` functie. Deze functie is een zeer algemene functie die een waarde teruggeeft die afhankelijk is van een logische test.⁸ De functie kan als volgt worden gebruikt.

```
if_else( <condition>, <value_a>, <value_b> )
```

Stel, we hebben een vector `score` die leerlingenscores bevat. We kunnen de `ifelse` functie gebruiken om een vector `grade` te maken met de waarden FAILED en PASSED.

```
grade <- ifelse(score >= 10, "PASSED", "FAILED")
```

In onze situaties is de vector die we willen aanpassen een kolom in een dataframe, dus we gebruiken `mutate` om dit te doen. Laten we de functie gebruiken om de variabele tv-uren bij te werken.

⁷ Dergelijke fouten kunnen vaak worden vermeden door een goede gegevensverzameling, d.w.z. als je online een enquête of een gegevensformulier maakt, zorg er dan voor dat alle velden redelijk gecontroleerd zijn: leeftijd kan niet negatief zijn, een postcode bestaat uit 4 cijfers, en men heeft niet meer dan 24 uur in een dag. Hoewel er later nog dingen mis kunnen gaan, kunnen de gegevens in ieder geval niet verkeerd worden ingevoerd door de respondenten. In het echte leven gebeurt veel gegevensverzameling helaas zonder veel nadenken.

⁸ Misschien ben je bekend met de IF functie in Excel? Het gebruik ervan is precies hetzelfde.

```
survey %>%
  mutate(tv_hours = ifelse(tv_hours > 24, NA, tv_hours)) -> survey
```

Dus, wat gebeurt er? De variabele `tv_hours` wordt bijgewerkt met `mutate`. Als deze groter is dan 24, is de nieuwe waarde `NA`, d.w.z. Not Available, missing. Anders is de nieuwe waarde gewoon de oude waarde van `tv_hours`, d.w.z. ongewijzigd. Nadat de kolom is bijgewerkt, wordt het nieuwe data.frame weer opgeslagen als `survey`.

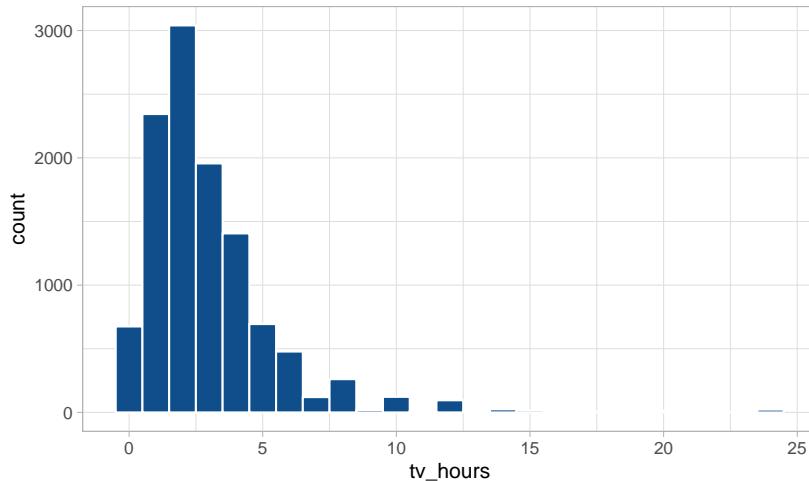
Het controleren op fouten, zowel categorische als continue, kan een *straat zonder einde* zijn. Meestal doe je je uiterste best door voor elke variabele de hierboven besproken testen toe te passen tijdens je beschrijvende analyse. Als dit veel werk lijkt, bedenk dan dat een doorsnee dataproject voor 70% bestaat uit het opschonen en transformeren van gegevens, en voor slechts 30% uit de eigenlijke analyse en interpretatie.

En zelfs dan kan het gebeuren dat je, ondanks al je tijd en inspanningen, in de analysefase fouten in de gegevens ontdekt. Dat is niet verwonderlijk, want op dat moment ga je de gegevens echt in detail bekijken. Wanneer dit gebeurt, ga je terug naar uw cleaning- en transformatiescripts en pas je ze aan waar nodig. Het is dus belangrijk dat je zowel de oorspronkelijke gegevens als alle wijzigingen die je hebt aangebracht, bewaart. Nog belangrijker is het om ervoor te zorgen dat al je analyses worden opgeslagen als een script van RMarkdown, zodat ze snel opnieuw kunnen worden uitgevoerd na het corrigeren van de fout. Dit heet *reproducible research* en zal je veel tijd en fouten besparen. Kijk eens naar deze video voor een illustratie van reproducerebare work-flows.

Outliers

Het komt ook voor dat sommige continue waarden niet noodzakelijk fout zijn, maar *uitzonderlijk* hoog of laag. Deze uitzonderingen noemen we geen fouten, maar outliers – een gegevenspunt dat ver af ligt van andere waarnemingen. Neem bijvoorbeeld tv-uren. Eerder verwijderden we de waarden groter dan 24 uur, omdat daar logischerwijze fouten in zitten. Laten we eens kijken naar de resterende waarden.

```
survey %>%
  ggplot(aes(tv_hours)) +
  geom_histogram(binwidth = 1,
                 color = "white", fill = "dodgerblue4") +
  theme_light()
```



Het lijkt erop dat, hoewel de meeste mensen tussen 0 en 5 uur per dag televisie kijken, er enkele uitzonderlijk hoge waarden zijn. Het is niet duidelijk of het hier gaat om vergissingen zoals we die eerder hebben verwijderd, of gewoon om abnormale tv-verslaafden. Laten we eens kijken naar de 25 hoogste waarden, en ze vergelijken met enkele verwante attributen, zoals leeftijd, inkomen en burgerlijke staat (ze vergelijken met godsdienst of partij zou kunnen worden beschouwd als politiek incorrect, dus laten we uit die gevarenzone weg blijven). ⁹

```
survey %>%
  arrange(-tv_hours) %>%
  slice(1:25) %>%
  select(tv_hours, marital, age, reported_income) %>%
  pander()
```

⁹ Merk op dat we pander gebruiken om de opmaak van onze tabellen te verbeteren, en dat je daar verder geen aandacht aan moet besteden in deze tutorial.

tv_hours	marital	age	reported_income
24	Never married	30	Not applicable
24	Separated	45	Not applicable
24	Never married	33	\$6000 to 6999
24	Divorced	53	Not applicable
24	Divorced	50	No answer
24	Never married	44	Not applicable
24	Never married	21	Don't know
24	Widowed	71	Not applicable
24	Widowed	62	Not applicable
24	Widowed	52	Refused

tv_hours	marital	age	reported_income
24	Never married	56	Not applicable
24	Divorced	51	Not applicable
24	Divorced	75	Not applicable
24	Separated	49	\$8000 to 9999
24	Divorced	65	Not applicable
24	Never married	27	Not applicable
24	Married	71	Not applicable
24	Never married	27	\$8000 to 9999
24	Separated	63	Not applicable
24	Divorced	31	\$5000 to 5999
24	Separated	37	Not applicable
24	Married	46	Not applicable
23	Never married	32	Not applicable
22	Divorced	69	Not applicable
22	Married	63	Not applicable

Het lijkt erop dat veel van de respondenten elke dag het maximumaal van 24 uur tv kijken, wat verrassend is. We kunnen de andere variabelen gebruiken om een beter beeld te krijgen van deze waarnemingen. We zien dat velen van hen alleenstaand zijn (nooit getrouwde, gescheiden of gescheiden). De leeftijd lijkt relatief hoog te zijn, maar niet erg opmerkelijk (de gemiddelde leeftijd voor de dataset was ongeveer 47 jaar). Ten slotte hebben velen hun inkomen niet gerapporteerd.

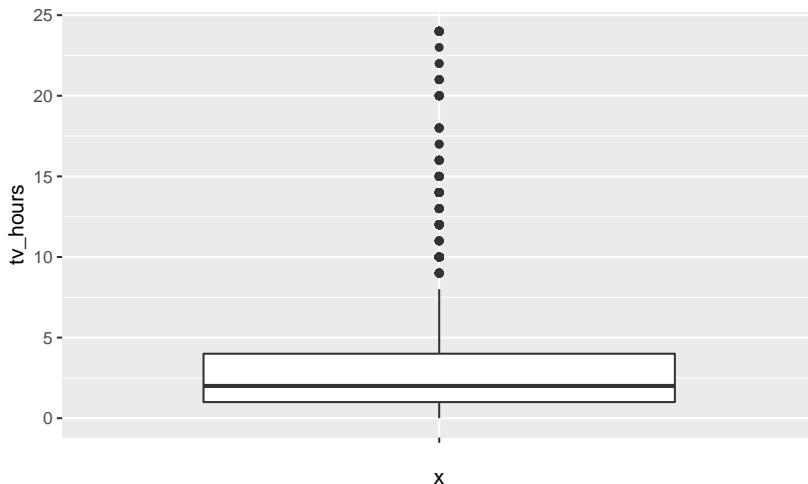
Deze informatie kan op verschillende manieren geïnterpreteerd worden:

- Dit zijn alleenstaande, luie mensen met een inkomen die de hele dag tv kijken
- Dit zijn mensen die niet erg eerlijk waren bij het invullen van hun informatie (ook geen inkomen opgeven).

Beslissen of iets fout of uitzonderlijk is, is niet triviaal en vereist bepaalde domeinkennis en aannames. In dit geval is de logische aanname dat deze cijfers onjuist zijn (zelfs luie, tv-verslaafde mensen moeten af en toe slapen). De moeilijkere vraag is op welk punt iets onjuist wordt. 20 uur? 16? Voor continue variabelen kan het helpen om naar een boxplot te kijken en te zien tot waar de whiskers gaan. Dit is echter geen exacte wetenschap, vooral wanneer variabelen niet

symmetrisch verdeeld zijn.

```
survey %>%
  ggplot(aes("", tv_hours)) +
  geom_boxplot()
```



Voor het huidige geval kunnen we zeggen dat de mensen die meer dan 8 uur tv kijken ofwel een onjuist getal hebben ingevuld, ofwel uitzonderlijke tv-kijkers zijn. Daarom verwijderen we alle waarden hoger dan 8.

```
survey %>%
  mutate(tv_hours = ifelse(tv_hours > 8, NA, tv_hours)) -> survey
```

8.4.4 Data inconsistencies

Een andere manier om op fouten te controleren is meer dan één variabele tegelijk te bekijken, en te controleren op duidelijke inconsistenties. Dit kunnen we zien als “regels” die worden overtreden. In de huidige dataset zouden we kunnen nagaan of alle gehuwde personen minstens 18 jaar oud moeten zijn. Aangezien alle waarnemingen mensen betreffen die minstens 18 jaar oud zijn, weten we dat deze regel niet geschonden is. Bijvoorbeeld regels in andere gevallen kunnen zijn:

- het vertrek van een vlucht moet plaatsvinden voordat deze kan aankomen. (Rekening houdend met verschillende tijdzones, uiteraard.)
- iemand wiens werkstatus “werkloos” is, kan geen gemeld inkomen hebben (tenzij rekening wordt gehouden met werkloosheidssuitkeringen).
- enz.

M.a.w., we gebruiken onze kennis over de variabelen en hun onderlinge verhoudingen om te kijken of er opvallende observaties zijn, die we desnoods moeten aanpassen.

Vaak zult u geen tijd hebben om alle mogelijke regels die je kunt bedenken te controleren. Bovendien zullen sommige regels gebaseerd zijn op bepaalde veronderstellingen die je moet controleren. Bijvoorbeeld, de leeftijd waarop men kan trouwen hangt af van het land, en kan dus lager of hoger zijn dan 18.

8.4.5 Missing values

In het echte leven worden gegevens meestal geleverd met ontbrekende waarden. De waarden kunnen ontbreken aan de start, of ze kunnen ontbreken omdat we ze verwijderd hebben als outliers of verkeerde waarden. In de volgende paragrafen zullen we zien hoe we ontbrekende waarden kunnen analyseren – bv. ontbreken ze willekeurig of niet? – en hoe ze te behandelen tijdens je analyse. Wij zijn verplicht te zeggen dat er ook technieken zijn die kunnen worden gebruikt om ontbrekende waarden te *schatten* op basis van de waarden voor andere attributen en andere waarnemingen met soortgelijke waarden. Dit wordt missing value *imputation* genoemd en is een apart veld op zich. Vanwege de complexiteit ervan zullen we het er hier niet over hebben, maar de geïnteresseerde lezer wordt verwezen naar deze handleiding van het mice-pakket.

Analyseren missing data

Er zijn drie verschillende manieren waarop ontbrekende gegevens kunnen voorkomen (zie theorie).

- Volledig willekeurig ontbrekend (MCAR)
- Willekeurig ontbrekend (MAR)
- Niet willekeurig ontbrekend (NMAR)

Hieronder illustreren we enkele technieken om de ontbrekende waarden in je gegevens te analyseren. De meest voor de hand liggende manier om na te gaan of je gegevens ontbrekende waarden bevatten is door naar de samenvatting te kijken.

```
summary(survey)
```

```
##      year          marital        age         race
##  Min.   :2000   Divorced     : 3383   Min.   :18.00   Black: 3129
##  1st Qu.:2002   Married     :10117   1st Qu.:33.00   Other: 1959
##  Median :2006   Never married: 5416   Median :46.00   White:16395
##  Mean    :2007   No answer   :     17   Mean    :47.18
```

```

## 3rd Qu.:2010 Separated : 743 3rd Qu.:59.00
## Max. :2014 Widowed : 1807 Max. :89.00
## NA's :76

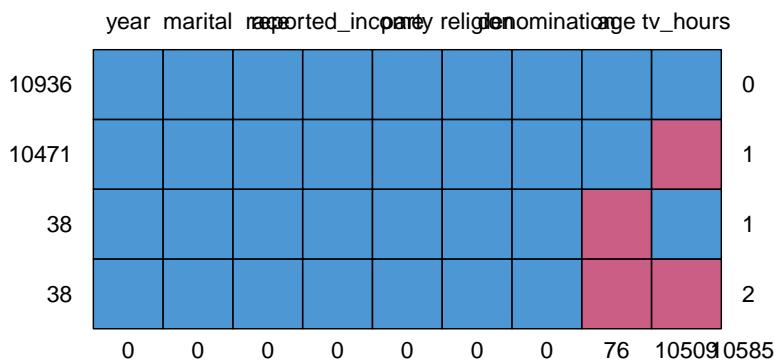
##           reported_income          party          religion
## $25000 or more:7363 Independent :4119 Protestant:10846
## Not applicable:7043 Democrat, weak :3690 Catholic : 5124
## $20000 - 24999:1283 Democrat, strong :3490 None     : 3523
## $10000 - 14999:1168 Republican, weak :3032 Christian: 689
## $15000 - 19999:1048 Independent, near dem:2499 Jewish    : 388
## Refused       : 975 Republican, strong :2314 Other     : 224
## (Other)        :2603 (Other)           :2339 (Other)   : 689

##           denomination      tv_hours
## Not applicable :10072 Min.   :0.000
## Other          : 2534 1st Qu.:1.000
## No denomination: 1683 Median :2.000
## Southern baptist: 1536 Mean   :2.659
## Baptist-dk which: 1457 3rd Qu.:4.000
## United methodist: 1067 Max.   :8.000
## (Other)         : 3134 NA's   :10509

```

Dit vertelt ons voor welke variabelen er ontbrekende gegevens zijn, en hoeveel. Het zegt ons echter niets over de relaties tussen ontbrekende waarden. Om naar *patronen* van ontbrekende gegevens te kijken, kunnen we gebruik maken van de `md.pattern` functie (missing data patterns) uit het pakket `mice`.

```
md.pattern(survey)
```



```

##           year marital race reported_income party religion denomination age
## 10936     1       1     1                      1       1       1                     1   1
## 10471     1       1     1                      1       1       1                     1   1

```

```

## 38      1      1      1      1      1      1      1      0
## 38      1      1      1      1      1      1      1      0
##      0      0      0      0      0      0      0     76
##      tv_hours
## 10936    1      0
## 10471    0      1
## 38       1      1
## 38       0      2
##      10509 10585

```

De uitvoer van `md.pattern` is een beetje cryptisch, maar laten we eens wat beter kijken. Elke kolom verwijst naar een van de variabelen, zoals is aangegeven. Elke rij is een *patroon* bestaande uit 1'en (gegevens ontbreken niet) en 0'en (gegevens ontbreken wel). De eerste rij, waarin alle variabelen een 1 hebben, is een patroon waarin geen van de variabelen ontbreekt. Dit wordt ook aangeduid door de nul in de laatste kolom. In de tweede rij wordt tv hours aangegeven met een nul, wat betekent dat voor dit patroon de variabele tv_hours ontbreekt. De laatste kolom geeft dus 1 ontbrekende waarde aan. Het laatste patroon is er een met 2 ontbrekende waarden, zoals aangegeven in de laatste kolom. Met name leeftijd en tv-uren ontbreken.

Het getal in de eerste kolom geeft aan hoeveel waarnemingen van elk patroon er zijn. Op die manier heeft het eerste patroon de meeste waarnemingen – 10936 personen zonder ontbrekende gegevens. Het laatste patroon (tv-uren en leeftijd ontbreken) komt 38 keer voor. De laatste rij is gelijk aan het aantal ontbrekende waarden voor elke variabele (dezelfde informatie die de samenvatting ons gaf). Tenslotte is het getal in de rechter benedenhoek het totaal aantal ontbrekende waarden.

De uitvoer van `md.patterns` (welke zowel een tabel als een visuele weergave is) kan ons tonen of het voorkomen van ontbrekende waarden met elkaar verband houdt. Bijvoorbeeld, als leeftijd ontbreekt, dan ontbreekt tv-uur ook? Dat laatste is niet het geval, want er zijn evenveel waarnemingen waar leeftijd ontbreekt en tv-uren niet, als waarnemingen waar leeftijd én tv-uren ontbreekt.

Naast `md.pattern` kunnen we ook nagaan of het voorkomen van ontbrekende waarden samenhangt met de *waarde* voor andere variabelen. Zo kunnen we ons afvragen of mensen van bepaalde religies of politieke voorkeuren meer of minder kans hebben om hun leeftijd of het aantal uren dat ze tv kijken op te geven. Deze patronen kunnen gecontroleerd worden met `ggplot/dplyr`, door een nieuwe variabele te creëren die aangeeft of een waarneming een ontbrekende waarde heeft of niet.

Laten we eens kijken naar leeftijd. We voegen een variabele toe die

aangeeft of leeftijd al dan niet ontbreekt. Merk op dat om dit te controleren, we een speciale functie nodig hebben. We kunnen age == NA niet gebruiken om te zien of leeftijd ontbreekt. In de laatste voorwaarde vergelijken we leeftijd met NA, d.w.z. we vergelijken leeftijd met iets dat we niet hebben. We kunnen nooit weten of leeftijd gelijk is aan iets dat we niet hebben, dus het resultaat daarvan is altijd NA, ongeacht de waarde van leeftijd. Neem als voorbeeld de variabelen a en b, waarvan a ontbreekt en b niet.

```
a <- NA
b <- 1
```

```
a == NA
## [1] NA

b == NA
## [1] NA
```

Beide logische condities geven NA terug, in plaats van de verwachte TRUE en FALSE.

Dus, hoe controleren we of iets NA is? Daar gebruiken we de speciale functie `is.na` voor.

```
is.na(a)
## [1] TRUE

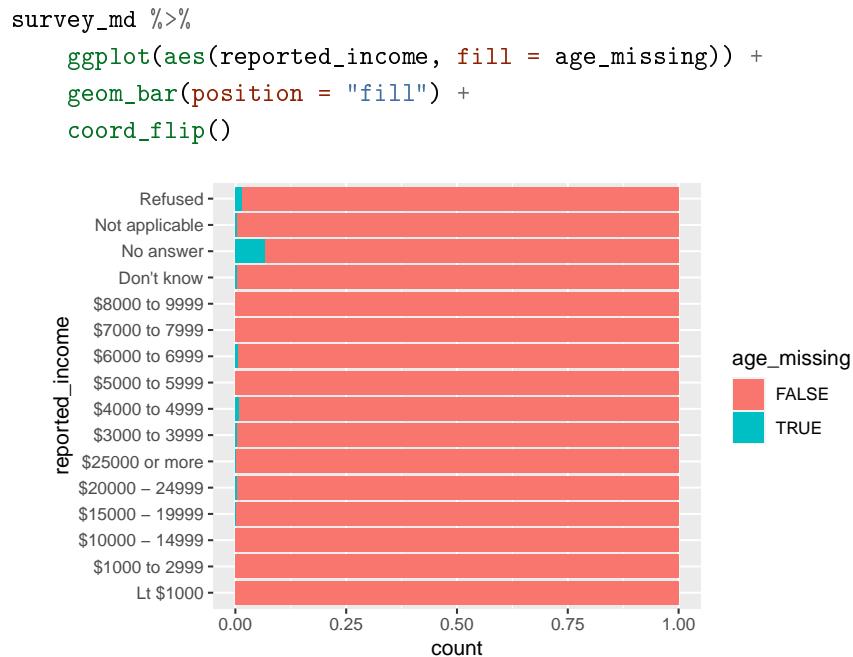
is.na(b)
## [1] FALSE
```

Dus, voor de survey data:

```
survey %>%
  mutate(age_missing = is.na(age),
        tv_missing = is.na(tv_hours)) -> survey_md
```

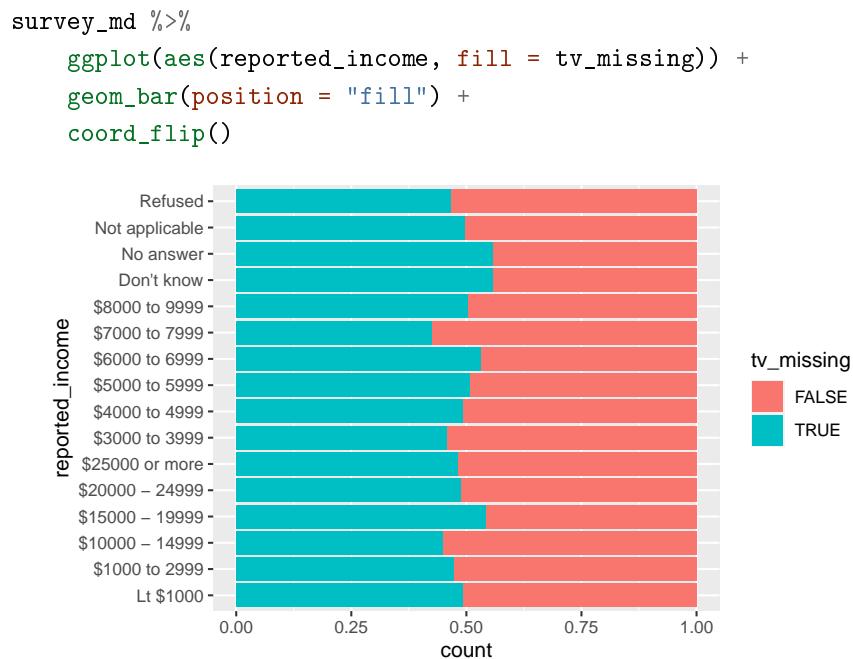
Merk op dat wij het data.frame met de bijkomende variabelen onder een andere naam opslaan, aangezien wij deze variabelen in de uiteindelijke analysefase niet nodig zullen hebben, alleen om de ontbrekende gegevens voorlopig te bekijken.

Wanneer we de ontbrekende/niet-ontbrekende variabele vergelijken met een categorische variabele, hebben wij in feite 2 categorische variabelen. We kunnen dus een grafiek of tabel gebruiken die geschikt is voor het vergelijken van categorische variabelen. Zoals we nu allemaal (zouden moeten) weten, kunnen we een staafdiagram gebruiken.



Uit het staafdiagram blijkt dat mensen die hun inkomen niet hebben opgegeven, vaker ook hun leeftijd niet hebben opgegeven.
10

We kunnen hetzelfde doen voor tv-uren. We zien dan dat het percentage ontbrekende waarden varieert - iets hoger of lager voor bepaalde inkomens - maar dat er geen duidelijke patronen zijn.

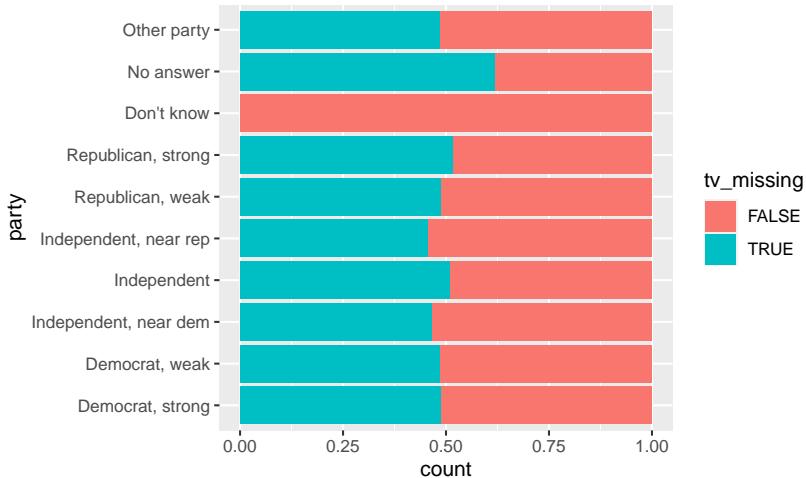


Natuurlijk kan het voorkomen van ontbrekende waarden worden

¹⁰ Men zou kunnen aanvoeren dat we ook de waarde “Geen antwoord” voor gemeld inkomen als NA hebben gecodeerd. In dit geval is echter besloten dat niet te doen om een duidelijk onderscheid tussen de speciale categorieën (Geweigerd, Niet van toepassing, Geen antwoord en Weet niet) te behouden.

vergeleken met meer dan één variabele, niet alleen inkomen. Hieronder tonen we de vergelijking van ontbrekende tv-uren met politieke voorkeur. Hier zien we duidelijk dat er slechts één antwoord is voor politieke partij dat alleen werd gebruikt voor waarnemingen waarbij de tv-uren niet ontbraken. Voor de andere politieke voorkeur-waarden is er geen duidelijk verband met het missing zijn van tvhours.

```
survey_md %>%
  ggplot(aes(party, fill = tv_missing)) +
  geom_bar(position = "fill") +
  coord_flip()
```



Als laatste voorbeeld kunnen we het ontbreken van het aantal tv-uren vergelijken met een continue variabele, laten we zeggen leeftijd. Zijn mensen die het aantal uren dat ze tv kijken niet hebben gerapporteerd over het algemeen ouder of jonger? De boxplots hieronder laten geen verschil zien.

```
survey_md %>%
  ggplot(aes(tv_missing, age, color = tv_missing)) +
  geom_boxplot()
```



Hoeveel vergelijkingen u maakt voor elke variabele waarvoor gegevens ontbreken, is aan jou – maar je moet redelijk bewijs verzamelen of de ontbrekende waarden MAR, MCAR of NMAR zijn.

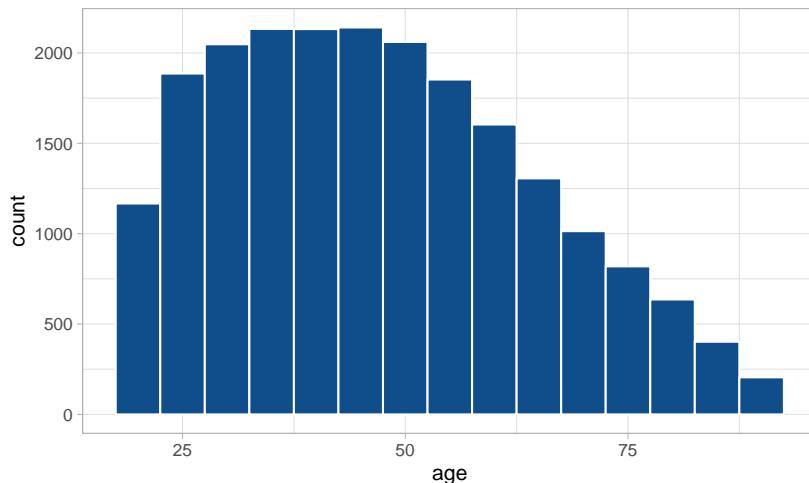
Werken met missing data

Hoewel we niet zullen zien hoe we ontbrekende waarden kunnen imputeren - *inschatten* -, is het toch belangrijk te weten hoe we ermee moeten werken.

Ten eerste, denk aan visualisaties. Wanneer we met ggplot werken, zullen ontbrekende waarden vaak automatisch genegeerd worden. Wanneer we bijvoorbeeld een histogram van de leeftijd proberen te maken, zal ggplot ons via een waarschuwing vertellen dat het enkele ontbrekende waarden negeerde.

```
survey %>%
  ggplot(aes(age)) +
  geom_histogram(binwidth = 5, fill = "dodgerblue4", color = "white") +
  theme_light()

## Warning: Removed 76 rows containing non-finite values (stat_bin).
```



De waarschuwing

```
## Warning: Removed 76 rows containing non-finite values (stat_bin).
```

vertelt ons wanneer en hoeveel ontbrekende waarden worden genegeerd. In het geval dat een categorische variabele ontbrekende waarden heeft, zal NA verschijnen als een aparte categorie. Neem bijvoorbeeld de dataset survey2 (die we hier even als voorbeeld zullen gebruiken), die ontbrekende waarden heeft voor verscheidene variabelen.

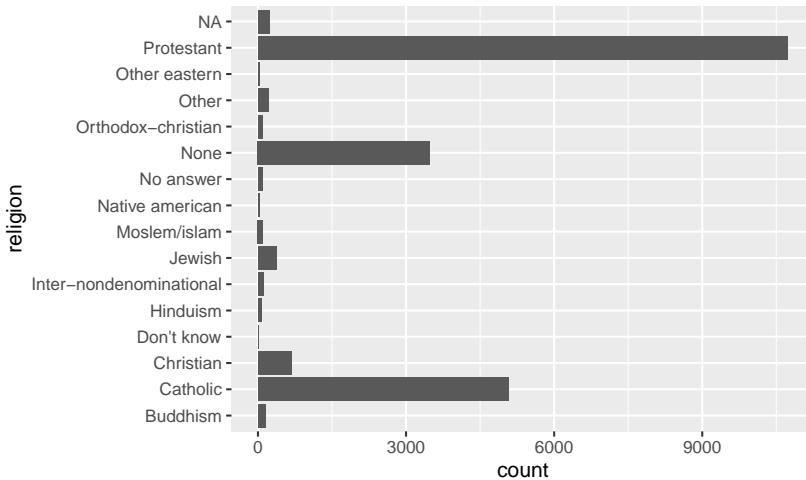
```
summary(survey2)
```

```
##      age           denomination       marital
##  Min.   :18.00   Not applicable :9960   Divorced    :3351
##  1st Qu.:33.00   Other          :2508   Married     :9999
##  Median  :46.00   No denomination:1665   Never married:5359
##  Mean    :47.18   Southern baptist:1521  No answer    :  17
##  3rd Qu.:59.00   Baptist-dk which:1439  Separated    : 738
##  Max.   :89.00   (Other)         :4159   Widowed    :1788
##  NA's    :280    NA's            : 231   NA's        : 231
##                  party          race        religion
##  Independent    :4080   Black: 3098   Protestant:10733
##  Democrat, weak :3658   Other: 1941   Catholic  : 5066
##  Democrat, strong:3453   White:16226  None     : 3485
##  Republican, weak:2997   NA's  :  218   Christian : 685
##  Independent, near dem:2479
##  (Other)          :4604
##  NA's            : 212
##      reported_income   tv_hours       year
##  $25000 or more:7294   Min.   :0.000   Min.   :2000
##  Not applicable:6973   1st Qu.:1.000   1st Qu.:2002
```

```
##   $20000 - 24999:1274   Median :2.000   Median :2006
##   $10000 - 14999:1159   Mean    :2.659   Mean    :2006
##   $15000 - 19999:1034   3rd Qu.:4.000   3rd Qu.:2010
##   (Other)      :3544     Max.   :8.000   Max.   :2014
##   NA's        : 205     NA's    :10596   NA's    :206
```

Laten we zeggen dat we een staafdiagram maken van de variabele godsdienst. De waarde NA verschijnt dan als een aparte categorie en krijgt zijn eigen staaf. Automatisch zal deze categorie worden uitgezet naast de andere, niet ertussen.

```
survey2 %>%
  ggplot(aes(religion)) +
  geom_bar() +
  coord_flip()
```



Als dusdanig behandelt ggplot ontbrekende waarden zonder probleem en vrij transparant. Hetzelfde is niet waar wanneer we naar numerieke berekeningen gaan.

Laten we beginnen met het goede nieuws. Als wij een frequentietabel maken van een categorische waarde met ontbrekende waarden, zullen de NA's net zo worden beschouwd als elke andere waarde. Wij kunnen bijvoorbeeld een frequentietabel maken ter ondersteuning van de bovenstaande grafiek voor survey2. NA wordt zelfs meegesorteerd door onze arrange in dit geval.

```
survey2 %>%
  group_by(religion) %>%
  summarize(frequency = n()) %>%
  mutate(relative_frequency = frequency/sum(frequency)) %>%
  arrange(-frequency) %>%
  pander
```

`summarise()` ungrouping output (override with `.groups` argument)

religion	frequency	relative_frequency
Protestant	10733	0.4996
Catholic	5066	0.2358
None	3485	0.1622
Christian	685	0.03189
Jewish	384	0.01787
NA	228	0.01061
Other	220	0.01024
Buddhism	147	0.006843
Inter-	107	0.004981
nondenominational		
Moslem/islam	103	0.004794
Orthodox-christian	94	0.004376
No answer	92	0.004282
Hinduism	70	0.003258
Other eastern	32	0.00149
Native american	22	0.001024
Don't know	15	0.0006982

Helaas is dat niet het geval wanneer we maten voor centraliteit of spreiding gaan berekenen. Per definitie, wanneer je een functie toepast op een vector die ontbrekende waarden bevat, zal de functie een ontbrekende waarde teruggeven. Beschouw de vector hieronder, waarvan we het gemiddelde en de som willen weten

```
x <- c(5, 6, 12, NA, 43)
mean(x)

## [1] NA

sum(x)

## [1] NA
```

Net als de waarschuwing die we kregen toen we `ggplot` gebruikten met ontbrekende waarden, is dit resultaat een waarschuwing. Het waarschuwt ons dat we iets willen berekenen over een vector die gedeeltelijk mist. De waarschuwing is hier echter vrij sterk; ze geeft ons niets wat we kunnen gebruiken.

Om dit te omzeilen heeft elk van deze functies een argument dat `na.rm` heet – wat “NA verwijderen” betekent. Als we zeggen `na.rm = T`, dan worden ontbrekende waarden verwijderd en genegeerd, en de functie berekent het resultaat met de overgebleven waarde. Dus,

```
mean(x, na.rm = T)
## [1] 16.5

sum(x, na.rm = T)
## [1] 66
```

Het feit dat we expliciet moeten aangeven dat we ontbrekende waarden negeren is het veiligheidsmechanisme van R dat voorkomt dat we ontbrekende waarden per ongeluk negeren.

Als we dus maten van centraliteit en spreiding willen berekenen voor, laten we zeggen, leeftijd, zullen we dit argument moeten gebruiken als we iets willen bereiken.¹¹

```
survey %>%
  summarize(min = min(age, na.rm = T),
            mean = mean(age, na.rm = T),
            max = max(age, na.rm = T),
            iqr = IQR(age, na.rm = T))

## # A tibble: 1 x 4
##       min   mean   max   iqr
##     <int> <dbl> <int> <dbl>
## 1     18    47.2    89    26
```

Tot slot, wat gebeurt er als we correlaties willen berekenen? Bijvoorbeeld, hoe zijn leeftijd en tv-uren gecorreleerd. Beide hebben ontbrekende waarden. We kunnen het volgende proberen:

```
survey %>%
  select(age, tv_hours) %>%
  cor(na.rm = T)

## Error in cor(., na.rm = T): unused argument (na.rm = T)
```

Oeps, dat was wishful thinking van mijn kant. Het lijkt erop dat het `na.rm` argument niet bestaat voor de `cor` functie. Tot zover de consistentie in de basis R functies.

Om correlaties te berekenen, moeten we ervoor zorgen dat we alleen rijen zonder ontbrekende waarden beschouwen. We kunnen dit doen met de `na.omit` functie. Deze functie verwijdert – omits – alle rijen die een of meer ontbrekende waarden hebben.

```
survey %>%
  select(age, tv_hours) %>%
  na.omit() %>%
  summary()
```

¹¹ Ja, we moeten dit herhalen voor elke functie die we willen gebruiken, en nee, er is geen gemakkelijkere manier.

```
##           age          tv_hours
##   Min.   :18.00   Min.   :0.000
##   1st Qu.:33.00  1st Qu.:1.000
##   Median :45.00  Median :2.000
##   Mean    :47.13  Mean   :2.659
##   3rd Qu.:60.00  3rd Qu.:4.000
##   Max.    :89.00  Max.   :8.000
```

Dan kunnen we de correlatie berekenen.

```
survey %>%
  select(age, tv_hours) %>%
  na.omit() %>%
  cor()

##           age   tv_hours
## age     1.0000000 0.1895392
## tv_hours 0.1895392 1.0000000
```

De `na.omit` functie kan wat gevvaarlijk zijn en zou alleen gebruikt moeten worden in uitzonderlijke gevallen als deze. Het kan een enorm verschil maken als we het vóór `select` doen. Beschouw opnieuw de `survey2` dataset, waar alle variabelen een aantal ontbrekende waarden hebben.

Deze dataset heeft het volgende aantal rijen.

```
survey2 %>% nrow()

## [1] 21483
```

Als we leeftijd en tv-uren selecteren, en rijen met ontbrekende waarden verwijderen, houden we het volgende aantal rijen over

```
survey2 %>%
  select(age, tv_hours) %>%
  na.omit() %>%
  nrow()

## [1] 10732
```

Maar als we eerst de rijen met ontbrekende waarden verwijderen, en dan de twee variabelen selecteren, houden we slechts het volgende aantal rijen over.

```
survey2 %>%
  na.omit() %>%
  select(age, tv_hours) %>%
  nrow()
```

```
## [1] 9974
```

Zie je wat er anders is? Kijk maar eens goed. Als we de verwijdering uitvoeren voor de select, zal het rekening houden met ontbrekende waarden voor ALLE variabelen. Als we de selectie eerst doen, wordt alleen gecontroleerd op de variabelen die we behouden. Deze nuance kan in de praktijk een groot verschil maken. Het gebruik van de `na.omit` functie moet altijd een waarschuwing in je achterhoofd oproepen dat je voorzichtig moet zijn.

Het is raadzaam steeds voorzichtig te zijn bij het werken met ontbrekende waarden. En met deze wijze woorden beëindigen we onze data cleaning inspanningen, en gaan we over tot transformaties.

8.5 Data transformatie

Terwijl het opschonen van gegevens gericht is op het vinden van fouten, gaat het bij het omvormen van gegevens om het analyseren ervan te vergemakkelijken, of resultaten te verbeteren. Er zijn vele manieren om dat te doen.

Ten eerste kunnen we het aantal niveaus in een categorische variabele verminderen, zodat er niet te veel verschillende categorieën worden getoond. Ten tweede kunnen we ook nieuwe variabelen creëren op basis van bestaande variabelen. Dit laatste wordt ook wel het verrijken van gegevens genoemd. Bovendien kunnen we variabelen aanpassen om ze gemakkelijker te interpreteren te maken. Bijvoorbeeld door afstanden in km te gebruiken in plaats van in mijlen (of andersom als je Brits of Amerikaans bent). Verder kunnen we continue variabelen discreet maken - door ze in categorische variabelen te veranderen - zodat we andere analyses of visualisaties kunnen gebruiken. Tenslotte zullen we ook enkele transformaties bespreken die nuttig zijn bij het visualiseren van gegevens, in het bijzonder om variabelen te ordenen.

Veel dingen te doen, dus tijd om te beginnen!

8.5.1 Discretisern van continue variabelen

Het omzetten van een continue variabele in een categorische variabele wordt discretisatie genoemd. Verschillende functies om dit te doen worden aangeboden door ggplot.

- `cut_width`: intervallen van een bepaalde breedte maken
- `cut_interval`: het creëren van een specifiek aantal intervallen van gelijke breedte
- `cut_number`: het creëren van een specifiek aantal intervallen met een gelijk aantal waarnemingen.

Bijvoorbeeld, we kunnen tv_hours discretiseren in intervallen van breedte 4.¹²

```
survey %>%
  mutate(tv_hours = cut_width(tv_hours, width = 4, boundary = 0)) %>%
  count(tv_hours)

## # A tibble: 3 x 2
##   tv_hours     n
##   <fct>     <int>
## 1 [0,4]     9421
## 2 (4,8]    1553
## 3 <NA>    10509
```

Als alternatief kunnen we tv_uren_hours in 4 gelijke intervallen verdelen.

```
survey %>%
  mutate(tv_hours = cut_interval(tv_hours, n = 4)) %>%
  count(tv_hours)

## # A tibble: 5 x 2
##   tv_hours     n
##   <fct>     <int>
## 1 [0,2]     6059
## 2 (2,4]    3362
## 3 (4,6]    1172
## 4 (6,8]     381
## 5 <NA>    10509
```

Of we kunnen tv_hours in drie intervallen verdelen die een gelijk aantal waarnemingen bevatten.

```
survey %>%
  mutate(tv_hours = cut_number(tv_hours, n = 3)) %>%
  count(tv_hours)

## # A tibble: 4 x 2
##   tv_hours     n
##   <fct>     <int>
## 1 [0,2]     6059
## 2 (2,3]    1956
## 3 (3,8]    2959
## 4 <NA>    10509
```

Hierbij moet worden opgemerkt dat de intervallen niet helemaal evenveel waarnemingen bevatten. Dat komt omdat er veel waarnemingen zijn met een unieke waarde, die niet verder kunnen worden

¹² Het grensargument is een optioneel argument om te bepalen waar het interval moet beginnen.

gesplitst. Maar toch zal `cut_number` zijn best doen, wat beter zal uitpakken als de waarden van de variabele unieker zijn. Bijvoorbeeld, leeftijd leent zich hier beter voor:

```
survey %>%
  mutate(age = cut_number(age, n = 5)) %>%
  count(age)

## # A tibble: 6 x 2
##   age      n
##   <fct>    <int>
## 1 [18,31]  4656
## 2 (31,41]  4305
## 3 (41,51]  4207
## 4 (51,63]  4149
## 5 (63,89]  4090
## 6 <NA>      76
```

Elk van de discretisatie-functies staat ons toe om de namen van de categorieën aan te passen – in plaats van de standaard interval notatie door een vector van namen mee te geven aan het `labels` argument.

Bijvoorbeeld:

```
survey %>%
  mutate(age_category = cut_number(age, n = 3, labels = c("Young","Middle-aged","Old"))) %>%
  group_by(age_category) %>%
  summarize(min = min(age), max = max(age), frequency = n())

## # A tibble: 4 x 4
##   age_category   min   max frequency
##   <fct>        <int> <int>     <int>
## 1 Young          18    37     7234
## 2 Middle-aged   38    54     7117
## 3 Old            55    89     7056
## 4 <NA>           NA    NA      76
```

In het laatste voorbeeld hebben wij de gediscretiseerde variabele onder een andere naam opgeslagen. Dit is zeer aan te raden, om de oorspronkelijke, meer gedetailleerde gegevens niet te verliezen.

8.5.2 Herschalen van continue variabelen

Wanneer we continue variabelen hebben, kunnen we ook de schaal aanpassen. We kunnen bijvoorbeeld het percentage `tv_hours` per dag berekenen door het aantal `tv_hours` te delen door 24.

```
survey %>%
  mutate(tv_per_day = tv_hours/24) %>%
  summary
```

```

##      year          marital        age         race
##  Min.   :2000   Divorced    : 3383   Min.   :18.00   Black: 3129
##  1st Qu.:2002  Married     :10117   1st Qu.:33.00  Other: 1959
##  Median :2006  Never married: 5416   Median :46.00  White:16395
##  Mean   :2007  No answer    :    17   Mean   :47.18
##  3rd Qu.:2010  Separated    :   743   3rd Qu.:59.00
##  Max.   :2014  Widowed     :1807   Max.   :89.00
##                               NA's    :76

##      reported_income       party        religion
## $25000 or more:7363  Independent      :4119  Protestant:10846
## Not applicable:7043  Democrat, weak   :3690  Catholic  : 5124
## $20000 - 24999:1283  Democrat, strong  :3490  None     : 3523
## $10000 - 14999:1168  Republican, weak  :3032  Christian : 689
## $15000 - 19999:1048  Independent, near dem:2499  Jewish    : 388
## Refused      : 975   Republican, strong  :2314  Other     : 224
## (Other)       :2603   (Other)           :2339  (Other)   : 689

##      denomination      tv_hours      tv_per_day
## Not applicable :10072  Min.   :0.000  Min.   :0.000
## Other          : 2534  1st Qu.:1.000  1st Qu.:0.042
## No denomination: 1683  Median :2.000  Median :0.083
## Southern baptist: 1536  Mean   :2.659  Mean   :0.111
## Baptist-dk which: 1457 3rd Qu.:4.000  3rd Qu.:0.167
## United methodist: 1067  Max.   :8.000  Max.   :0.333
## (Other)         : 3134  NA's    :10509  NA's    :10509

```

Andere veel voorkomende transformaties zijn:

- verschillende munteenheden (euro vs dollar, enz.)
- verschillende meetschalen (mijl vs km, inch vs cm, enz.)
- verschillende tijdzones of tijdseenheden (waarover later meer).

8.5.3 Toevoegen van calculated variables

De toevoeging van berekende variabelen is vergelijkbaar met het herschalen van variabelen. Het enige verschil is dat herschalen slechts betrekking heeft op één variabele, terwijl berekende variabelen betrekking kunnen hebben op verschillende variabelen.

Wij kunnen bijvoorbeeld het geboortejaar berekenen voor de personen in onze gegevens.

```

survey %>%
  mutate(year_of_birth = year - age) %>%
  select(year, age, year_of_birth) %>%
  summary

##      year          age        year_of_birth
##  Min.   :2000   Min.   :17.00   Min.   :1830
##  1st Qu.:2002  1st Qu.:17.00  1st Qu.:1830
##  Median :2006  Median :17.00  Median :1830
##  Mean   :2007  Mean   :17.00  Mean   :1830
##  3rd Qu.:2010  3rd Qu.:17.00  3rd Qu.:1830
##  Max.   :2014  Max.   :17.00  Max.   :1830
##                               NA's    :76

```

```
##   Min. :2000   Min. :18.00   Min. :1911
## 1st Qu.:2002   1st Qu.:33.00   1st Qu.:1947
## Median :2006   Median :46.00   Median :1960
## Mean   :2007   Mean   :47.18   Mean   :1959
## 3rd Qu.:2010   3rd Qu.:59.00   3rd Qu.:1973
## Max.   :2014   Max.   :89.00   Max.   :1996
##          NA's    :76        NA's    :76
```

8.5.4 Factoren transformeren

Bij de transformatie van categorische variabelen is het vaak de bedoeling het aantal waarden te verminderen. Dit kan op verschillende manieren gebeuren:

- het combineren van waarden die op elkaar lijken tot één enkele waarde
- het combineren van niet vaak voorkomende waarden in een “Diverse” of “Andere” waarde.

De eerste manier kan worden gedaan met `fct_collapse`, die factor niveaus samenvouwt (collapsed) tot een nieuw niveau. De tweede manier kan worden bereikt met `fct_lump`, die niet vaak voorkomende waarden samenvoegt in één aparte groep (lump)

Het samenvouwen van een factor kan als volgt worden gedaan. Voor elke groep van niveaus die je wilt samenvouwen, maak je een vector.

Vervolgens kan je elke groep een nieuwe naam geven.¹³ Alle niveaus die je niet noemt, blijven onaangeroerd.

```
fct_collapse(factor,
  new_group_1 = c("old_level_1", "old_level_2", "..."),
  new_group_2 = c("old_level_a", "old_level_b", "..."),
  ...)
```

In onze survey-data kunnen we de partijvariabele in verschillende groepen indelen: ‘Democrat’, ‘Republican’, ‘Independent’ en ‘Other’. We slaan de nieuwe variabele op als `party_group`:

```
survey %>%
  mutate(party_group = fct_collapse(party,
    Other = c("No answer",
              "Don't know",
              "Other party"),
    Republican = c("Republican, strong",
                  "Republican, weak"),
    Independent = c("Independent, near rep",
                  "Independent",
```

¹³ De oplettende lezer heeft misschien een overeenkomst opgemerkt tussen `fct_collapse` en `fct_recode`. Inderdaad, je zou hetzelfde resultaat kunnen bereiken met de laatste functie, maar je zou meer typewerk hebben.

```

    "Independent, near dem"),
Democrat = c("Democrat, weak",
            "Democrat, strong")
)) -> survey

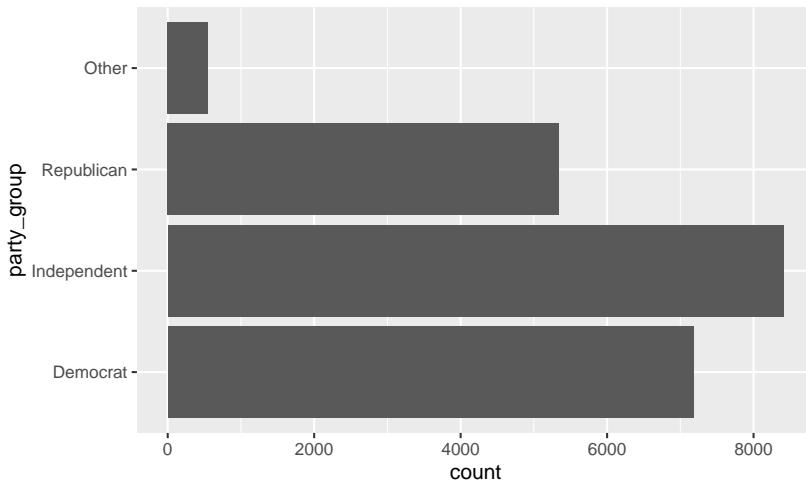
```

Transformaties als deze kunnen op zichzelf al nuttig zijn: het aantal categorieën verminderen, zoals in deze plot.

```

survey %>%
  ggplot(aes(party_group)) +
  geom_bar() +
  coord_flip()

```



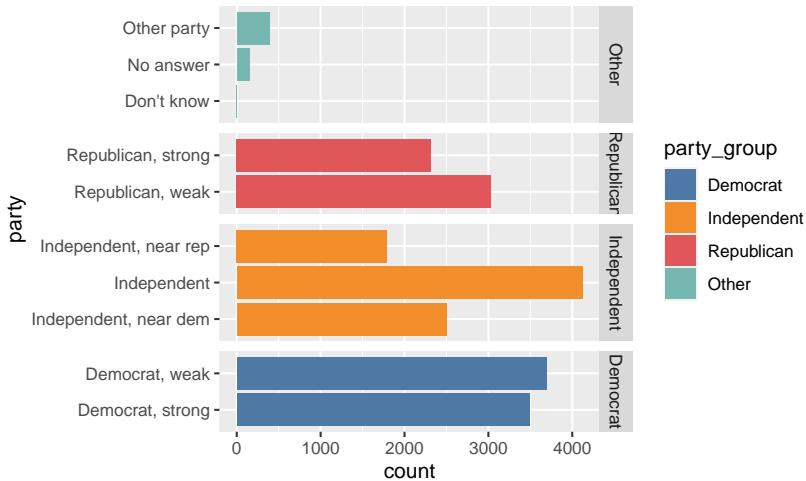
Maar ze kunnen net zo goed gebruikt worden in combinatie met de originele levels. Bijvoorbeeld om de visuals te verbeteren:¹⁴

```

library(ggthemes)
survey %>%
  ggplot(aes(party, fill = party_group)) +
  geom_bar() +
  facet_grid(fct_rev(party_group) ~ .,
             scales = "free", space = "free") +
  coord_flip() +
  scale_fill_tableau()

```

¹⁴ In dit voorbeeld gebruiken we de `scale_fill_tableau` kleurenschaal uit het pakket `ggthemes` om een kleurenschaal te krijgen waarbij de Democraten blauw zijn en de Republikeinen rood. Verder gebruiken we `fct_rev` om de volgorde van de factoren om te keren. `fct_rev` zal zo dadelijk worden besproken.



Als alternatief kunnen we `fct_lump` gebruiken om een “Andere” categorie te maken. Kijk bijvoorbeeld eens naar de religies.

```
survey %>%
  count(religion)

## # A tibble: 15 x 2
##   religion      n
##   <fct>     <int>
## 1 Buddhism     147
## 2 Catholic    5124
## 3 Christian    689
## 4 Don't know    15
## 5 Hinduism      71
## 6 Inter-nondenominational 109
## 7 Jewish        388
## 8 Moslem/islam   104
## 9 Native american  23
## 10 No answer     93
## 11 None       3523
## 12 Orthodox-christian  95
## 13 Other        224
## 14 Other eastern   32
## 15 Protestant   10846
```

Laten we zeggen dat we alleen de 5 meest voorkomende godsdiensten willen behouden. Dat kunnen we als volgt doen.

```
survey %>%
  mutate(religion = fct_lump(religion, n = 5)) %>%
  count(religion)

## # A tibble: 6 x 2
```

```
##   religion      n
##   <fct>     <int>
## 1 Catholic    5124
## 2 Christian   689
## 3 Jewish      388
## 4 None        3523
## 5 Protestant  10846
## 6 Other       913
```

Het label “Other” kan naar believen worden gewijzigd.

```
survey %>%
  mutate(religion = fct_lump(religion, n = 5, other_level = "Other religions")) %>%
  count(religion)

## # A tibble: 6 x 2
##   religion      n
##   <fct>     <int>
## 1 Catholic    5124
## 2 Christian   689
## 3 Jewish      388
## 4 None        3523
## 5 Protestant  10846
## 6 Other religions  913
```

In plaats van het aantal te behouden niveaus op te geven, kan je ook een minimale relatieve frequentie opgeven met het `prop` argument.

```
survey %>%
  mutate(religion = fct_lump(religion, prop = 0.02)) %>%
  count(religion)

## # A tibble: 5 x 2
##   religion      n
##   <fct>     <int>
## 1 Catholic    5124
## 2 Christian   689
## 3 None        3523
## 4 Protestant  10846
## 5 Other       1301
```

Meer nog dan het cleanen van data, zijn alle transformaties die we gezien hebben zeer iteratief van aard, en kunnen ze soms alleen gebruikt worden voor specifieke analyses. We willen bijvoorbeeld misschien de weinig voorkomende godsdiensten op één hoop gooien om een staafdiagram te maken zonder al te veel balken – maar we willen

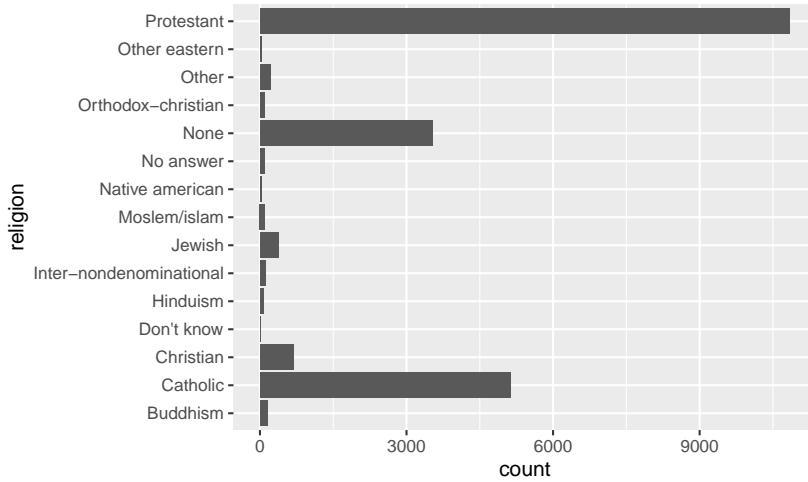
waarschijnlijk niet de weinig voorkomende godsdiensten helemaal verwijderen. Transformaties zullen dus vaak gebeuren in de opbouw naar een grafiek of tabel, en niet permanent in de gegevens worden opgeslagen. Dit geldt vooral voor de laatste functies die we hier zullen bespreken: herordeningsfuncties.

8.6 Data transformaties tijdens data visualisatie

Bij het visualiseren van categorische variabelen, willen we vaak de volgorde van de niveaus veranderen op basis van frequentie of op basis van een andere variabele. We zagen al `fct_relevel` om handmatig niveaus te herschikken, maar het is niet erg geschikt om automatisch niveaus te herschikken. Hiervoor zullen we twee nieuwe functies gebruiken: `fct_infreq` en `fct_reorder`.

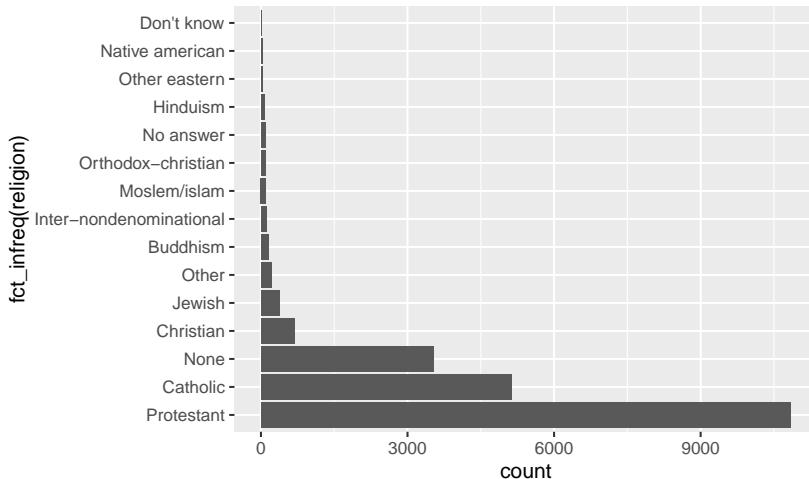
We beginnen met de volgende grafiek.

```
survey %>%
  ggplot(aes(religion)) +
  geom_bar() +
  coord_flip()
```



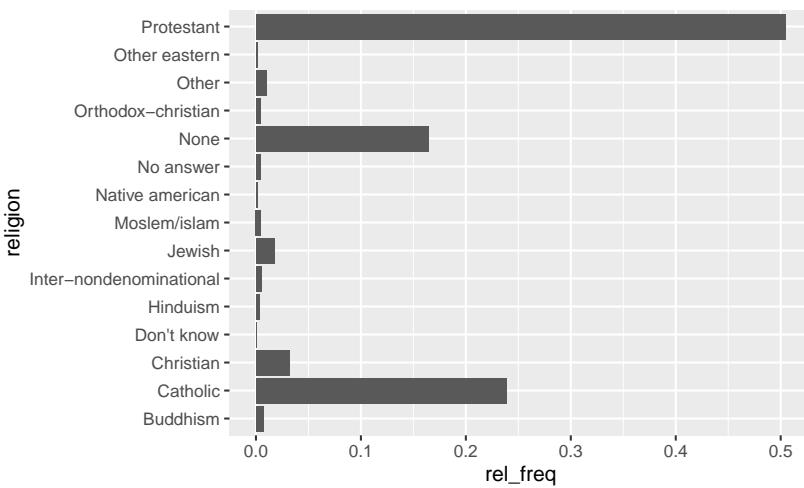
Een factor kan worden geordend op basis van de (in)frequentie van elk niveau met behulp van de functie `fct_infreq`. Het gebruik ervan is eenvoudig. We kunnen kiezen om de functie rechtstreeks in `ggplot` toe te voegen, of om de religie variabele bij te werken met `mutate` alvorens te plotten.

```
survey %>%
  ggplot(aes(fct_infreq(religion))) +
  geom_bar() +
  coord_flip()
```



`fct_infreq` zal voor elk van de labels - religies in dit geval - tellen hoe vaak het voorkomt, en de niveaus overeenkomstig te herschikken. Stel nu dat we geen absolute frequenties willen zoals in de laatste plot, maar relatieve. In dat geval moeten we ze zelf berekenen en `geom_col` gebruiken.

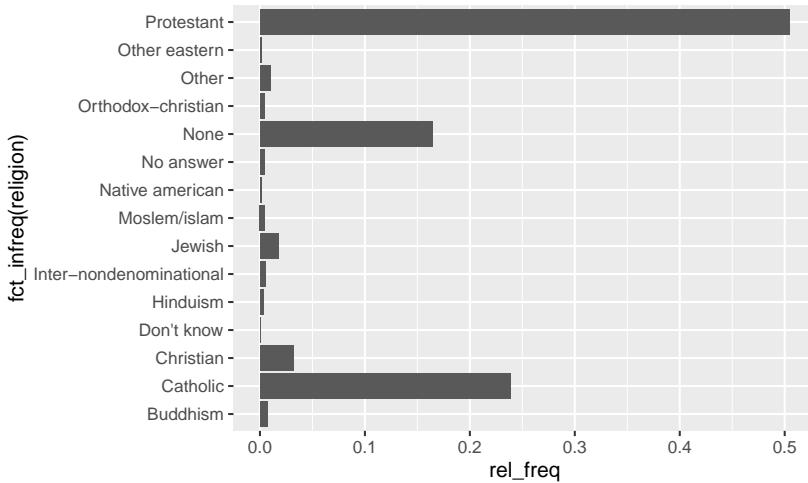
```
survey %>%
  group_by(religion) %>%
  summarise(freq = n()) %>%
  mutate(rel_freq = freq/sum(freq)) %>%
  ggplot(aes(x = religion, y = rel_freq)) +
  geom_col() +
  coord_flip()
```



We kunnen nu de relatieve frequenties uit de grafiek aflezen. Laten we nu de staven nog eens ordenen.

```
survey %>%
```

```
group_by(religion) %>%
summarise(freq = n()) %>%
mutate(rel_freq = freq/sum(freq)) %>%
ggplot(aes(x = fct_infreq(religion), y = rel_freq)) +
geom_col() +
coord_flip()
```



Oeps, dat leek niet te werken. Waarom niet? Laten we eens kijken naar de gegevens die we aan ggplot gaven.

```
survey %>%
  group_by(religion) %>%
  summarise(freq = n()) %>%
  mutate(rel_freq = freq/sum(freq))

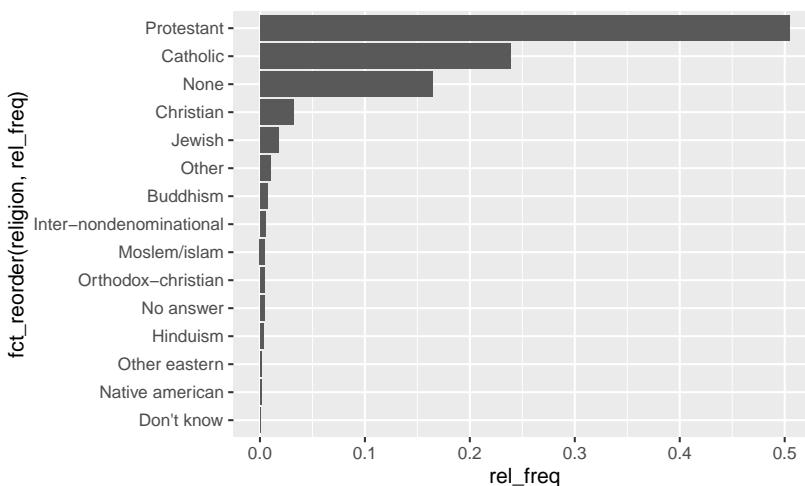
## # A tibble: 15 x 3
##   religion      freq  rel_freq
##   <fct>       <int>    <dbl>
## 1 Buddhism     147  0.00684
## 2 Catholic    5124  0.239
## 3 Christian   689  0.0321
## 4 Don't know  15  0.000698
## 5 Hinduism    71  0.00330
## 6 Inter-nondenominational 109  0.00507
## 7 Jewish      388  0.0181
## 8 Moslem/islam 104  0.00484
## 9 Native american 23  0.00107
## 10 No answer  93  0.00433
## 11 None       3523 0.164
## 12 Orthodox-christian 95  0.00442
## 13 Other      224  0.0104
## 14 Other eastern 32  0.00149
```

```
## 15 Protestant 10846 0.505
```

De gegevens – een frequentietabel – bevat een rij voor elke religie. Als we `fct_infreq` gebruiken op deze tabel, dan komen alle religies één keer voor, dus ze komen allemaal even vaak voor. `fct_infreq` probeert impliciet frequenties te berekenen, maar dat hebben we al gedaan. Het resultaat is dat de ordening mislukt. Het is vergelijkbaar met het gebruik van `geom_bar` op basis van een frequentietabel – we proberen de frequenties twee keer te berekenen, wat resulteert in onbedoelde plots.

Dus, wat kunnen we in plaats daarvan doen? Wel, we willen de religies ordenen op basis van frequentie. Dat zou niet moeilijk mogen zijn, want de frequentie is er al. De functie `fct_reorder` kan ons daarbij helpen. In tegenstelling tot `fct_infreq` zal deze een tweede variabele gebruiken die we opgeven om de factor te ordenen. Dus:

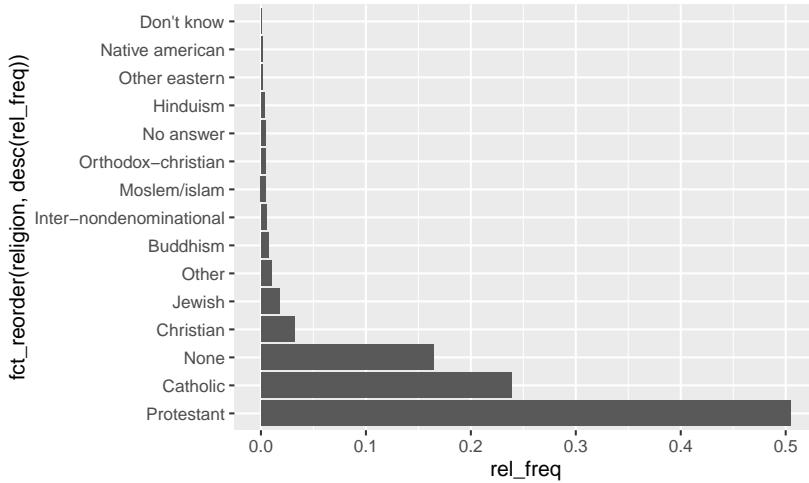
```
survey %>%
  group_by(religion) %>%
  summarise(freq = n()) %>%
  mutate(rel_freq = freq/sum(freq)) %>%
  ggplot(aes(x = fct_reorder(religion, rel_freq), y = rel_freq)) +
  geom_col() +
  coord_flip()
```



Merk op dat we tweemaal verwijzen naar `rel_freq`: eenmaal om te ordenen, en eenmaal om te gebruiken als y-as. Merk ook op dat de volgorde van de staven is omgekeerd: `fct_infreq` zal altijd ordenen van meest naar minst frequent, terwijl onze huidige configuratie met `fct_reorder` ordent van minst naar meest frequent. We kunnen dit eenvoudig veranderen met `desc()`, zoals we eerder hebben gedaan.

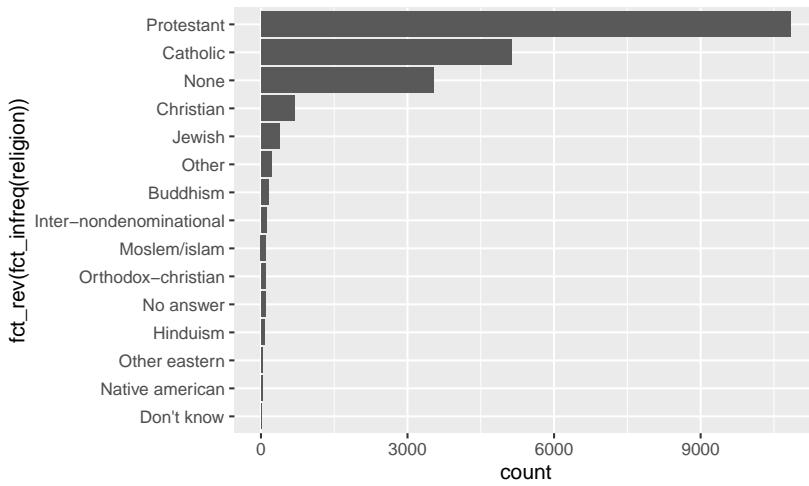
```
survey %>%
  group_by(religion) %>%
```

```
summarise(freq = n()) %>%
mutate(rel_freq = freq/sum(freq)) %>%
ggplot(aes(x = fct_reorder(religion, desc(rel_freq)), y = rel_freq)) +
geom_col() +
coord_flip()
```



Als alternatief kunnen we de `fct_rev` functie gebruiken: deze functie zal de volgorde van een factor omkeren. Zo kunnen we bijvoorbeeld de volgorde van `fct_infreq` omkeren.

```
survey %>%
ggplot(aes(fct_rev(fct_infreq(religion)))) +
geom_bar() +
coord_flip()
```



Dus, samenvattend:

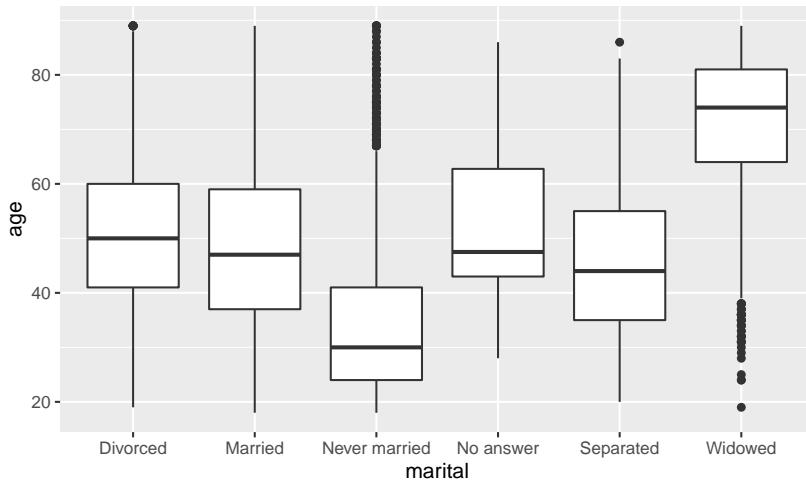
- `fct_infreq(f)`: herorden de niveaus van factor f van meest naar minst frequent

- `fct_reorder(f, x)`: herschik de niveaus van factor f volgens variabele x
- `fct_rev(f)`: draai de volgorde van de niveaus van factor f om

In het algemeen moet je `fct_infreq` alleen gebruiken op de originele gegevens, en als je een frequentietabel als invoer hebt voor `ggplot`, moet je `fct_reorder` gebruiken.

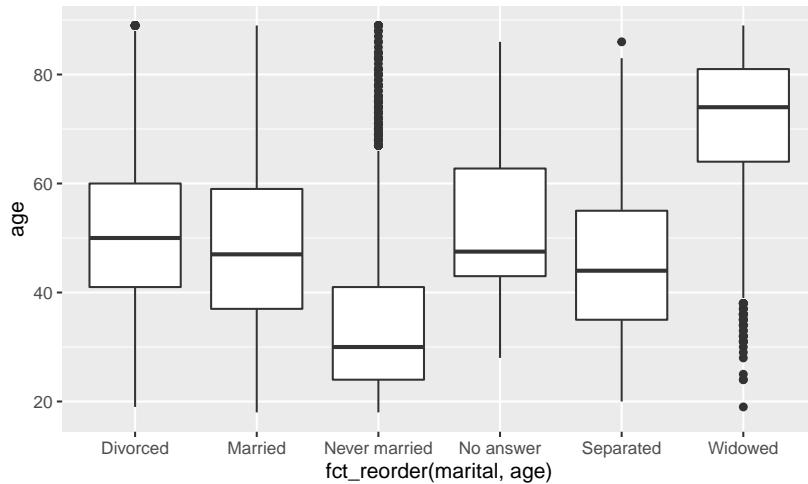
Er is echter nog één ding dat we moeten behandelen. `fct_reorder(f,x)` kan herordenen op elke variabele x, niet alleen op frequentie. Kijk bijvoorbeeld eens naar de volgende plot.

```
survey %>%
  ggplot(aes(marital, age)) +
  geom_boxplot()
```



Deze plot toont de verdeling van de leeftijd voor verschillende burgerlijke standen. Laten we zeggen dat we deze boxplots willen sorteren op leeftijd. We gebruiken `fct_reorder` net zoals we eerder deden.

```
survey %>%
  ggplot(aes(fct_reorder(marital, age), age)) +
  geom_boxplot()
```



Ook hier zijn de resultaten niet zoals we zouden verwachten. Wat is er anders dan bij het vorige gebruik met frequenties?

There are actually two differences.

1. Vroeger hadden we één enkele frequentie voor elke godsdienst, waardoor het gemakkelijk was ze te sorteren.

Nu hebben we voor elke burgerlijke staat veel personen met verschillende leeftijden. We moeten ze samenvatten in een enkele waarde, zoals het gemiddelde of de mediaan. Dit kan gedaan worden door het .fun argument toe te voegen aan factor reorder.

`fct_reorder(marital, age, .fun = median)`

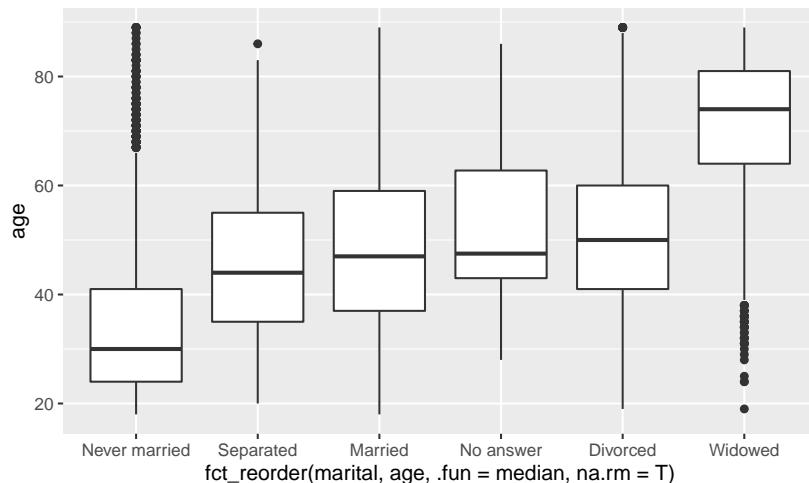
2. Sommige personen hebben geen leeftijd, maar een ontbrekende waarde.

Het berekenen van een functie met ontbrekende waarden leidt tot een ontbrekende waarde. We moeten ervoor zorgen dat ontbrekende waarden worden genegeerd. Elk argument dat we toevoegen aan `fct_reorder` na het .fun argument zal worden beschouwd als een argument voor deze functie. Dus, het volgende zal de marital waarden correct sorteren

`fct_reorder(marital, age, .fun = median, na.rm = T)`

Laat ons het proberen:

```
survey %>%
  ggplot(aes(fct_reorder(marital, age, .fun = median, na.rm = T), age)) +
  geom_boxplot()
```



Dit ziet er beter uit. Dus, laten we het nog eens samenvatten.

- **fct_infreq(f)**: herorden de niveaus van factor f van meest naar minst frequent
- **fct_reorder**: herschik de niveaus van factor f volgens variabele x
 - `fct_reorder(f, x)` als we zeker weten dat er een enkele x-waarde is voor elk f-niveau
 - `fct_reorder(f, x, .fun = summarize_function)` als er meer dan één x-waarde kan zijn voor een bepaald f-niveau. De `summarize_function` zal worden gebruikt om meerdere waarden te combineren. Dit kan zijn gemiddelde, mediaan, som, lengte, ... elke functie die een enkele waarde teruggeeft.
 - Als we extra argumenten aan de `summarize` functie moeten meegeven, zoals `na.rm = T`, kunnen we dit als volgt doen:
`fct_reorder(f, x, .fun = summarize_function, na.rm = T).`
- **fct_rev(f)**: draai de volgorde van de niveaus van factor f om

8.7 Achtergrondmateriaal

- Meer informatie over `forcats` kan gelezen worden in Hoofdstuk 15 van R for Data Science

9

[Lecture notes] Tidy data

9.1 Inleiding

- In werkelijkheid komt data niet altijd in het geschikte formaat om de gewenste analyses op uit te voeren.
 - Vaak is data verspreid over meerdere datasets en moeten we hier 1 dataframe van maken voor onze analyses.
 - Soms stelt een rij niet de observatie voor die willen bestuderen (bv: één rij stelt de gegevens van één auto betrokken in een ongeval voor, terwijl we willen dat iedere rij een ongeval voorstelt met de gegevens van alle betrokken voertuigen).
- Het manipuleren van de data opdat het in het juiste formaat staat, wordt ook wel de creatie van ‘tidy data’ genoemd.
- **Bestudeer secties 12.1 tot en met 12.4 en 12.6 in ‘R for Data Science’ van Grolemund en Wickham!**
- **Bestudeer hoofdstuk 13 in ‘R for Data Science’ van Grolemund en Wickham!**

9.2 Case: NYC Vluchten 2013

9.2.1 Datasets samenvoegen

- We vertrekken van een dataset met vluchten opgestegen vanuit NYC in 2013. Hieronder een overzicht van de variabelen in de dataset.

```
glimpse(df)

## #> #> Rows: 319,809
## #> #> Columns: 13
## #> #> $ id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ...
## #> #> $ vertrekvluchthaven <chr> "EWR", "LGA", "JFK", "LGA", "EWR", "EWR", "LGA"...
## #> #> $ aankomstvluchthaven <chr> "George Bush Intercontinental", "George Bush In...
```

```

## $ maatschappij      <chr> "United Air Lines Inc.", "United Air Lines Inc...."
## $ tijdstip_aankomst <dttm> 2013-01-01 08:30:00, 2013-01-01 08:50:00, 2013...
## $ vertrek_vertraging <dbl> 2, 4, 2, -6, -4, -5, -3, -2, -2, -2, -2, -2...
## $ aankomst_vertraging <dbl> 11, 20, 33, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## $ afstand           <dbl> 1400, 1416, 1089, 762, 719, 1065, 229, 944, 733...
## $ tijdstip_vertrek   <dttm> 2013-01-01 05:17:00, 2013-01-01 05:33:00, 2013...
## $ weekdag_vertrek    <ord> Tue, Tue, Tue, Tue, Tue, Tue, Tue, Tue, Tu...
## $ week_vertrek       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ maand_vertrek      <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ maanddag_vertrek    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

```

- We beschikken nu ook over een tweede dataset met de gegevens van de luchthavens. Hieronder een overzicht van de variabelen in deze dataset.

```
glimpse(airports)
```

```

## Rows: 1,458
## Columns: 8
## $ faa    <chr> "04G", "06A", "06C", "06N", "09J", "0A9", "0G6", "0G7", "0P2"...
## $ name   <chr> "Lansdowne Airport", "Moton Field Municipal Airport", "Schaum...
## $ lat    <dbl> 41.13047, 32.46057, 41.98934, 41.43191, 31.07447, 36.37122, 4...
## $ lon    <dbl> -80.61958, -85.68003, -88.10124, -74.39156, -81.42778, -82.17...
## $ alt    <dbl> 1044, 264, 801, 523, 11, 1593, 730, 492, 1000, 108, 409, 875, ...
## $ tz     <dbl> -5, -6, -6, -5, -5, -5, -5, -8, -5, -6, -5, -5, -5, -5, -...
## $ dst    <chr> "A", "A", "A", "A", "A", "A", "U", "A", "A", "U", "...
## $ tzone  <chr> "America/New_York", "America/Chicago", "America/Chicago", "Am...

```

- Als we deze datasets vergelijken zien we een mogelijke relatie tussen beiden.
 - In de oorspronkelijke dataset stelt iedere rij een vlucht voor en wordt de vertrekvluchthaven voorgesteld door een 3-letterige code.
 - In de airports-dataset stelt iedere rij een luchthaven voor en vinden we een 3-letterige code terug in de kolom ‘faa’.
- We willen nu graag deze twee datasets aan elkaar koppelen door de gegevens van de vertrekvluchthavens uit de airports-dataset te halen en toe te voegen aan iedere vlucht.
- Alvorens we dit kunnen doen, moeten we eerst controleren of de faa-code in de airports-dataset uniek is.
 - Dit is een essentiële vereiste om de gegevens van de airports-dataset te kunnen toevoegen aan de oorspronkelijke dataset.
 - Indien er bijvoorbeeld 2 luchthavens in de airports-dataset zouden zitten met faa-code ‘EWR’, dan zou R niet kunnen achterhalen van welke luchthaven de gegevens moeten worden toegevoegd aan de vluchten met als vertrekvluchthaven ‘EWR’.

- In zulke gevallen gaat R de vlucht dupliveren en iedere kopie (van de vlucht) koppelen aan een andere luchthaven uit de airports-dataset met faa-code EWR.

```
airports %>%
  count(faa) %>%
  filter(n>1)

## # A tibble: 0 x 2
## # ... with 2 variables: faa <chr>, n <int>
```

- Uit bovenstaande analyse blijkt dat er geen twee rijen zijn in de airports-dataset met dezelfde faa-code.
- We kunnen nu de gegevens van de airports-dataset toevoegen aan het oorspronkelijk dataframe. We doen dit met behulp van een *left_join()* en geven aan via welke variabelen de link gelegd moet worden.

```
df <- df %>% left_join(airports, by=c("vertrek_luchthaven"="faa"))
glimpse(df)

## # Rows: 319,809
## # Columns: 20
## # $ id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ...
## # $ vertrek_luchthaven <chr> "EWR", "LGA", "JFK", "LGA", "EWR", "EWR", "LGA"...
## # $ aankomst_luchthaven <chr> "George Bush Intercontinental", "George Bush In...
## # $ maatschappij <chr> "United Air Lines Inc.", "United Air Lines Inc...."
## # $ tijdstip_aankomst <dttm> 2013-01-01 08:30:00, 2013-01-01 08:50:00, 2013...
## # $ vertrek_vertraging <dbl> 2, 4, 2, -6, -4, -5, -3, -3, -2, -2, -2, -2...
## # $ aankomst_vertraging <dbl> 11, 20, 33, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## # $ afstand <dbl> 1400, 1416, 1089, 762, 719, 1065, 229, 944, 733...
## # $ tijdstip_vertrek <dttm> 2013-01-01 05:17:00, 2013-01-01 05:33:00, 2013...
## # $ weekdag_vertrek <ord> Tue, Tue, Tue, Tue, Tue, Tue, Tue, Tu...
## # $ week_vertrek <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ maand_vertrek <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ maanddag_vertrek <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ name <chr> "Newark Liberty Intl", "La Guardia", "John F Ke...
## # $ lat <dbl> 40.69250, 40.77725, 40.63975, 40.77725, 40.6925...
## # $ lon <dbl> -74.16867, -73.87261, -73.77893, -73.87261, -74...
## # $ alt <dbl> 18, 22, 13, 22, 18, 18, 22, 13, 22, 13, 13, ...
## # $ tz <dbl> -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, -5, ...
## # $ dst <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A...
## # $ tzone <chr> "America/New_York", "America/New_York", "Americ...
```

- Bovenstaande output laat zien dat 7 kolommen zijn toegevoegd aan de oorspronkelijke dataset.

- Merk op dat de faa-kolom van het airports-dataframe niet is toegevoegd.
Dit is niet nodig aangezien we in de join-functie hadden aangegeven dat deze kolom overeenkwam met de kolom vertrekvluchthaven uit de oorspronkelijke dataset.
- Controleer ook altijd of het aantal observaties niet gewijzigd is, daar dit vaak wijst op een fout in de join. In dit geval is het aantal observaties niet veranderd.
- In een volgende stap verwijderen we een aantal kolommen die we verder niet nodig gaan hebben en veranderen we de kolom ‘name’ in ‘vertrekvluchthaven’. Zoals je in het resultaat kan zien bevat onze nieuwe dataset nu de volledige naam van de vertrekvluchthaven en niet enkel de faa-code.

```
df <- df %>%
  select(-vertrekvluchthaven, -lat, -lon, -alt, -tz, -dst, -tzone) %>%
  rename(vertrekvluchthaven = name)
glimpse(df)

## # Rows: 319,809
## # Columns: 13
## # $ id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ...
## # $ aankomstvluchthaven <chr> "George Bush Intercontinental", "George Bush In...
## # $ maatschappij <chr> "United Air Lines Inc.", "United Air Lines Inc.....
## # $ tijdstip_aankomst <dttm> 2013-01-01 08:30:00, 2013-01-01 08:50:00, 2013...
## # $ vertrek_vertraging <dbl> 2, 4, 2, -6, -4, -5, -3, -2, -2, -2, -2, -2...
## # $ aankomst_vertraging <dbl> 11, 20, 33, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## # $ afstand <dbl> 1400, 1416, 1089, 762, 719, 1065, 229, 944, 733...
## # $ tijdstip_vertrek <dttm> 2013-01-01 05:17:00, 2013-01-01 05:33:00, 2013...
## # $ weekdag_vertrek <ord> Tue, Tue, Tue, Tue, Tue, Tue, Tue, Tu...
## # $ week_vertrek <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ maand_vertrek <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ maanddag_vertrek <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## # $ vertrekvluchthaven <chr> "Newark Liberty Intl", "La Guardia", "John F Ke...
```

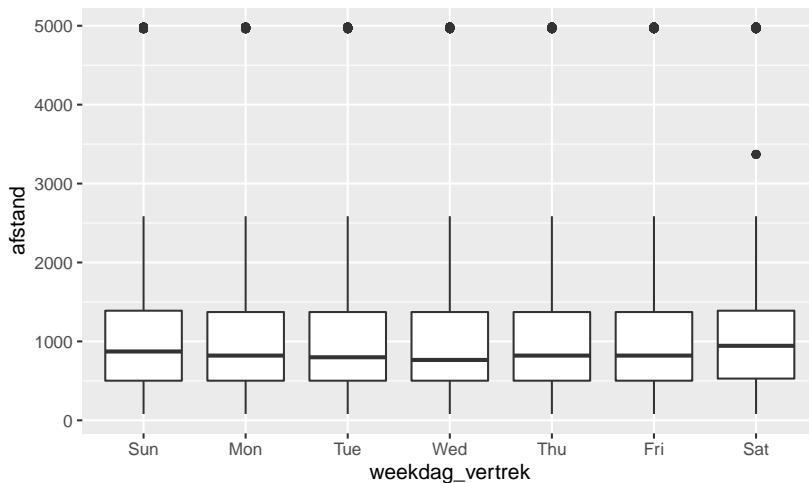
9.3 Data in een lang formaat plaatsen (voor visuele analyses)

- Bij een bivariate visualisatie heb je steeds het basisprincipe dat je de relatie tussen twee variabelen wenst weer te geven.
- Bij een multivariate visualisatie ga je vaak weergeven hoe deze relatie verandert in functie van een derde variabele.
- Deze derde variabele is vaak categorisch en de verschillende categorieën stellen hierbij groeperingen van de observaties voor waarvoor je de relatie tussen X en Y wenst weer te geven.
 - Je wil bijvoorbeeld initieel de relatie tussen weekdag en afstand van de vluchten weergeven. Hiervoor kan je een bivariate plot

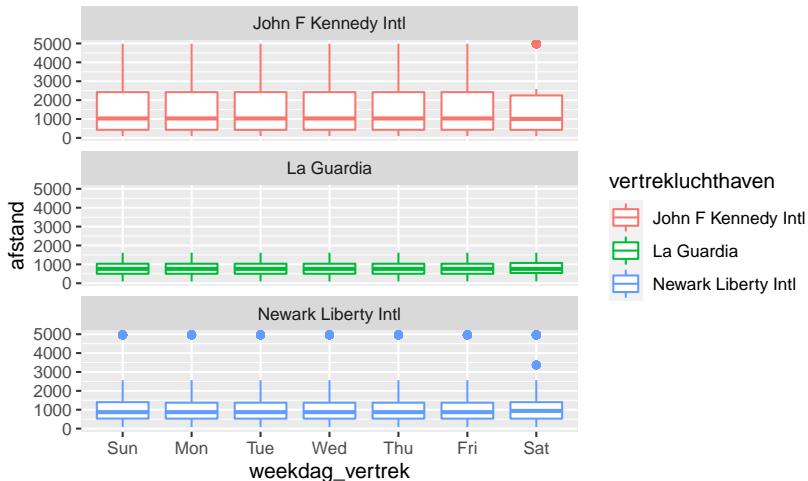
maken waarbij X categorisch is en Y continu. Een mogelijkheid hiervoor is een boxplot.

- In een volgende stap kan je de relatie tussen afstand en weekdag opsplitsen per luchthaven. Je wil dus weten hoe deze relatie verschilt tussen diverse luchthavens. Hiervoor gebruik je de categorische variabele ‘vertrekluchthaven’ en kan je bijvoorbeeld de kleur van de boxplot koppelen aan de vertrekluchthaven of aparte ‘facets’ maken voor iedere luchthaven.
- Hieronder zie je de bijhorende plots.

```
df %>%
  ggplot(aes(x=weekdag_vertrek, y=afstand)) +
  geom_boxplot()
```



```
df %>%
  ggplot(aes(x=weekdag_vertrek, y=afstand, colour=vertrekluchthaven)) +
  geom_boxplot() +
  facet_wrap(~vertrekluchthaven, ncol=1)
```



- Stel nu dat je het effect wenst te weten van de weekdag van vertrek op de vertraging van een vlucht, maar je wil hierbij onderscheid maken tussen vertrek- en aankomstvertraging.
- Volgens bovenstaande aanpak zou je dan een Y-variabele moeten hebben die de vertraging meet en een Z-variabele die het type van vertraging aangeeft (aankomst of vertrek).
- Onze dataset is echter anders opgebouwd. In de beschikbare data is de vertraging van een vlucht opgeslagen met behulp van twee aparte variabelen, namelijk vertrek- en aankomstvertraging. Dit blijkt uit onderstaande tabel.

```
df_temp <- df %>%
  select(id, vertrekvluchthaven, vertrek_vertraging, aankomst_vertraging, weekdag_vertrek)
df_temp

## # A tibble: 319,809 x 5
##       id vertrekvluchthaven  vertrek_vertraging aankomst_vertraging weekdag_vertrek
##   <int> <chr>                <dbl>            <dbl> <ord>
## 1     1 Newark Liberty Int~        2             11 Tue 
## 2     2 La Guardia              4             20 Tue 
## 3     3 John F Kennedy Int~      2             33 Tue 
## 4     4 La Guardia             -6            -25 Tue 
## 5     5 Newark Liberty Int~     -4             12 Tue 
## 6     6 Newark Liberty Int~     -5             19 Tue 
## 7     7 La Guardia             -3            -14 Tue 
## 8     8 John F Kennedy Int~     -3             -8 Tue 
## 9     9 La Guardia             -2              8 Tue 
## 10    10 John F Kennedy Int~    -2             -2 Tue 
## # ... with 319,799 more rows
```

- We moeten de data dus omzetten zodat het type vertraging niet

gecodeerd wordt als aparte variabelen, maar door middel van 1 categorische variabele.

- Hiervoor kunnen we de *gather()* functie hanteren. Deze functie zal een set van variabelen (in dit geval ‘vertrek_vertraging’ en ‘aankomst_vertraging’) transformeren naar 2 variabelen, namelijk een key-variabele en een value-variabele.
 - De key-variabele is een categorische variabele en de categorieën komen overeen met de variabelennamen in onze set van variabelen die we wensen te transformeren. In ons geval zijn dit dus de categorieën ‘vertrek_vertraging’ en ‘aankomst_vertraging’.
 - De value-variabele bevat de bijhorende waarde uit de oorspronkelijke dataset.
- De *gather()* functie bestaat uit 3 delen.
 - Eerst vermeld je alle variabelen die je wenst te vervangen.
 - Vervolgens geef je de naam van de nieuwe key-variabele.
 - Tenslotte geef je de naam van de nieuwe value-variabele.

```
df_long <- df_temp %>%
  gather(vertrek_vertraging, aankomst_vertraging, key="type_vertraging", value="vertraging") %>%
  arrange(id)
df_long

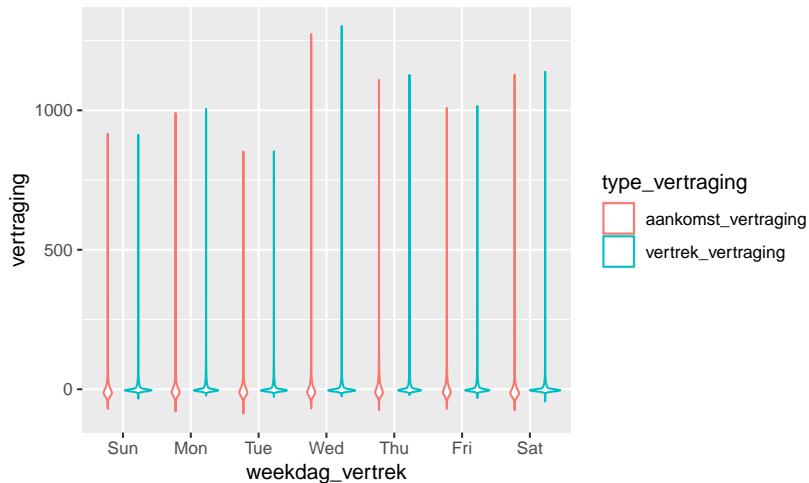
## # A tibble: 639,618 x 5
##       id vertrekluchthaven weekdag_vertrek type_vertraging    vertraging
##   <int> <chr>           <ord>          <chr>            <dbl>
## 1     1 Newark Liberty Intl Tue  vertrek_vertraging     2
## 2     1 Newark Liberty Intl Tue  aankomst_vertraging   11
## 3     2 La Guardia           Tue  vertrek_vertraging     4
## 4     2 La Guardia           Tue  aankomst_vertraging  20
## 5     3 John F Kennedy Intl Tue  vertrek_vertraging     2
## 6     3 John F Kennedy Intl Tue  aankomst_vertraging  33
## 7     4 La Guardia           Tue  vertrek_vertraging    -6
## 8     4 La Guardia           Tue  aankomst_vertraging  -25
## 9     5 Newark Liberty Intl Tue  vertrek_vertraging    -4
## 10    5 Newark Liberty Intl Tue  aankomst_vertraging   12
## # ... with 639,608 more rows
```

- Merk op dat het aantal rijen nu verdubbeld is. Dit komt omdat je nu voor zowel vertrek- als aankomstvertraging een aparte rij hebt gecreëerd.
 - Hierdoor krijg je een andere definitie van de observatie die in een rij staat. In de oorspronkelijke dataset was iedere rij (observatie) een vlucht vanuit NYC in 2013. In de nieuwe dataset stelt iedere

rij het vertrek of de aankomst van een vlucht vanuit NYC in 2013 voor!

- Indien je dus de `gather()` functie hanteert gaat het aantal rijen toenemen. Het aantal kolommen zal afnemen indien de variabelenset, die je wenst te transformeren, uit meer dan 2 variabelen bestaat.
- Hierdoor krijg je een dataset die minder breed is en vooral langer. Daarom wordt dit het lange formaat genoemd.
- Data in een lang formaat zijn voornamelijk nuttig om visualisaties te realiseren met ggplot.
- Met dit lange formaat kunnen we de relatie tussen weekdag van vertrek en de vertraging, uitgesplitst volgens vertrek- of aankomstvertraging, visualiseren.

```
df_long %>%
  ggplot(aes(x=weekdag_vertrek, y= vertraging, colour=type_vertraging)) + geom_violin()
```



- Indien we de relatie tussen we weekdag en de gemiddelde vertraging, uitgesplitst volgens vertragingstype, wensen te visualiseren, moeten we eerst de gemiddelde vertraging berekenen.

```
df_long_summary <- df_long %>%
  group_by(vertrekluchthaven, type_vertraging, weekdag_vertrek) %>%
  summarise(gem_vertraging = mean(vertraging))
df_long_summary

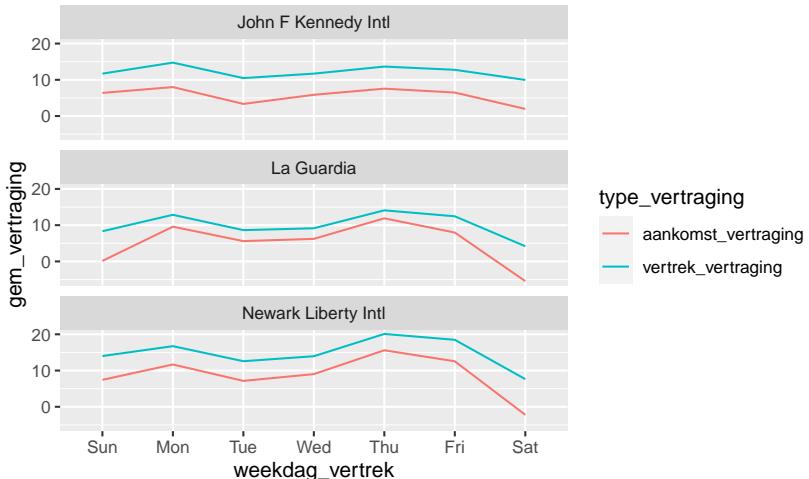
## # A tibble: 42 x 4
## # Groups:   vertrekluchthaven, type_vertraging [6]
##       vertrekluchthaven     type_vertraging    weekdag_vertrek gem_vertraging
##       <chr>                 <chr>           <ord>                  <dbl>
## 1 John F Kennedy Intl aankomst_vertraging Sun             6.39
```

```

## 2 John F Kennedy Intl aankomst_vertraging Mon 7.99
## 3 John F Kennedy Intl aankomst_vertraging Tue 3.34
## 4 John F Kennedy Intl aankomst_vertraging Wed 5.86
## 5 John F Kennedy Intl aankomst_vertraging Thu 7.56
## 6 John F Kennedy Intl aankomst_vertraging Fri 6.49
## 7 John F Kennedy Intl aankomst_vertraging Sat 1.96
## 8 John F Kennedy Intl vertrek_vertraging Sun 11.7
## 9 John F Kennedy Intl vertrek_vertraging Mon 14.7
## 10 John F Kennedy Intl vertrek_vertraging Tue 10.5
## # ... with 32 more rows

df_long_summary %>%
  ggplot(aes(x=weekdag_vertrek,
             y=gem_vertraging,
             colour=type_vertraging,
             group=interaction(type_vertraging,vertrekluchthaven))) +
  geom_line() + facet_wrap(~ vertrekluchthaven, ncol=1)

```



9.4 Data in een breed formaat plaatsen (voor overzichtelijke tabellen)

- Voor de laatste visualisatie hebben we een dataset gecreëerd met gemiddelde vertragingen per vertrekluchthaven, weekdag van vertrek en type vertraging.
- Om snel verbanden te zoeken en te evalueren is dit formaat niet erg handig. Voor zulke situaties kan je best voor een breed formaat opteren.
 - Hierbij moet je 2 variabelen selecteren: de key-variabele en de value-variabele.

Table 9.1: Gemiddelde vertraging
(lang formaat)

weekdag_vertrek	vertrekluchthaven	type_vertraging	gem_vertraging
Sun	John F Kennedy Intl	aankomst_vertraging	6.39
Sun	John F Kennedy Intl	vertrek_vertraging	11.70
Sun	La Guardia	aankomst_vertraging	0.15
Sun	La Guardia	vertrek_vertraging	8.33
Sun	Newark Liberty Intl	aankomst_vertraging	7.44
Sun	Newark Liberty Intl	vertrek_vertraging	14.01
Mon	John F Kennedy Intl	aankomst_vertraging	7.99
Mon	John F Kennedy Intl	vertrek_vertraging	14.74
Mon	La Guardia	aankomst_vertraging	9.58
Mon	La Guardia	vertrek_vertraging	12.86
Mon	Newark Liberty Intl	aankomst_vertraging	11.67
Mon	Newark Liberty Intl	vertrek_vertraging	16.73
Tue	John F Kennedy Intl	aankomst_vertraging	3.34
Tue	John F Kennedy Intl	vertrek_vertraging	10.47
Tue	La Guardia	aankomst_vertraging	5.60
Tue	La Guardia	vertrek_vertraging	8.63
Tue	Newark Liberty Intl	aankomst_vertraging	7.15
Tue	Newark Liberty Intl	vertrek_vertraging	12.57
Wed	John F Kennedy Intl	aankomst_vertraging	5.86
Wed	John F Kennedy Intl	vertrek_vertraging	11.71
Wed	La Guardia	aankomst_vertraging	6.23
Wed	La Guardia	vertrek_vertraging	9.15
Wed	Newark Liberty Intl	aankomst_vertraging	9.02
Wed	Newark Liberty Intl	vertrek_vertraging	13.95
Thu	John F Kennedy Intl	aankomst_vertraging	7.56
Thu	John F Kennedy Intl	vertrek_vertraging	13.65
Thu	La Guardia	aankomst_vertraging	11.89
Thu	La Guardia	vertrek_vertraging	14.10
Thu	Newark Liberty Intl	aankomst_vertraging	15.60
Thu	Newark Liberty Intl	vertrek_vertraging	20.10
Fri	John F Kennedy Intl	aankomst_vertraging	6.49
Fri	John F Kennedy Intl	vertrek_vertraging	12.76
Fri	La Guardia	aankomst_vertraging	7.97
Fri	La Guardia	vertrek_vertraging	12.45
Fri	Newark Liberty Intl	aankomst_vertraging	12.55
Fri	Newark Liberty Intl	vertrek_vertraging	18.49
Sat	John F Kennedy Intl	aankomst_vertraging	1.96
Sat	John F Kennedy Intl	vertrek_vertraging	9.97
Sat	La Guardia	aankomst_vertraging	-5.44
Sat	La Guardia	vertrek_vertraging	4.19
Sat	Newark Liberty Intl	aankomst_vertraging	-2.22
Sat	Newark Liberty Intl	vertrek_vertraging	7.63

- De key-variabele is altijd een categorische variabele en de value-variabele kan zowel categorisch als continu zijn.
- Voor ieder level van de categorische key-variabele zal er een aparte kolom aangemaakt worden.
- Je kan een dataset van lang naar breed formaat omzetten met behulp van de *spread()* functie.

```
df_long_summary %>%
  spread(key=weekdag_vertrek, value=gem_vertraging) %>%
  arrange(vertrekluchthaven, type_vertraging)
```

vertrekluchthaven	type_vertraging	Table 9.2: Gemiddelde vertraging (breed formaat)						
		Sun	Mon	Tue	Wed	Thu	Fri	Sat
John F Kennedy Intl	aankomst_vertraging	6.39	7.99	3.34	5.86	7.56	6.49	1.96
John F Kennedy Intl	vertrek_vertraging	11.70	14.74	10.47	11.71	13.65	12.76	9.97
La Guardia	aankomst_vertraging	0.15	9.58	5.60	6.23	11.89	7.97	-5.44
La Guardia	vertrek_vertraging	8.33	12.86	8.63	9.15	14.10	12.45	4.19
Newark Liberty Intl	aankomst_vertraging	7.44	11.67	7.15	9.02	15.60	12.55	-2.22
Newark Liberty Intl	vertrek_vertraging	14.01	16.73	12.57	13.95	20.10	18.49	7.63

9.5 Referenties

1. ‘R for Data Science’ van Grolemund en Wickham

10

[Tutorial] Tidy data

10.1 Voor je begint

Tijdens deze tutorial zullen we verschillende r-pakketten gebruiken. Zorg ervoor dat je ze installeert, indien nodig, en laadt.

```
library(dplyr)  
library(tidyr)  
library(stringr)
```

Onze oude vriend dplyr zal ons voorzien van enkele functies om verschillende datasets te combineren tot één. We zullen tidyr gebruiken om datasets te transformeren, en stringr om enkele manipulaties van tekstvariabelen te doen.

Deze handleiding bestaat uit twee grote delen:

1. Samenvoegen van datasets
2. Datasets transformeren

Daarna, in een extra deel, zullen we een case study als voorbeeld doornemen.

Disclaimer

Er worden veel datasets gebruikt in deze handleiding. Een loadscript is voorzien om alle datasets voor je aan te maken. Voer het script uit en je kan aan de slag/

Laten we beginnen!

10.2 Data samenvoegen

We kunnen verschillende datasets samenvoegen door **joining** of **binding**.

- We **joinen** verschillende datasets die verschillende informatie bevatten over dezelfde waarnemingen. Zo kunnen we bijvoorbeeld 1) een dataset hebben van landen met hun bevolking en 2) een dataset van landen met hun levensverwachting. Deze kunnen we *joinen*.

```
countries_population
```

```
## # A tibble: 114 x 2
##   country           pop
##   <fct>        <int>
## 1 Botswana      1639131
## 2 Greece        10706290
## 3 South Africa  43997828
## 4 Ethiopia       76511887
## 5 Zimbabwe       12311143
## 6 Yemen, Rep.    22211743
## 7 Nepal          28901790
## 8 Netherlands     16570613
## 9 United States  301139947
## 10 New Zealand    4115771
## # ... with 104 more rows
```

```
countries_lifeExp
```

```
## # A tibble: 85 x 2
##   country      lifeExp
##   <fct>        <dbl>
## 1 United States  78.2
## 2 Argentina     75.3
## 3 Korea, Dem. Rep. 67.3
## 4 Bulgaria       73.0
## 5 Chile          78.6
## 6 Croatia         75.7
## 7 Canada          80.7
## 8 Honduras        70.2
## 9 Liberia          45.7
## 10 Mexico         76.2
## # ... with 75 more rows
```

- We **binden** verschillende datasets die dezelfde informatie bevatten op verschillende waarnemingen. Zo kunnen wij bijvoorbeeld beschikken over 1) een dataset van Europese landen met hun bevolking en 2) een dataset van Afrikaanse landen met hun bevolking. We kunnen deze twee aan elkaar *binden*.¹

```
population_africa
```

¹ Er zijn eigenlijk nog meer gevallen waarin we datasets aan elkaar kunnen binden, maar daar hoeft u zich nu niet druk over te maken. Even ter herinnering: bind verschillende waarnemingen, voeg verschillende informatie samen.

```

## # A tibble: 52 x 2
##   country              pop
##   <fct>                <int>
## 1 Algeria            33333216
## 2 Angola             12420476
## 3 Benin              8078314
## 4 Botswana           1639131
## 5 Burkina Faso      14326203
## 6 Burundi            8390505
## 7 Cameroon           17696293
## 8 Central African Republic 4369038
## 9 Chad               10238807
## 10 Comoros           710960
## # ... with 42 more rows

```

`population_europe`

```

## # A tibble: 30 x 2
##   country              pop
##   <fct>                <int>
## 1 Albania            3600523
## 2 Austria            8199783
## 3 Belgium            10392226
## 4 Bosnia and Herzegovina 4552198
## 5 Bulgaria           7322858
## 6 Croatia            4493312
## 7 Czech Republic     10228744
## 8 Denmark             5468120
## 9 Finland            5238460
## 10 France            61083916
## # ... with 20 more rows

```

Laten we eens kijken hoe we gegevens kunnen samenvoegen.

10.2.1 Joining data

Vergeet niet dat we datasets samenvoegen als ze verschillende informatie over dezelfde waarnemingen bevatten. Dit betekent dat er een manier moet zijn om de datasets te *linken*. Deze koppelingen noemen we *ids of keys*.

Als we bevolkings- en levensverwachtingsgegevens over landen hebben, dan is de naam, code of afkorting van het land onze key om beide datasets te koppelen.

Merk op dat, wanneer beide datasets verschillende sleutels gebruiken, bijvoorbeeld de ene gebruikt de naam (België) en de andere de code (BE), we ze niet kunnen samenvoegen. In dat geval zouden

wij een van de variabelen moeten hercoderen of een andere dataset moeten vinden die als intermediaire link kan dienen (d.w.z. een die zowel de namen als de codes bevat. Er bestaan veel verschillende landencodes, dus dit is een veel voorkomend probleem. Maar in ons geval kunnen we aan de slag)

De join functies die we zo zullen introduceren zullen altijd zoeken naar variabelen met dezelfde namen in beide tabellen en deze gebruiken als de sleutels om ze te koppelen. Je kunt de sleutels expliciet instellen met het by argument. Dit is vooral nuttig indien

- a) De sleutels in beide datasets een verschillende naam hebben. Bijvoorbeeld country vs ctry
- b) Niet alle gemeenschappelijke variabelen ook daadwerkelijk sleutels zijn.

Voorlopig zullen we de sleutels altijd laten kiezen door de functies. Een bericht zal ons vertellen welke sleutels ze hebben gebruikt.

Er zijn 4 manieren om datasets samen te voegen.

- inner_join
- left_join
- right_join
- full_join

Waarom vier? Wel, als we twee datasets willen samenvoegen, gebeurt het meestal dat ze informatie bevatten over niet *exact* dezelfde waarnemingen. Kijk maar eens naar de gegevens over de bevolking en de levensverwachting. De eerste bevat informatie over 114 landen en de tweede bevat informatie over 85 landen. Ze kunnen dus onmogelijk informatie bevatten over dezelfde set van landen. De verschillende joins zullen dit probleem verschillend aanpakken.

10.2.2 Inner join

Inner join betekent: Ik bewaar alleen informatie over keys die in beide tabellen voorkomen. Dus, als ik de bevolking van land A niet heb, wil ik ook de levensverwachting niet hebben.

```
inner_join(countries_population, countries_lifeExp)

## # A tibble: 73 x 3
##   country          pop lifeExp
##   <fct>        <int>   <dbl>
## 1 Botswana     1639131   50.7
## 2 South Africa 43997828   49.3
## 3 Ethiopia      76511887   52.9
## 4 Zimbabwe     12311143   43.5
```

```

## 5 Yemen, Rep.    22211743   62.7
## 6 Netherlands   16570613   79.8
## 7 United States 301139947  78.2
## 8 Kuwait         2505559   77.6
## 9 Colombia       44227550  72.9
## 10 Austria        8199783   79.8
## # ... with 63 more rows

```

Deze join geeft ons 73 waarnemingen, dat is de deelverzameling van landen waarover we beide soorten informatie hebben. Merk ook op hoe de `inner_join` je vertelt welke key het gebruikt.

10.2.3 Left join

Left join betekent: Ik bewaar alle informatie in mijn eerste (linker) tabel. Dus, zelfs als ik de levensverwachting niet heb, geef me nog steeds de bevolking. Het ontbrekende deel van de nieuwe waarneming (d.w.z. de levensverwachting), is nu NA.

```

left_join(countries_population, countries_lifeExp)

## # A tibble: 114 x 3
##       country          pop  lifeExp
##       <fct>      <int>   <dbl>
## 1 Botswana     1639131   50.7
## 2 Greece       10706290    NA
## 3 South Africa 43997828  49.3
## 4 Ethiopia      76511887  52.9
## 5 Zimbabwe      12311143  43.5
## 6 Yemen, Rep.   22211743  62.7
## 7 Nepal        28901790    NA
## 8 Netherlands   16570613  79.8
## 9 United States 301139947  78.2
## 10 New Zealand  4115771    NA
## # ... with 104 more rows

```

Deze join geeft ons 114 waarnemingen, dat is het aantal landen waarvoor we informatie hebben over de bevolking. Merk ook op hoe het NA's invoegt voor de lifeExp variabele.

```

left_join(countries_population, countries_lifeExp) %>%
  summary()

##       country          pop      lifeExp
##       <fct>      <int>   <dbl>
## 1 Algeria     : 1   Min.   :1.996e+05   Min.   :42.59
## 2 Angola      : 1   1st Qu.:4.120e+06   1st Qu.:59.44
## 3 Australia   : 1   Median :1.009e+07   Median :71.88

```

```

##   Austria    : 1   Mean     :4.751e+07   Mean     :67.78
##   Bahrain   : 1   3rd Qu.:2.885e+07   3rd Qu.:76.44
##   Bangladesh: 1   Max.     :1.319e+09   Max.     :82.21
##   (Other)    :108          NA's     :41

```

10.2.4 Right join

Right join betekent: het tegenovergestelde van left join. Ik bewaar alle informatie in mijn tweede (rechter) tabel.

```

right_join(countries_population, countries_lifeExp)

## # A tibble: 85 x 3
##   country           pop lifeExp
##   <fct>        <int>   <dbl>
## 1 Botswana      1639131   50.7
## 2 South Africa  43997828   49.3
## 3 Ethiopia       76511887   52.9
## 4 Zimbabwe       12311143   43.5
## 5 Yemen, Rep.   22211743   62.7
## 6 Netherlands    16570613   79.8
## 7 United States 301139947  78.2
## 8 Kuwait         2505559   77.6
## 9 Colombia       44227550   72.9
## 10 Austria        8199783   79.8
## # ... with 75 more rows

```

Deze join geeft ons 85 waarnemingen, dat is het aantal landen waarvoor we informatie hebben over de levensverwachting.

10.2.5 Full join

Full join betekent: Ik wil alle informatie behouden die ik heb. Dus ook populaties voor landen zonder levensverwachting en vice versa blijven in de dataset. Alle ontbrekende informatie wordt ingevuld als NA.

```

full_join(countries_population, countries_lifeExp)

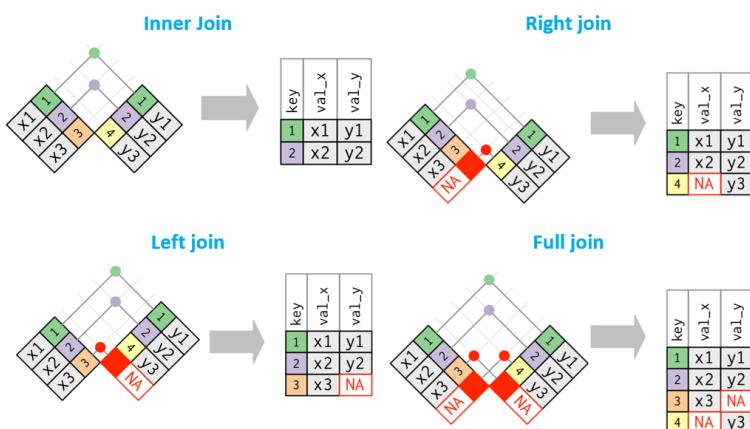
## # A tibble: 126 x 3
##   country           pop lifeExp
##   <fct>        <int>   <dbl>
## 1 Botswana      1639131   50.7
## 2 Greece         10706290    NA
## 3 South Africa  43997828   49.3
## 4 Ethiopia       76511887   52.9
## 5 Zimbabwe       12311143   43.5
## 6 Yemen, Rep.   22211743   62.7

```

```
## 7 Nepal          28901790    NA
## 8 Netherlands    16570613    79.8
## 9 United States 301139947   78.2
## 10 New Zealand   4115771     NA
## # ... with 116 more rows
```

Deze join geeft ons 126 waarnemingen, dat is het totale aantal landen waarvoor we minstens één stukje informatie hebben.

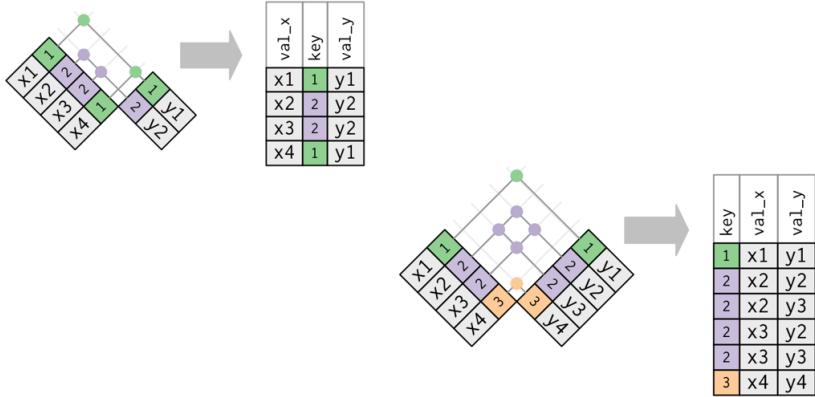
Een schematisch overzicht van de vier typen is hieronder te zien. De gekleurde getallen stellen de sleutels voor (landen in ons voorbeeld), terwijl de x- en y-waarden de waarden voorstellen (bevolking en levensverwachting in ons voorbeeld). Natuurlijk kunnen er zoveel waarden zijn als mogelijk in elke tabel, het hoeft er niet maar één te zijn. We zullen snel genoeg andere voorbeelden zien.



10.2.6 Duplicates

Soms bevatten één of beide datasets dubbele keys: stel, we hebben informatie over de bevolking in elk land voor meer dan één jaar, zodat wij voor elk land meer dan één waarneming hebben. In dergelijke gevallen zal elke waarneming meerdere malen worden samengevoegd, zoals in de onderstaande figuur.²

² Als we zowel bevolkingsgegevens over meerdere jaren als gegevens over de levensverwachting over meerdere jaren hebben, moeten we natuurlijk gewoon het *jaar* als key opnemen. We willen niet dat ze door elkaar lopen. In dat geval wordt elke waarneming gedefinieerd door zowel land als jaar.



10.2.7 Een voorbeeld

Het pakket nycflights13 bevat verschillende datasets over vluchten vanuit NYC in 2013, known from the lecture notes.

```
library(nycflights13)
```

Een van de datasets heet flights

```
flights %>%
  glimpse()

## # Rows: 336,776
## # Columns: 19
## # $ year              <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013...
## # $ month             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## # $ day               <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## # $ dep_time           <int> 517, 533, 542, 544, 554, 555, 557, 557, 558, 55...
## # $ sched_dep_time    <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 60...
## # $ dep_delay          <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, ...
## # $ arr_time           <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 8...
## # $ sched_arr_time    <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 8...
## # $ arr_delay          <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## # $ carrier            <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6"...
## # $ flight              <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301...
## # $ tailnum             <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N...
## # $ origin              <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LG...
## # $ dest                <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IA...
## # $ air_time             <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149...
## # $ distance             <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 73...
## # $ hour                 <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6...
## # $ minute                <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59...
## # $ time_hour            <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-0...
```

Een andere heet airlines, met meer informatie over de luchtvaartmaatschappijen.

```
airlines %>%
  glimpse()

## # Rows: 16
## # Columns: 2
## $ carrier <chr> "9E", "AA", "AS", "B6", "DL", "EV", "F9", "FL", "HA", "MQ",...
## $ name     <chr> "Endeavor Air Inc.", "American Airlines Inc.", "Alaska Airl...
```

Je kan zien dat ze de variabele “carrier” gemeen hebben, die voor elke luchtvaartmaatschappij een code bevat. We kunnen de naam van de luchtvaartmaatschappij dus toevoegen aan de vluchten

```
flights %>%
  inner_join(airlines)

## # A tibble: 336,776 x 20
##       year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##       <int> <int> <int>      <int>        <int>      <dbl>    <int>        <int>
## 1  2013     1     1      517          515        2.00     830        819
## 2  2013     1     1      533          529        4.00     850        830
## 3  2013     1     1      542          540        2.00     923        850
## 4  2013     1     1      544          545       -1.00    1004       1022
## 5  2013     1     1      554          600       -6.00     812        837
## 6  2013     1     1      554          558       -4.00     740        728
## 7  2013     1     1      555          600       -5.00     913        854
## 8  2013     1     1      557          600       -3.00     709        723
## 9  2013     1     1      557          600       -3.00     838        846
## 10 2013     1     1      558          600       -2.00     753        745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   name <chr>
```

Merk op dat we een inner join hebben gedaan en dat het aantal vluchten niet is afgenomen. Dit betekent dat elke luchtvaartmaatschappij in vluchten ook beschikbaar is in luchtvaartmaatschappijen. Met andere woorden, voor alle luchtvaartmaatschappijen waarvan we vluchten hebben gezien, kennen we de naam van de luchtvaartmaatschappij.

Voor een meer geavanceerd voorbeeld, laten we eens kijken naar de dataset weather.

```
weather %>%
  glimpse
```

```
## Rows: 26,115
## Columns: 15
## $ origin      <chr> "EWR", "EWR", "EWR", "EWR", "EWR", "EWR", ...
## $ year        <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 20...
## $ month       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ hour        <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 1...
## $ temp         <dbl> 39.02, 39.02, 39.02, 39.92, 39.02, 37.94, 39.02, 39.92, ...
## $ dewp         <dbl> 26.06, 26.96, 28.04, 28.04, 28.04, 28.04, 28.04, 28.04, ...
## $ humid        <dbl> 59.37, 61.63, 64.43, 62.21, 64.43, 67.21, 64.43, 62.21, ...
## $ wind_dir     <dbl> 270, 250, 240, 250, 260, 240, 240, 250, 260, 260, 260, 3...
## $ wind_speed   <dbl> 10.35702, 8.05546, 11.50780, 12.65858, 12.65858, 11.5078...
## $ wind_gust    <dbl> NA, ...
## $ precip       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ pressure     <dbl> 1012.0, 1012.3, 1012.5, 1012.2, 1011.9, 1012.4, 1012.2, ...
## $ visib        <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, ...
## $ time_hour    <dttm> 2013-01-01 01:00:00, 2013-01-01 02:00:00, 2013-01-01 03...
```

Het bevat informatie over plaats en tijd: dezelfde die we ook hebben voor vluchten, en het bevat verschillende variabelen over het weer (wind, temperatuur, neerslag, enz.).

Laten we de vluchtgegevens samenvoegen met het weer.

```
flights %>%
  inner_join(airlines) %>%
  inner_join(weather) -> flights

flights %>%
  glimpse

## Rows: 335,220
## Columns: 29
## $ year        <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013...
## $ month       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ day         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ dep_time    <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 60...
## $ dep_delay   <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -2, -2, -2, -2, ...
## $ arr_time    <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 8...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 8...
## $ arr_delay   <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, ...
## $ carrier     <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6"...
## $ flight       <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301...
## $ tailnum     <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N...
## $ origin       <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LG...
```

```

## $ dest <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IA...
## $ air_time <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149...
## $ distance <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 73...
## $ hour <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6...
## $ minute <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59...
## $ time_hour <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-0...
## $ name <chr> "United Air Lines Inc.", "United Air Lines Inc.", "A...
## $ temp <dbl> 39.02, 39.92, 39.02, 39.02, 39.92, 39.02, 37.94, 39...
## $ dewp <dbl> 28.04, 24.98, 26.96, 26.96, 24.98, 28.04, 28.04, 24...
## $ humid <dbl> 64.43, 54.81, 61.63, 61.63, 54.81, 64.43, 67.21, 54...
## $ wind_dir <dbl> 260, 250, 260, 260, 260, 240, 260, 260, 260, 26...
## $ wind_speed <dbl> 12.65858, 14.96014, 14.96014, 14.96014, 16.11092, 12...
## $ wind_gust <dbl> NA, 21.86482, NA, NA, 23.01560, NA, NA, 23.01560, NA...
## $ precip <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ pressure <dbl> 1011.9, 1011.4, 1012.1, 1012.1, 1011.7, 1011.9, 1012...
## $ visib <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, ...

```

Merk op dat de tweede join de variabelen jaar, maand, oorsprong, uur en time_hour gebruikte om het weer van de juiste plaats (vertrekplaats, meer specifiek) en tijd bij elke vlucht te voegen.

10.2.8 Binding data

De gegevens die we hierboven hebben samengevoegd waren steeds verschillende stukken informatie die we op de een of andere manier aan elkaar hebben gekoppeld (zelfde land, zelfde tijd, zelfde plaats,zelfde luchtvaartmaatschappij, enz.) Soms hebben we datasets over afzonderlijke objecten die niet aan elkaar gekoppeld zijn, maar wel dezelfde informatie bevatten. Denk aan de datasets over Afrikaanse en Europese landen.

```

population_africa

## # A tibble: 52 x 2
##   country           pop
##   <fct>          <int>
## 1 Algeria      33333216
## 2 Angola       12420476
## 3 Benin        8078314
## 4 Botswana     1639131
## 5 Burkina Faso 14326203
## 6 Burundi      8390505
## 7 Cameroon     17696293
## 8 Central African Republic 4369038
## 9 Chad         10238807
## 10 Comoros      710960

```

```
## # ... with 42 more rows

population_europe

## # A tibble: 30 x 2
##   country           pop
##   <fct>            <int>
## 1 Albania          3600523
## 2 Austria          8199783
## 3 Belgium          10392226
## 4 Bosnia and Herzegovina 4552198
## 5 Bulgaria         7322858
## 6 Croatia          4493312
## 7 Czech Republic  10228744
## 8 Denmark          5468120
## 9 Finland          5238460
## 10 France          61083916
## # ... with 20 more rows
```

Deze waarnemingen zijn niet aan elkaar gekoppeld (er is geen verband tussen een Afrikaans land en een Europees land), maar ze bevatten wel dezelfde gegevens (nl. bevolking).

We kunnen deze **rijen** aan elkaar **binden**.

```
bind_rows(population_africa, population_europe)

## # A tibble: 82 x 2
##   country           pop
##   <fct>            <int>
## 1 Algeria          33333216
## 2 Angola           12420476
## 3 Benin            8078314
## 4 Botswana         1639131
## 5 Burkina Faso    14326203
## 6 Burundi          8390505
## 7 Cameroon         17696293
## 8 Central African Republic 4369038
## 9 Chad              10238807
## 10 Comoros         710960
## # ... with 72 more rows
```

Merk op dat we 52 Afrikaanse landen hadden en 30 Europese landen. Samen maakt dit 82 landen.

Voor bind_row is het niet noodzakelijk om precies dezelfde informatie te hebben in beide datasets. Stel dat we de levensverwachting hebben voor Afrikaanse landen, maar niet voor Europese. Beschouw de dataset information_africa.

```
information_africa

## # A tibble: 52 x 3
##   country           pop lifeExp
##   <fct>        <int>   <dbl>
## 1 Algeria      33333216   72.3
## 2 Angola       12420476   42.7
## 3 Benin        8078314    56.7
## 4 Botswana     1639131    50.7
## 5 Burkina Faso 14326203   52.3
## 6 Burundi      8390505    49.6
## 7 Cameroon     17696293   50.4
## 8 Central African Republic 4369038   44.7
## 9 Chad          10238807   50.7
## 10 Comoros      710960    65.2
## # ... with 42 more rows
```

En we binden deze twee datasets.

Wat we hadden kunnen verwachten is inderdaad gebeurd: de 30 Europese landen kregen een NA voor levensverwachting. Wees echter op je hoede: als beide datasets verschillende informatie hebben, is bind_rows misschien niet wat je zoekt, en heb je misschien een join nodig? Zorg ervoor dat je begrijpt hoe je datasets zich tot elkaar verhouden en hoe je ze moet combineren.

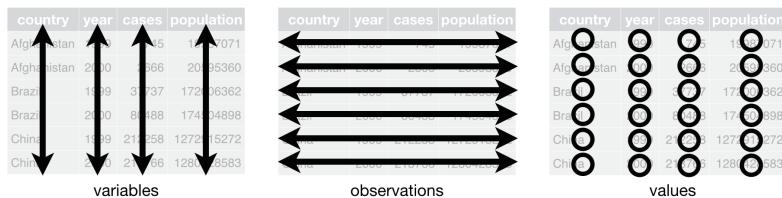
Dat gezegd zijnde, nog een opmerking over het samenvoegen van gegevens. Als er een bind_rows is, moet er toch ook een bind_cols zijn om kolommen te binden? Ja, die is er. We zullen deze functie echter niet gebruiken (hoera!). bind_cols kan doen wat het zegt: kolommen samenbinden net zoals bind_rows rijen samenbindt. Maar kolommen samenbinden betekent dat we 2 sets informatie hebben over dezelfde waarnemingen? Dat klinkt erg alsof er een join voor nodig is, nietwaar? Inderdaad! Het belangrijkste verschil tussen bind_rows en joins is dat joins rijen zullen combineren die dezelfde key hebben.

Echter, bind_rows zal rijen combineren op positie, d.w.z. de eerste rij van dataset A zal gecombineerd worden met de eerste rij van dataset B. Er wordt niet gekeken naar keys. Dus als dataset A en B in een verschillende volgorde staan, heb je je data verknood. Dus, vergeet gewoon bind_cols. Bind_rows en joins moeten in staat zijn om je te brengen waar je wilt zijn.

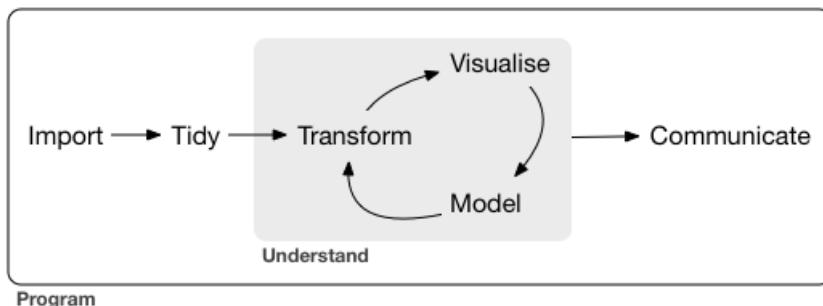
10.3 Transforming data

Naast het samenvoegen van gegevens, zullen we ook leren hoe we gegevens kunnen transformeren. Het verschil? Voor het samenvoegen hebben we twee datasets nodig, voor het transformeren zullen we slechts één dataset gebruiken.

Het belangrijkste doel van het transformeren van onze gegevens is ervoor te zorgen dat ze *tidy* zijn. Dit betekent: elke rij is een observatie, en elke kolom is een variabele.



Tidying data is vooral belangrijk in de beginfase van uw project, zoals blijkt uit de onderstaande figuur. Het kan echter ook nuttig zijn tijdens analyses. Voor sommige grafieken kan het gebeuren dat je je gegevens moet transformeren - veranderen wat je waarnemingen zijn. Dit maakt data transformatie zowel essentieel als moeilijk. Het is zeer belangrijk te begrijpen wat de huidige vorm van je gegevens is, en in welke vorm je ze nodig hebt voor je analyse. Dit vergt oefening en tijd.



We zullen vier verschillende transformaties bespreken³.

Er zijn 2 eenvoudige transformaties:

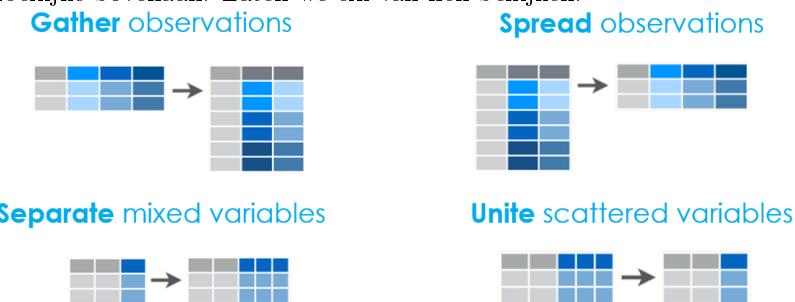
1. Combineer variabelen
2. Splits variabelen

³ Merk op dat wij de term *transformatie* voor verschillende dingen hebben gebruikt. We hebben het eerder gebruikt om *variabelen* te transformeren (factoren hercoderen, getallen herschalen, enz.). Op dit moment gebruiken we het om *data* te transformeren, wat betekent dat we het hebben over meerdere variabelen of volledige datasets. De woordkeuze is niet om je in verwarring te brengen, we doen eigenlijk hetzelfde, maar op verschillende niveaus.

en 2 moeilijke

3. Verspreid een dataset
4. Verzamel een dataset

Hieronder tonen we schematisch - de makkelijke aan onderaan, en de moeilijke bovenaan. Laten we elk van hen bekijken.⁴



⁴ Merk op dat alle functies voor samenvoeging en transformatie die hier worden besproken, zijn opgenomen het formularium. Zorg ervoor dat je het kunt gebruiken tijdens oefeningen en examen!

10.3.1 Unite variables

We gebruiken de functie `unite` wanneer we verschillende variabelen hebben die we willen combineren tot een enkele. De syntaxis voor `unite` is als volgt. Stel, we hebben informatie over studenten, met een voornaam en achternaam, en we willen één enkele “naam” variabele.

`students`

```
## # A tibble: 10 x 2
##   first_name last_name
##   <chr>      <chr>
## 1 Kemba      Raylin
## 2 Orean      Elisha
## 3 Kirstyn    Francico
## 4 Amparo     Theoplis
## 5 Belen      Ashea
## 6 Rayshaun   Angela
## 7 Brazil     Essie
## 8 Chaston   Allyn
## 9 Reyn       Tanita
## 10 Ogechi    Sherriann

students %>%
  unite(col = name, first_name, last_name)

## # A tibble: 10 x 1
##   name
##   <chr>
## 1 Kemba_Raylin
```

```
## 2 Orean_Elisha
## 3 Kirstyn_Francico
## 4 Amparo_Theoplis
## 5 Belen_Ashea
## 6 Rayshaun_Angela
## 7 Brazil_Essie
## 8 Chaston_Allyn
## 9 Reyn_Tanita
## 10 Ogechi_Sherriann
```

We specificeren eerst de naam voor de nieuwe kolom (die hier gewoon name is), daarna sommen we alle kolommen op die we willen verenigen. Merk op dat unite standaard een `_` tussen de kolommen zet. We kunnen dit veranderen met het argument sep.

```
students %>%
  unite(col = name, first_name, last_name, sep = " ")

## # A tibble: 10 x 1
##   name
##   <chr>
## 1 Kemba Raylin
## 2 Orean Elisha
## 3 Kirstyn Francico
## 4 Amparo Theoplis
## 5 Belen Ashea
## 6 Rayshaun Angela
## 7 Brazil Essie
## 8 Chaston Allyn
## 9 Reyn Tanita
## 10 Ogechi Sherriann
```

Soms geven we er ook de voorkeur aan de oorspronkelijke variabelen te behouden. We kunnen vragen ze niet te verwijderen, als volgt.

```
students %>%
  unite(col = name, first_name, last_name, sep = " ", remove = F)

## # A tibble: 10 x 3
##   name      first_name last_name
##   <chr>      <chr>     <chr>
## 1 Kemba      Kemba     Raylin
## 2 Orean      Orean     Elisha
## 3 Kirstyn    Kirstyn   Francico
## 4 Amparo     Amparo    Theoplis
## 5 Belen      Belen     Ashea
```

```

## 6 Rayshaun Angela Rayshaun Angela
## 7 Brazil Essie Brazil Essie
## 8 Chaston Allyn Chaston Allyn
## 9 Reyn Tanita Reyn Tanita
## 10 Ogechi Sherriann Ogechi Sherriann

```

10.3.2 Separate variables

Scheiden werkt andersom: het scheidt een enkele variabele in meerdere variabelen. Stel dat we een lijst hebben van studenten (`students2`) met hun volledige namen, en we willen ze scheiden.⁵

`students_2`

```

## # A tibble: 10 x 1
##       name
##   <chr>
## 1 Vashawn Heathr
## 2 Hans Musab
## 3 Shihab Mahogany
## 4 Daden Braun
## 5 Shiloh Billi
## 6 Hashir Magdalena
## 7 Latangela Lois
## 8 Lydon Aliha
## 9 Garcelle Ziah
## 10 Jaleal Nancy

```

We kunnen `separate` op een vergelijkbare manier gebruiken met `unit`. Vertel eerst welke kolom je gescheiden wilt hebben. Zeg dan in welke kolommen je de stukken wil zetten.⁶

```

students_2 %>%
  separate(col = name, into = c("first_name", "last_name"))

## # A tibble: 10 x 2
##       first_name last_name
##   <chr>      <chr>
## 1 Vashawn    Heathr
## 2 Hans        Musab
## 3 Shihab      Mahogany
## 4 Daden       Braun
## 5 Shiloh      Billi
## 6 Hashir      Magdalena
## 7 Latangela   Lois
## 8 Lydon       Aliha

```

⁵ Let op hoe je `separate` spelt. Een e, gevolgd door een a, nog een a, en nog een e. Kun je dat onthouden? Gefeliciteerd, je hebt zojuist een aantal veelgemaakte fouten vermeden!

⁶ Merk op dat het argument `col` in `unit` de nieuwe kolom is, terwijl het argument `col` in `separate` de bestaande kolom! Merk ook op dat de nieuwe kolommen gecreëerd door `separate` moeten worden gegeven als een character vector, niet als een lijst van unquoted namen zoals we deden in `unit`.

```
## 9 Garcelle   Ziah
## 10 Jaleal    Nancy
```

Standaard zal separate de kolommen splitsen op elk teken dat niet alfanumeriek is: alles behalve cijfers en letters. Dus, hij heeft correct spaties gebruikt, waarmee we perfect tevreden zijn. Als je dit wilt veranderen, kan je het sep argument opnieuw instellen. Wanneer er bijvoorbeeld een gecombineerde achternaam is zoals Janssen-Swilden, dan zou die gesplitst worden op het - teken. Dat willen we niet, dus we moeten tegen sep zeggen dat het alleen op spaties moet splitsen, dus sep = " ".

Separate zal precies zoveel kolommen maken als het aantal namen dat je opgeeft in into. Als hij meer of minder stukken vindt dan dat aantal voor een waarneming, zal hij je daarvoor waarschuwen. Als er minder zijn, zal NA verschijnen, als er meer zijn, zullen de laatste worden weggegooid. Je kunt ook remove = F gebruiken om de originele variabelen te behouden.

Tot zover unite en separate. Laten we ons nu eens concentreren op die moeilijke!

10.3.3 Spread data

We kunnen spread gebruiken om een paar variabelen - een *key* en een *value* - te nemen en ze over verschillende kolommen te verdelen: één voor elke *key* met de overeenkomstige *value* erin.



Als je het op dit moment hoort donderen in Keulen, dan wordt het misschien tijd om eerdere tutorials te herzien. Want spread hebben we eigenlijk al eerder gezien (Hebben we dat?) (Ja dat hebben we.)

Het volgende voorbeeld frist de boel misschien een beetje op.

```
library(ggplot2)
diamonds %>%
  count(color, clarity)

## # A tibble: 56 x 3
##       color clarity     n
## * <fct>   <fct>   <dbl>
## 1 D      SI2     1.00
## 2 E      SI1     1.00
## 3 F      VS1     1.00
## 4 G      VS2     1.00
## 5 H      SI2     1.00
## 6 I      SI1     1.00
## 7 J      VS1     1.00
## 8 K      VS2     1.00
## 9 L      SI2     1.00
## 10 M     SI1     1.00
## # ... with 46 more rows, and 1 more variable: clarity <fct>
```

```

##      <ord> <ord>   <int>
## 1 D     I1       42
## 2 D     SI2      1370
## 3 D     SI1      2083
## 4 D     VS2      1697
## 5 D     VS1       705
## 6 D     VVS2      553
## 7 D     VVS1      252
## 8 D     IF        73
## 9 E     I1       102
## 10 E    SI2      1713
## # ... with 46 more rows

diamonds %>%
  count(color, clarity) %>%
  spread(clarity, n)

## # A tibble: 7 x 9
##   color   I1   SI2   SI1   VS2   VS1   VVS2   VVS1   IF
##   <ord> <int> <int> <int> <int> <int> <int> <int>
## 1 D       42   1370  2083  1697   705   553   252   73
## 2 E      102   1713  2426  2470  1281   991   656   158
## 3 F      143   1609  2131  2201  1364   975   734   385
## 4 G      150   1548  1976  2347  2148  1443   999   681
## 5 H      162   1563  2275  1643  1169   608   585   299
## 6 I       92   912   1424  1169   962   365   355   143
## 7 J       50   479   750   731   542   131    74    51

```

Wanneer we gegevens *spreiden*, gaan we van een *lange* dataset naar een *brede* dataset. Kijk maar terug naar het voorbeeld en de schematische figuur. Zorg ervoor dat je dit onthoudt!

10.3.4 Gather data

Als we spread al kenden, is gather een fluitje van een cent. Het doet het tegenovergestelde van spread. Hoe eenvoudig! Dus, met gather gaan we van een *brede* dataset naar een *lange* dataset, door verschillende waarnemingen te *verzamelen* in één enkele.

Kijk maar naar dit cijfer.



Laten we eens kijken naar een voorbeeld.

De dataset hieronder toont de bevolking voor elk land op aarde na elk interval van 5 jaar, beginnend in 1952 en eindigend in 2007.

`yearly_population`

```
## # A tibble: 142 x 14
##   country continent `1952` `1957` `1962` `1967` `1972` `1977` `1982` `1987` 
##   <fct>    <fct>     <int>   <int>   <int>   <int>   <int>   <int>   <int> 
## 1 Afghan~ Asia        8.43e6  9.24e6  1.03e7  1.15e7  1.31e7  1.49e7  1.29e7 1.39e7
## 2 Albania Europe     1.28e6  1.48e6  1.73e6  1.98e6  2.26e6  2.51e6  2.78e6  3.08e6
## 3 Algeria Africa     9.28e6  1.03e7  1.10e7  1.28e7  1.48e7  1.72e7  2.00e7  2.33e7
## 4 Angola  Africa     4.23e6  4.56e6  4.83e6  5.25e6  5.89e6  6.16e6  7.02e6  7.87e6
## 5 Argent~ Americas   1.79e7  1.96e7  2.13e7  2.29e7  2.48e7  2.70e7  2.93e7  3.16e7
## 6 Austra~ Oceania    8.69e6  9.71e6  1.08e7  1.19e7  1.32e7  1.41e7  1.52e7  1.63e7
## 7 Austria Europe     6.93e6  6.97e6  7.13e6  7.38e6  7.54e6  7.57e6  7.57e6  7.58e6
## 8 Bahrain Asia       1.20e5  1.39e5  1.72e5  2.02e5  2.31e5  2.97e5  3.78e5  4.55e5
## 9 Bangla~ Asia       4.69e7  5.14e7  5.68e7  6.28e7  7.08e7  8.04e7  9.31e7  1.04e8
## 10 Belgium Europe    8.73e6  8.99e6  9.22e6  9.56e6  9.71e6  9.82e6  9.86e6  9.87e6
## # ... with 132 more rows, and 4 more variables: `1992` <int>, `1997` <int>,
## #   `2002` <int>, `2007` <int>
```

Best een overzichtelijke tabel, nietwaar? Laten we een lijngrafiek maken van de evolutie. We zouden tijd (year) nodig hebben op de x-as en bevolking op de y-as. Maar...? Wel, f*ck! Die variabelen bestaan niet?! Hoe kan ik mijn lijngrafiek maken?

Laten we de gegevens *verzamelen* in die twee variabelen.

- Het belangrijkste argument is de **new** variabele waarin we de oude variabelen willen hebben. In ons geval willen we alle jaren als een *time* variabele, zodat we ze kunnen gebruiken, in plaats van ze te verspreiden over 12 variabelen.
- Het value-argument is de **nieuwe** variabele waarin de **waarden** van de oude variabelen terechtkomen. Dit zouden dus de bevolkingsaantallen zijn, oftewel *population*.
- Daarna geven we alle kolommen op die we willen verzamelen. In ons geval alle jaren. Gelukkig kunnen we ons typwerk besparen, en

gewoon zeggen dat we land en continent **niet** willen verzamelen.⁷

Laten we eens kijken wat er gebeurt.

```
yearly_population %>%
  gather(key = time, value = population, -country, -continent)

## # A tibble: 1,704 x 4
##   country   continent time  population
##   <fct>     <fct>    <chr>    <int>
## 1 Afghanistan Asia     1952     8425333
## 2 Albania      Europe   1952     1282697
## 3 Algeria       Africa   1952     9279525
## 4 Angola        Africa   1952     4232095
## 5 Argentina     Americas 1952     17876956
## 6 Australia     Oceania  1952     8691212
## 7 Austria       Europe   1952     6927772
## 8 Bahrain        Asia    1952     120447
## 9 Bangladesh    Asia    1952     46886859
## 10 Belgium      Europe   1952     8730405
## # ... with 1,694 more rows
```

Voila, precies het tegenovergestelde van spreiding, is het niet? Een stel oude variabelen (1952, 1957, 1962, enz.) worden *verzameld* in één nieuwe variabele *time*. Terwijl de inhoud van die oude variabelen ernaast wordt geplaatst in de *population* variabele.

Merk op hoe we van een dataset met 13 kolommen en 142 rijen (= BREED) zijn gegaan naar een dataset met slechts 3 kolommen maar 1704 rijen (= LANG).

Dus, laten we dit afronden.

- Voor verzamelen (gather, breed naar lang): key en value zijn *nieuwe* kolomnamen. Je kunt ze kiezen zoals je wilt (net zoals ik tijd en bevolking koos)
- Voor spreiden (spread, lang naar breed): key en value zijn *bestaande* kolommen. Degenen die je wilt verspreiden.
- Bij gather geef je een lijst van *bestaande* kolommen die je wilt verzamelen/combineren. Je kunt ook aangeven welke je niet wilt met -. In feite kun je hier alle select-trucs gebruiken. Als je niets zegt behalve key en value, worden alle kolommen verzameld.
- Bij spread zijn alleen key en value noodzakelijke argumenten.

Makkelijk, is het niet?

Helaas, nee. Dat is het niet.

Spread en gather zijn waarschijnlijk de minst intuïtieve functies die je in deze cursus zult leren. Probeer deze paragraaf een paar keer te

⁷ Eigenlijk is er een belangrijkere reden waarom we -land en -continent willen gebruiken in plaats van alle jaren op te sommen, behalve dat we lui zijn. Weet je nog dat alle objecten variabelennamen in R moeten beginnen met een letter, niet met een cijfer? Wel, de jaar-kolommen doen dat duidelijk niet. Om ze te selecteren zou een speciale techniek nodig zijn. Gewoon 1952:2007 zeggen zou helaas niet werken. Maar, gelukkig, dat is een verhaal voor een andere keer.

lezen, en kijk heel goed naar de voorbeelden. Probeer te zien wat er gebeurt. Dingen kunnen erg ingewikkeld worden met spread en gather, omdat ze de structuur van je gegevens volledig veranderen. Door ze te combineren met joins wordt het alleen maar moeilijker. Spendeer wat tijd aan het begrijpen van de functies, en leer hoe je de cheatsheet moet gebruiken. Nog belangrijker dan het verschil kennen tussen lange en brede datasets, is het begrijpen wanneer je elk van deze nodig hebt.

De functies zijn helemaal niet gemakkelijk, maar je zult ze sneller nodig hebben dan je denkt. Laten we ze aan het werk zien in een ander voorbeeld. We zullen wat echte gegevens gebruiken van de World Health Organisation WHO!

Oh, dat vergat ik bijna! We zouden een lijngrafiek maken van de bevolkingsgegevens. Wel, zie je, eens we gather gebruikt hebben hebben, wordt het gemakkelijk. We kunnen bijna direct naar ggplot gaan.

```
yearly_population %>%
  gather(key = time, value = population, -country, -continent) %>%
  mutate(time = as.numeric(time)) %>%
  ggplot(aes(time, population/(10^9), group = country, color = continent)) +
  geom_line() +
  facet_grid(. ~ continent) +
  theme_light() +
  labs(y = "Population (in billion)") +
  theme(legend.position = "top")
```

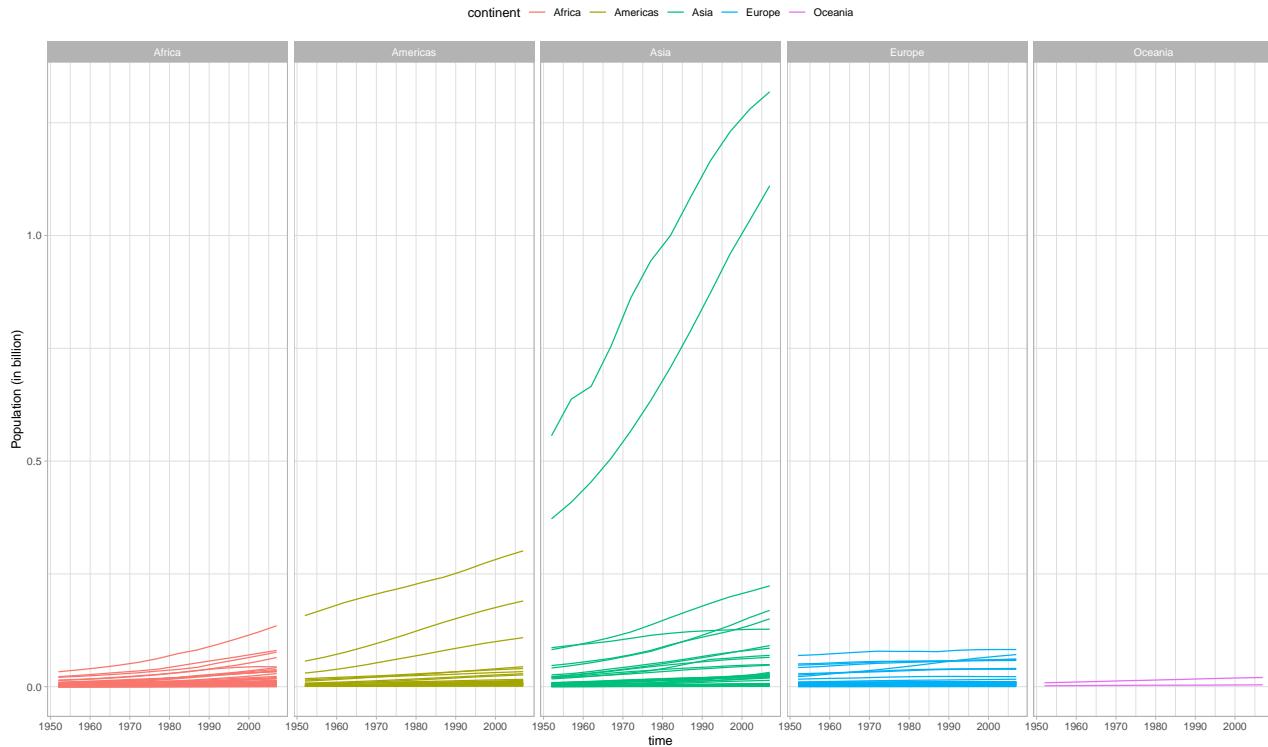
Kan je vertellen welke landen de twee stijgende lijnen in Azië zijn?
(Zeg me alsjeblieft van wel.)

Dus, laten we wat gezondheid bestuderen!

10.4 [Case study]: WHO

Wij verzamelden (pun intended) gegevens over het aantal (nieuwe) tuberculosegevallen uitgesplitst naar

- year
- country
- age (7 groepen)
- gender
- type of TB
 - new/old -> (allemaal nieuw in dit voorbeeld)
 - diagnosis method
 - * rel: relapse
 - * sp: smear positive
 - * sn: smear negative



* ep: extrapulmonary

(Je hoeft de verschillende diagnosemethoden niet te kennen.)

De gegevens zien er als volgt uit.

```
who %>%  
  glimpse()  
  
## # Rows: 7,240  
## # Columns: 60  
## # $ country      <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanis...  
## # $ iso2          <chr> "AF", "AF", "AF", "AF", "AF", "AF", "AF", "AF", ...  
## # $ iso3          <chr> "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG", "AFG"...  
## # $ year          <int> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, ...  
## # $ new_sp_m014   <int> NA, NA,...  
## # $ new_sp_m1524  <int> NA, NA,...  
## # $ new_sp_m2534  <int> NA, NA,...  
## # $ new_sp_m3544  <int> NA, NA,...  
## # $ new_sp_m4554  <int> NA, NA,...  
## # $ new_sp_m5564  <int> NA, NA,...  
## # $ new_sp_m65    <int> NA, NA,...  
## # $ new_sp_f014   <int> NA, NA,...  
## # $ new_sp_f1524  <int> NA, NA,...
```



```
## $ newrel_f4554 <int> NA, NA...
## $ newrel_f5564 <int> NA, NA...
## $ newrel_f65    <int> NA, NA...
```

Om eerlijk te zijn: nogal een puinhoop. We hebben toch niet echt 60 variabelen nodig voor de gegevens die we net beschreven? Wat is er aan de hand?

Het lijkt erop dat alle gegevens voor elk land en jaar op één rij staan. Laten we eens kijken.

```
who %>%
  count(country, year)

## # A tibble: 7,240 x 3
##   country     year     n
##   <chr>       <int> <int>
## 1 Afghanistan 1980     1
## 2 Afghanistan 1981     1
## 3 Afghanistan 1982     1
## 4 Afghanistan 1983     1
## 5 Afghanistan 1984     1
## 6 Afghanistan 1985     1
## 7 Afghanistan 1986     1
## 8 Afghanistan 1987     1
## 9 Afghanistan 1988     1
## 10 Afghanistan 1989    1
## # ... with 7,230 more rows
```

We zien vooral eentjes. Laten we het voor de zekerheid controleren.

```
who %>%
  count(country, year) %>%
  filter(n > 1)

## # A tibble: 0 x 3
## # ... with 3 variables: country <chr>, year <int>, n <int>
```

Ok. Dus, elk jaar, elk land, één rij. We hebben 7240 rijen omdat we

```
who %>% count(year) %>% nrow
```

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

34 jaren hebben, en

```
who %>% count(country) %>% nrow()

## [1] 219
```

219 landen.

Dus verwachten we het volgende aantal rijen:

219*34

[1] 7446

Het lijkt erop dat we 206 rijen missen. D.w.z. dat er landen zijn waarvoor we niet alle jaren hebben, of omgekeerd. Het is hier niet echt belangrijk, maar dit zijn het soort dingen die een goede data analist controleert.

Laten we teruggaan naar ons probleem.

Van de 60 variabelen geven de eerste 3 elk het land weer (herinner dat ik vertelde dat er verschillende manieren zijn om een land af te korten), en de 4e bevat het jaar. Er blijven dus 56 variabelen over.

Welnu, we hebben informatie over 7 leeftijdsgroepen, 2 geslachten, en 4 diagnosemethoden. 7 maal 2 maal 4 is gelijk aan 56. Aha! Alle verschillende gevallen worden in een andere variabele gestopt. Dat is niet echt gemakkelijk om mee te werken.

Waarom niet, hoorde ik je denken?

Laten we proberen de volgende vragen op te lossen.

- Hoeveel vrouwen ouder, 25 jaar of ouder in België werden gediagnosticeerd met TB in 2000? Hoeveel van hen hadden een terugval?
- Wat is het totaal aantal TB-gevallen in België in elk jaar?
- Kan ja de evolutie van het aantal gevallen voor verschillende geslachten en leeftijdsgroepen grafisch weergeven?

Nee, dat kan je niet. Tenminste, niet zonder veel werk, of zonder onze gegevens op te transformeren. Dus, laten we beginnen.

Het is vaak nuttig om na te denken over het formaat waarin we onze gegevens zouden willen hebben, zonder te verdwalen in de transformatie. Idealiter zouden we de volgende variabelen willen hebben:

- country
- year
- is_new
- diagnosis
- gender
- age
- cases (het aantal TB-gevallen)

Laten we eerst naar een dataset in een lang formaat gaan, door alle verschillende soorten diagnoses en gevallen in een lange lijst te verzamelen. We zullen de eerste 4 kolommen niet verzamelen. De oude kolomnamen zullen een variabele “type” zijn, en de getallen zullen “gevallen” worden genoemd.

```

who %>%
  gather(key = type, value = cases, -country:-year)

## # A tibble: 405,440 x 6
##   country     iso2   iso3   year type      cases
##   <chr>      <chr>  <chr>  <int> <chr>      <int>
## 1 Afghanistan AF    AFG    1980 new_sp_m014    NA
## 2 Afghanistan AF    AFG    1981 new_sp_m014    NA
## 3 Afghanistan AF    AFG    1982 new_sp_m014    NA
## 4 Afghanistan AF    AFG    1983 new_sp_m014    NA
## 5 Afghanistan AF    AFG    1984 new_sp_m014    NA
## 6 Afghanistan AF    AFG    1985 new_sp_m014    NA
## 7 Afghanistan AF    AFG    1986 new_sp_m014    NA
## 8 Afghanistan AF    AFG    1987 new_sp_m014    NA
## 9 Afghanistan AF    AFG    1988 new_sp_m014    NA
## 10 Afghanistan AF   AFG    1989 new_sp_m014   NA
## # ... with 405,430 more rows

```

Zie je wat er gebeurd is? Kijk maar eens goed.

Had je al begrepen dat we eerst de data moesten verzamelen? Zo ja, gefeliciteerd, je begint te snappen wat data-transformatie is en welke transformaties je waar nodig hebt. Zo nee, maak je geen zorgen. Onthoud dat ik je vertelde dat dit een moeilijke vaardigheid is. Bovendien zijn er waarschijnlijk verschillende manieren om dit te doen.

We kunnen ons ontdoen van iso2 en iso3. Merk op dat ze nuttig kunnen zijn om de gegevens te verbinden met andere gegevens over landen, maar we hebben geen plannen om dat te doen. Laten we ze gewoon uit de weg ruimen.

```

who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3)

## # A tibble: 405,440 x 4
##   country     year type      cases
##   <chr>      <int> <chr>      <int>
## 1 Afghanistan 1980 new_sp_m014    NA
## 2 Afghanistan 1981 new_sp_m014    NA
## 3 Afghanistan 1982 new_sp_m014    NA
## 4 Afghanistan 1983 new_sp_m014    NA
## 5 Afghanistan 1984 new_sp_m014    NA
## 6 Afghanistan 1985 new_sp_m014    NA
## 7 Afghanistan 1986 new_sp_m014    NA
## 8 Afghanistan 1987 new_sp_m014    NA
## 9 Afghanistan 1988 new_sp_m014    NA
## 10 Afghanistan 1989 new_sp_m014   NA

```

```
## # ... with 405,430 more rows
```

Nu, er is veel informatie in de `type` variabele. Eigenlijk zijn er meer variabelen in deze ene variabele. Laten we ze scheiden. (Zie je hoe dat denkproces gaat?)

Laten we eerst eens kijken naar de verschillende niveaus door een snelle telling te doen.

```
who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3) %>%
  count(type) %>%
  print(n = Inf) # I want to see all of them

## # A tibble: 56 x 2
##   type          n
##   <chr>     <int>
## 1 new_ep_f014    7240
## 2 new_ep_f1524    7240
## 3 new_ep_f2534    7240
## 4 new_ep_f3544    7240
## 5 new_ep_f4554    7240
## 6 new_ep_f5564    7240
## 7 new_ep_f65      7240
## 8 new_ep_m014    7240
## 9 new_ep_m1524    7240
## 10 new_ep_m2534    7240
## 11 new_ep_m3544    7240
## 12 new_ep_m4554    7240
## 13 new_ep_m5564    7240
## 14 new_ep_m65      7240
## 15 new_sn_f014    7240
## 16 new_sn_f1524    7240
## 17 new_sn_f2534    7240
## 18 new_sn_f3544    7240
## 19 new_sn_f4554    7240
## 20 new_sn_f5564    7240
## 21 new_sn_f65      7240
## 22 new_sn_m014    7240
## 23 new_sn_m1524    7240
## 24 new_sn_m2534    7240
## 25 new_sn_m3544    7240
## 26 new_sn_m4554    7240
## 27 new_sn_m5564    7240
## 28 new_sn_m65      7240
```

```

## 29 new_sp_f014    7240
## 30 new_sp_f1524   7240
## 31 new_sp_f2534   7240
## 32 new_sp_f3544   7240
## 33 new_sp_f4554   7240
## 34 new_sp_f5564   7240
## 35 new_sp_f65     7240
## 36 new_sp_m014    7240
## 37 new_sp_m1524   7240
## 38 new_sp_m2534   7240
## 39 new_sp_m3544   7240
## 40 new_sp_m4554   7240
## 41 new_sp_m5564   7240
## 42 new_sp_m65     7240
## 43 newrel_f014    7240
## 44 newrel_f1524   7240
## 45 newrel_f2534   7240
## 46 newrel_f3544   7240
## 47 newrel_f4554   7240
## 48 newrel_f5564   7240
## 49 newrel_f65     7240
## 50 newrel_m014    7240
## 51 newrel_m1524   7240
## 52 newrel_m2534   7240
## 53 newrel_m3544   7240
## 54 newrel_m4554   7240
## 55 newrel_m5564   7240
## 56 newrel_m65     7240

```

Oh sh*t. De eerste 42 levels zijn netjes gescheiden door 2 underscores. Maar de laatste niet. Het is overall “newrel” in plaats van “new_rel”. Separate zal niet in staat zijn om dat te splitsen...

Dus, laten we een truc uithalen. We gaan al het “newrel” teksten vervangen door “new_rel”. Hoe? Met behulp van het **stringr** pakket voor string manipulatie. Het heeft een handige functie **str_replace**. Daar gaan we.

```

who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3) %>%
  mutate(type = str_replace(type, "newrel", "new_rel")) %>%
  count(type) %>%
  print(n = Inf)

## # A tibble: 56 x 2

```

```
##      type          n
##      <chr>        <int>
## 1 new_ep_f014    7240
## 2 new_ep_f1524   7240
## 3 new_ep_f2534   7240
## 4 new_ep_f3544   7240
## 5 new_ep_f4554   7240
## 6 new_ep_f5564   7240
## 7 new_ep_f65     7240
## 8 new_ep_m014    7240
## 9 new_ep_m1524   7240
## 10 new_ep_m2534  7240
## 11 new_ep_m3544  7240
## 12 new_ep_m4554  7240
## 13 new_ep_m5564  7240
## 14 new_ep_m65    7240
## 15 new_rel_f014  7240
## 16 new_rel_f1524 7240
## 17 new_rel_f2534 7240
## 18 new_rel_f3544 7240
## 19 new_rel_f4554 7240
## 20 new_rel_f5564 7240
## 21 new_rel_f65    7240
## 22 new_rel_m014   7240
## 23 new_rel_m1524  7240
## 24 new_rel_m2534  7240
## 25 new_rel_m3544  7240
## 26 new_rel_m4554  7240
## 27 new_rel_m5564  7240
## 28 new_rel_m65    7240
## 29 new_sn_f014    7240
## 30 new_sn_f1524   7240
## 31 new_sn_f2534   7240
## 32 new_sn_f3544   7240
## 33 new_sn_f4554   7240
## 34 new_sn_f5564   7240
## 35 new_sn_f65     7240
## 36 new_sn_m014    7240
## 37 new_sn_m1524   7240
## 38 new_sn_m2534   7240
## 39 new_sn_m3544   7240
## 40 new_sn_m4554   7240
## 41 new_sn_m5564   7240
## 42 new_sn_m65     7240
```

```

## 43 new_sp_f014    7240
## 44 new_sp_f1524   7240
## 45 new_sp_f2534   7240
## 46 new_sp_f3544   7240
## 47 new_sp_f4554   7240
## 48 new_sp_f5564   7240
## 49 new_sp_f65     7240
## 50 new_sp_m014   7240
## 51 new_sp_m1524   7240
## 52 new_sp_m2534   7240
## 53 new_sp_m3544   7240
## 54 new_sp_m4554   7240
## 55 new_sp_m5564   7240
## 56 new_sp_m65     7240

```

Dat is beter, nietwaar? Trouwens, zie je hoe we op elk punt voortbouwen op wat we eerder deden? Op deze manier kunnen we gemakkelijk fouten veranderen als we er maken. Pas als onze gegevens correct zijn getransformeerd, slaan we ze op, en zetten we de code in ons load-script.

Maar nu kunnen we de gegevens scheiden. Het eerste deel wordt de `is_new` variabele, het tweede deel de diagnose variabele, en het laatste deel... wel, het bevat zowel het geslacht (f/m) als de leeftijdscategorie. Laten we het gewoon `age_gender` noemen, en dat probleem later aanpakken.

```

who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3) %>%
  mutate(type = str_replace(type, "newrel", "new_rel")) %>%
  separate(type, into = c("is_new", "diagnosis", "gender_age"))

## # A tibble: 405,440 x 6
##   country      year is_new diagnosis gender_age cases
##   <chr>       <int> <chr>   <chr>      <chr>     <int>
## 1 Afghanistan  1980  new     sp        m014      NA
## 2 Afghanistan  1981  new     sp        m014      NA
## 3 Afghanistan  1982  new     sp        m014      NA
## 4 Afghanistan  1983  new     sp        m014      NA
## 5 Afghanistan  1984  new     sp        m014      NA
## 6 Afghanistan  1985  new     sp        m014      NA
## 7 Afghanistan  1986  new     sp        m014      NA
## 8 Afghanistan  1987  new     sp        m014      NA
## 9 Afghanistan  1988  new     sp        m014      NA
## 10 Afghanistan 1989  new     sp        m014     NA
## # ... with 405,430 more rows

```

Cool, dat werkte! We hoefden niet eens te vertellen hoe we moesten splitsen. Hij besliste dit automatisch. Wat een slimmerik!

Nu, laten we `age_gender` splitsen. Maar op wat? Er is geen karakter om op te splitsen. Maar, separate is zo slim, dat we het kunnen vertellen te splitsen na het *eerste* teken - want dat is het geslacht, de rest is de leeftijd. We kunnen dit eigenlijk voor elk karakter doen. We hoeven alleen `sep = n` in te stellen, waarbij `n` ons getal is. In dit geval 1. Laten we het proberen!

```
who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3) %>%
  mutate(type = str_replace(type, "newrel", "new_rel")) %>%
  separate(type, into = c("is_new", "diagnosis", "gender_age")) %>%
  separate(gender_age, into = c("gender", "age"), sep = 1)

## # A tibble: 405,440 x 7
##   country     year is_new diagnosis gender age   cases
##   <chr>      <int> <chr>    <chr>    <chr> <chr> <int>
## 1 Afghanistan 1980 new      sp       m     014    NA
## 2 Afghanistan 1981 new      sp       m     014    NA
## 3 Afghanistan 1982 new      sp       m     014    NA
## 4 Afghanistan 1983 new      sp       m     014    NA
## 5 Afghanistan 1984 new      sp       m     014    NA
## 6 Afghanistan 1985 new      sp       m     014    NA
## 7 Afghanistan 1986 new      sp       m     014    NA
## 8 Afghanistan 1987 new      sp       m     014    NA
## 9 Afghanistan 1988 new      sp       m     014    NA
## 10 Afghanistan 1989 new     sp       m     014    NA
## # ... with 405,430 more rows
```

Ik weet niet hoe het met jou zit, maar ik denk dat dit precies is hoe we de gegevens wilden hebben! Laten we het nu opslaan.

```
who %>%
  gather(key = type, value = cases, -country:-year) %>%
  select(-iso2, -iso3) %>%
  mutate(type = str_replace(type, "newrel", "new_rel")) %>%
  separate(type, into = c("is_new", "diagnosis", "gender_age")) %>%
  separate(gender_age, into = c("gender", "age"), sep = 1) -> tidy_who
```

En laten we voor ons plezier ook de vragen oplossen die we eerder hadden.

- Bij hoeveel vrouwen van 25 jaar of ouder in België werd in 2000 tbc vastgesteld? Hoeveel van hen hadden een terugval?

```

tidy_who %>%
  filter(gender == "f", !(age %in% c("014", "1524")), country == "Belgium", year == 2000) %>%
  group_by(diagnosis) %>%
  summarize(n_cases = sum(cases, na.rm = T))

## # A tibble: 4 x 2
##   diagnosis n_cases
##   <chr>      <int>
## 1 ep          0
## 2 rel          0
## 3 sn          0
## 4 sp         78

```

Volgens deze gegevens waren er 78 gevallen, en geen daarvan waren hervallen.

- Wat is het totaal aantal TB-gevallen in België in elk jaar?

```

tidy_who %>%
  filter(country == "Belgium") %>%
  group_by(year) %>%
  summarize(n_cases = sum(cases, na.rm = T))

## # A tibble: 34 x 2
##   year n_cases
##   <int>    <int>
## 1 1980     0
## 2 1981     0
## 3 1982     0
## 4 1983     0
## 5 1984     0
## 6 1985     0
## 7 1986     0
## 8 1987     0
## 9 1988     0
## 10 1989    0
## # ... with 24 more rows

```

(Het lijkt erop dat er vóór 1995 geen gevallen van TB in België waren. Of missen we gewoon gegevens? Wat is de invloed van na.rm? Wees voorzichtig.)

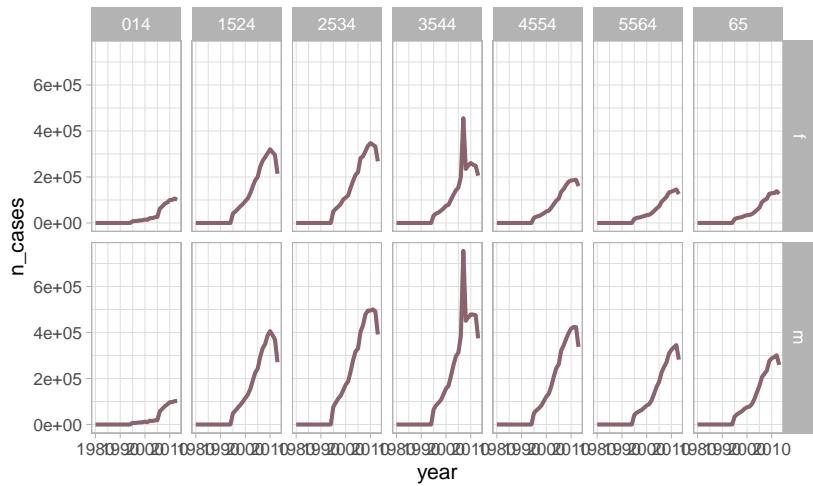
- Kan je de ontwikkeling van het aantal gevallen voor verschillende geslachten en leeftijdsgroepen grafisch weergeven?

```

tidy_who %>%
  group_by(year, age, gender) %>%

```

```
summarize(n_cases = sum(cases, na.rm = T)) %>%
ggplot(aes(year, n_cases)) +
geom_line(color = "pink4", lwd = 1) +
facet_grid(gender~age) +
theme_light()
```



11

[Lecture notes] Tijdsdata

11.1 Inleiding

11.1.1 Tijdstippen versus periodes

- We kunnen tijdgerelateerde data in twee categorieën onderverdelen:
tijdstippen en periodes.
- Tijdstip.
 - Verwijst naar een specifiek moment in de tijd.
 - 3 varianten:
 - * datum (“01-01-2017”) verwijst naar een specifieke dag.
 - * datum-tijdstip (“01-01-2017 13:54”) verwijst naar een specifiek moment op een specifieke dag.
 - * tijdstip (“13:54”) verwijst naar een specifiek moment op een ongedefinieerde dag.
- Periode.
 - Verwijst naar een periode en wordt typisch uitgedrukt aan de hand van de duur van de periode.
 - * Bijvoorbeeld: Een periode van “3605 seonden” of een periode van “2 maanden en 1 dag”.
 - Soms wordt een periode specifiek gedefinieerd aan de hand van twee specifieke tijdstippen die het begin en het einde van de periode aangeven.
 - * Bijvoorbeeld: De periode van 01-01-2017 tot 03-01-2017.
- **Bestudeer hoofdstuk 16 van het boek ‘R for Data Science’ van Grolemund en Wickham !**

11.1.2 Afronden van tijdstippen

- Ieder tijdstip heeft een zekere nauwkeurigheid. Sommige tijdstippen zijn tot op de seconde gedefinieerd terwijl andere slechts een nauwkeurigheid hebben van weken of maanden.

- Soms kan het voor visualisaties of analyses zinvol zijn om tijdstippen minder nauwkeurig te maken en deze af te ronden.

11.2 Periode-data

- We kunnen 3 soorten van periodes onderscheiden, waarbij het eerste type (interval) naar een specifieke periode tussen 2 tijdstippen verwijst en de 2 andere types (*duration* en *period*) naar een periode van een specifieke duur verwijzen maar telkens onafhankelijk van het specifieke tijdstip.

11.2.1 Interval

- Een interval is een periode die bepaald wordt door twee specifieke tijdstippen.
- Intervals worden weinig gebruikt om rechtstreeks te analyseren, maar kunnen als tussenstap gebruikt worden om de duurtijd van specifieke periodes te bepalen.

11.2.2 Duration

- Duration is de duur van een periode uitgedrukt als het exact aantal seconden die feitelijk verstrekken zijn tussen twee tijdstippen.
- Tussen ‘26 maart 2017 02:00:00’ en ‘26 maart 2017 03:00:01’ is slechts 1 seconde feitelijk verstrekken omdat we van 2u naar 3u zijn overgeschakeld op het zomertijd.
- Durations gebruik je voornamelijk als je de werkelijke tijd tussen twee tijdstippen wenst te berekenen of wanneer je een aantal seconden wenst toe te voegen bij of af te trekken van een specifiek tijdstip.

11.2.3 Period

- De tijd die verstrekken ‘lijkt’ te zijn (op een klok) tussen twee tijdstippen.
- Dus tussen ‘26 maart 2017 02:00:00’ en ‘26 maart 2017 03:00:01’ zit een period van 1 uur en 1 seconde.
- Periods gebruik je voornamelijk als je periodes wilt toevoegen aan tijdstippen zonder rekening te moeten houden met onverwachte sprongen in de tijd (zomertijd/wintertijd, schrikkeljaren, . . .).
 - Dus als je bij ieder tijdstip 1 dag (24u) wenst toe te voegen, kan je beter een period gebruiken dan een duration, omdat je anders rekening moet houden met de dag waarop we van zomer- naar winteruur gaan en omgekeerd.

11.3 Analyseren van tijdgerelateerde data

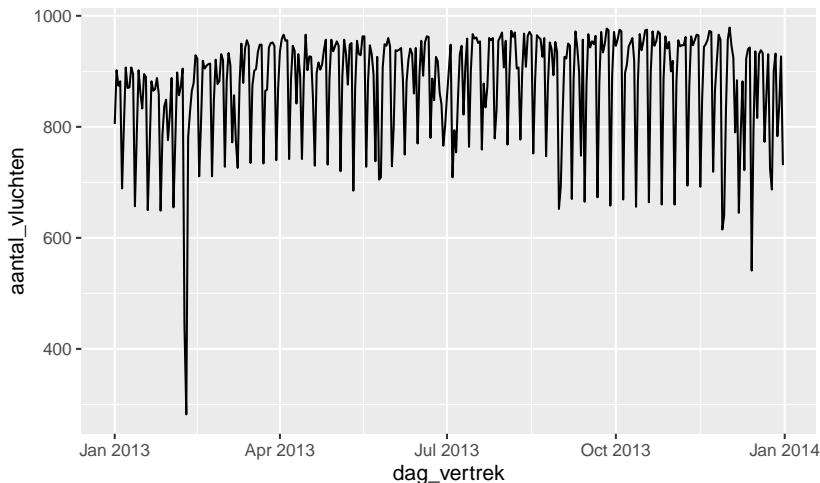
- Een eerste stap om inzicht te krijgen in de tijdgerelateerde data is met behulp van de *summary()* functie. Het is vooral nuttig om naar de minima en maxima te kijken. Dit geeft vaak aan of de tijdsperiode waarvoor de data verzameld is overeenkomt met de verwachte periode. In onderstaand geval blijkt dit in orde te zijn.

```
##  vertrek_luchthaven  aankomst_luchthaven maatschappij
##  Length:319809      Length:319809      Length:319809
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##  vertrek_gepland          vertrek_werkelijk          vertrek_vertraging
##  Min.   :2013-01-01 05:17:00  Min.   :2013-01-01 05:19:00  Min.   :-43.00
##  1st Qu.:2013-04-05 09:07:00  1st Qu.:2013-04-05 09:10:00  1st Qu.: -5.00
##  Median :2013-07-04 13:43:00  Median :2013-07-04 13:47:00  Median : -2.00
##  Mean   :2013-07-03 21:27:29  Mean   :2013-07-03 21:40:07  Mean   : 12.62
##  3rd Qu.:2013-10-01 20:37:00  3rd Qu.:2013-10-01 20:39:00  3rd Qu.: 11.00
##  Max.   :2013-12-31 23:32:00  Max.   :2014-01-01 00:19:00  Max.   :1301.00
##
##  aankomst_gepland          aankomst_werkelijk
##  Min.   :2013-01-01 07:02:00  Min.   :2013-01-01 06:55:00
##  1st Qu.:2013-04-05 11:22:00  1st Qu.:2013-04-05 11:21:00
##  Median :2013-07-04 15:42:00  Median :2013-07-04 15:36:00
##  Mean   :2013-07-03 23:42:08  Mean   :2013-07-03 23:49:08
##  3rd Qu.:2013-10-01 22:27:00  3rd Qu.:2013-10-01 22:12:00
##  Max.   :2014-01-01 01:10:00  Max.   :2014-01-01 01:53:00
##
##  aankomst_vertraging      afstand      weekdag_vertrek  week_vertrek
##  Min.   :-86.000      Min.   : 80  Sun:44396        Min.   : 1.00
##  1st Qu.:-17.000      1st Qu.: 502 Mon:48246        1st Qu.:14.00
##  Median : -5.000      Median : 828 Tue:48084        Median :27.00
##  Mean   :  6.987      Mean   :1035 Wed:47597        Mean   :26.77
##  3rd Qu.: 14.000      3rd Qu.:1372 Thu:47378        3rd Qu.:40.00
##  Max.   :1272.000     Max.   :4983 Fri:47455        Max.   :53.00
##
##                                Sat:36653
##  maand_vertrek      dag_vertrek          maanddag_vertrek
##  Min.   : 1.000      Min.   :2013-01-01 00:00:00  Min.   : 1.00
##  1st Qu.: 4.000      1st Qu.:2013-04-05 00:00:00  1st Qu.: 8.00
##  Median : 7.000      Median :2013-07-04 00:00:00  Median :16.00
##  Mean   : 6.569      Mean   :2013-07-03 07:43:47  Mean   :15.74
```

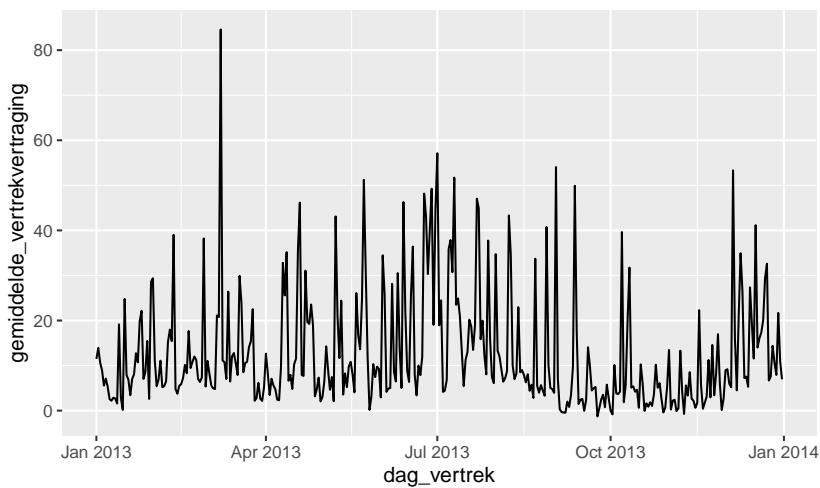
```
## 3rd Qu.:10.000 3rd Qu.:2013-10-01 00:00:00 3rd Qu.:23.00
## Max. :12.000 Max. :2013-12-31 00:00:00 Max. :31.00
##
```

Analyse visuele tijdreeks patronen

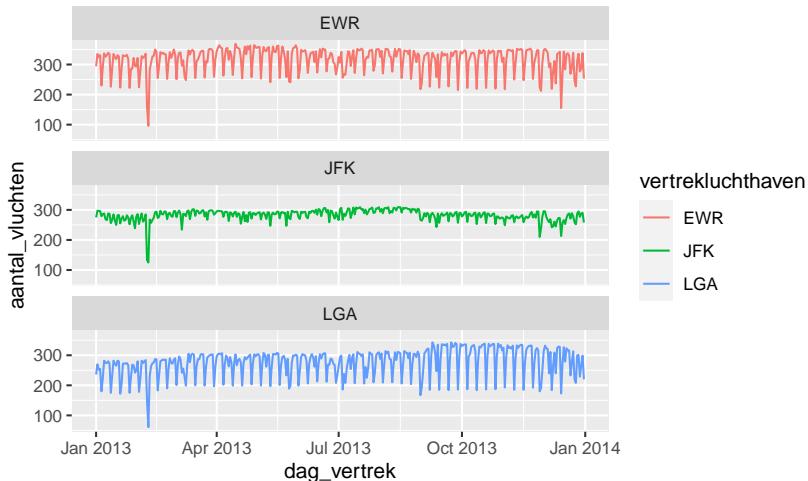
- Eén van de meest voorkomende exploratieve visuele analysetechnieken voor tijdgerelateerde data is het zoeken naar patronen hoe een variabele doorheen de tijd verandert.
- De eerste stap is hierbij telkens de tijdreeks patronen te visualiseren. Om dit te doen kan je volgend stappenplan toepassen.
 - Bepaal over welke tijdsdimensie je patronen wenst te bestuderen.
Dit is je **X**-variabele. De **X**-variabele bepaalt de granulariteit van je visualisatie. Wens je op niveau van dagen te visualiseren, dan is je tijdsdimensie ‘dag’, en dan ga je gedetailleerder naar de patronen kijken, dan wanneer je op niveau van bijvoorbeeld ‘maand’ naar de data kijkt.
 - Bepaal welke variabele je doorheen de tijd wenst te bestuderen.
Dit is je **Y**-variabele.
 - Je gaat voor iedere **X** waarde 1 **Y** waarde moeten hebben. Vaak betekent dit dat je deze **Y**-variabele nog moet aanmaken. Mogelijke **Y** variabelen zijn *het aantal observaties* per tijdsseenheid of de centrummaat (bv. mediaan) van een specifieke variabele.
 - Je R-code vertrekt steeds van de oorspronkelijke dataset, groepeert vervolgens op de tijdsdimensie, berekent de gewenste samenvattende statistiek (*summarise()*) en visualiseert vervolgens via *ggplot() + geom_line()*.
- We willen bijvoorbeeld de evolutie zien van het aantal vluchten per dag. De tijdsdimensie is dus *dag_vertrek* en de **Y**-variabele wordt gemaakt door het aantal rijen per dag te tellen.
- De analyse van onderstaande grafiek toont een aantal opvallende zaken:
 - Er is een zware en niet-wederkerende daling tussen januari en april. Hier moet iets uitzonderlijks gebeurd zijn.
 - We zien een terugkerend patroon, waarbij om de aantal dagen een daling is in het aantal vluchten.
 - De schommelingen en met name de daling op het einde van ieder terugkerend patroon wordt groter op het einde van het jaar.



- We kunnen een soortgelijke analyse doen voor de gemiddelde vertrekvertraging.



- Een volgende stap is vaak om de tijdreeks patronen apart te visualiseren voor de verschillende waarden van een categorische variabele.
- Dit kan op eenvoudige wijze door in onze R-code deze categorische variabele op te nemen in het `group_by()` gedeelte en vervolgens aparte plots te creëren met behulp van `facet_wrap()`.
- Laten we de evolutie van het aantal vluchten per dag bijvoorbeeld uitsplitsen per luchthaven.
- Uit onderstaande analyse blijkt dan dat het aantal vluchten vanuit JFK veel minder sterk schommelt dan EWR en LGA. Wel valt op dat alle drie de luchthavens een sterke uitzonderlijke daling kenden in de eerste helft van het jaar.



Identificeren van opmerkelijke gebeurtenissen in een tijdreeks

- In de evolutie van het aantal vluchten valt op dat er een uitzonderlijke daling plaatsvond in de periode tussen januari en april.
- In zulke gevallen is het best te achterhalen wat hier precies de oorzaak is.
- De eerste stap is dan ook het exacte tijdstip te identificeren.
- We kunnen dit doen door de data te filteren op die dagen dat er zeer weinig vluchten zijn.

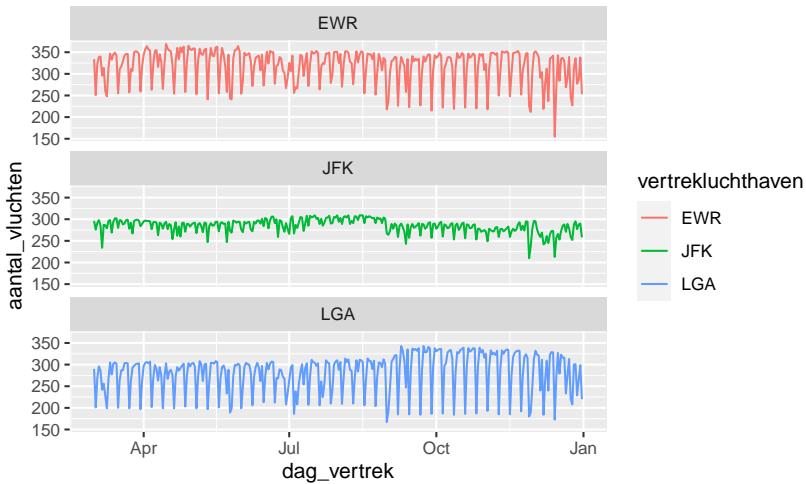
```
## # A tibble: 7 x 3
## # Groups:   dag_vertrek [3]
##   dag_vertrek     vertrekvluchthaven aantal_vluchten
##   <dttm>           <chr>                  <int>
## 1 2013-02-08 00:00:00 EWR                   159
## 2 2013-02-08 00:00:00 JFK                   133
## 3 2013-02-08 00:00:00 LGA                   148
## 4 2013-02-09 00:00:00 EWR                   96
## 5 2013-02-09 00:00:00 JFK                  125
## 6 2013-02-09 00:00:00 LGA                   61
## 7 2013-12-14 00:00:00 EWR                  155
```

- Uit deze analyse blijkt dat de daling plaatsvond op 8 en 9 februari 2013. Na enig opzoekwerk blijkt dat New York toen geteisterd werd door een hevige sneeuwstorm waardoor zeer veel vluchten geannuleerd moesten worden.
- Omdat dit moment niet representatief is voor een normaal jaar, beslissen we om enkel met de tijdgerelateerde data van maart tot en met december verder te gaan.

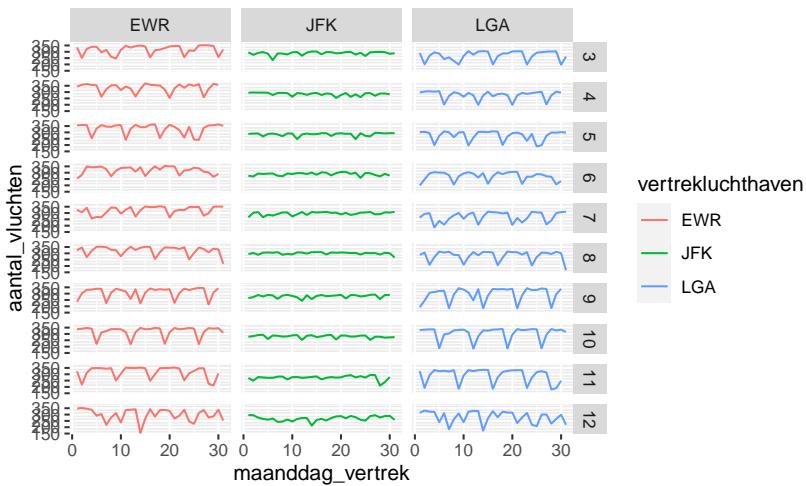
```
df %>%
  filter(vertrek_gepland >= ymd(130301)) -> df_mardec
```

- We kunnen de tijdreeks van de nieuwe periode opnieuw visualiseren.

```
df_mardec %>%
  group_by(dag_vertrek, vertrekvluchthaven) %>%
  summarise(aantal_vluchten=n()) %>%
  ggplot(aes(x = dag_vertrek, y= aantal_vluchten, colour=vertrekvluchthaven)) +
  geom_line() + facet_wrap(~ vertrekvluchthaven, ncol = 1)
```



- We kunnen verder inzoomen in de data door naar de tijdreekspatronen te kijken per maand en per luchthaven.



11.4 Referenties

1. ‘R for Data Science’ van Grolemund en Wickham

12

[Tutorial] Tijdsdata

12.1 Voor we beginnen

```
library(dplyr)  
library(lubridate)
```

Gefeliciteerd met het bereiken van de laatste tutorial van deze cursus. In deze tutorial gaan we spelen met datums en tijdstippen. Om dit op een gemakkelijke manier te doen, zullen we het pakket `lubridate` gebruiken, dat je misschien moet installeren. We zullen ook `dplyr` laden voor wanneer we tijd zullen gebruiken in een `data.frame` context. We hebben geen dataset nodig, we zullen alle data zelf maken (Hoe cool is dat?).

12.2 Introduction

In de huidige *tijden* zijn tijdsgegevens zeer belangrijk, en het is dus van cruciaal belang dat je weet hoe ermee moet werken. Als je erover nadenkt, zijn er zo veel soorten gegevens die een tijdscomponent hebben. Kijk maar naar de verschillende datasets die we hebben gebruikt.

- ordertgegevens (elke order wordt op een bepaald moment aangemaakt, verpakt, verzonden enz.)
- terrorismegegevens (elke terroristische aanslag vond op een bepaald tijdstip plaats, helaas)
- gapminder-gegevens (we hadden informatie over verschillende jaren)
- nydeaths (sterfgevallen gebeuren op een specifiek moment)
- vluchten vertrekken en komen aan op een specifiek tijdstip (en hebben soms vertraging)
- concerten (concerten vinden plaats op een specifiek moment, kaartjes worden gekocht op een specifiek moment)

Tijd is overal!

In essentie gedragen tijd en datums zich als een continue variabele. Je kunt twee datums vergelijken en zien welke datum later was. Je kunt het tijdsverschil tussen twee datums berekenen. De meeste dingen die werken voor getallen zullen op een bepaalde manier ook werken voor tijden of datums.

Goed, continue gegevens. Dus waar is al die ophef over?

Wel, tijd is een beetje complexer. Om te beginnen heeft het verschillende componenten. Laten we beginnen met het bekijken van enkele datums.

12.3 What's in a date?

Makkelijke vraag: dagen, maand, en jaar.

Het lastige is natuurlijk de volgorde. Sommigen schrijven eerst het jaar, anderen de dag. Sommigen zullen 1998 schrijven, anderen 98. Er is echt geen volgorde te bedenken die je niet kunt gebruiken. En dan zijn er nog verschillende scheidingstekens zoals spaties, -, ., /. Laten we eens kijken naar deze volkomen willekeurige datum, die we in verschillende formaten hebben genoteerd.

- 1991-06-28
- 28 06 1991
- 28 June 91
- June 28, 1991
- 280691
- 1991/06/28

De mogelijkheden zijn eindeloos. Echter, dankzij lubridate, hoeven we niet veel moeite te doen. We hoeven alleen maar de volgorde van de componenten Dag, Maand en Jaar te kennen.

Beschouw de volgende code. We maken een character vectpr dat de bovenstaande datum bevat.

```
date <- "1991-06-28"
```

Laten we zeggen dat ik het verschil wil weten tussen die dag en vandaag. Het volgende zal natuurlijk niet werken.

```
## "2021-03-21" - "1991-06-28"
```

Ik weet niet hoe het met jou zit, maar ik heb geleerd dat we geen twee woorden van elkaar kunnen aftrekken. En als we de "'s weghalen?

```
2019-04-29 - 1991-06-28
```

```
## [1] -39
```

Tenzij we in de tijd kunnen reizen, gebeurde 28-06-1991, bijna 28 jaar voor 29-04-2019, niet 39 jaar de andere kant op. Het verschil is ook geen 39 dagen of 39 maanden. Ik neem aan dat je ziet wat hier aan de hand is.

Wat we nodig hebben is een nieuw type object. Geen karakter, geen numeriek, geen geheel getal, geen factor. We hebben een datum nodig!

We kunnen een datum maken met de toepasselijk genaamde ‘make_date’ functie in lubridate, waar we jaar, maand en datum als argument kunnen geven.

```
date <- make_date(year = 1991, month = 6, day = 28)

date

## [1] "1991-06-28"
```

Het printen doet niet echt iets speciaal vergeleken met een character. Laten we eens kijken naar de klasse.¹

```
class(date)

## [1] "Date"
```

Dat lijkt in orde.

Laten we een reeks datums maken - laten we zeggen elke dag in april. Herinner je je de code onderaan nog? We zagen het om data.frames te maken een lange tijd geleden. We gebruiken as_tibble om het data.frame te transformeren naar het makkelijker te printen tibble formaat.

```
dates_of_april <- data.frame(year = 2019,
                               month = 4,
                               day = 1:30) %>%
  as_tibble()
dates_of_april

## # A tibble: 30 x 3
##       year   month   day
##   <dbl>   <dbl>   <int>
## 1 2019     4      1
## 2 2019     4      2
## 3 2019     4      3
## 4 2019     4      4
## 5 2019     4      5
## 6 2019     4      6
## 7 2019     4      7
## 8 2019     4      8
```

¹ De woorden datum en data zijn te verschillende dingen. Het is weer een veel voorkomende bron van typefouten.

```
##   9  2019      4    9
## 10  2019      4   10
## # ... with 20 more rows
```

Met mutate kunnen we nu de datum toevoegen als een aparte variabele.

```
dates_of_april %>%
  mutate(date = make_date(year, month, day)) -> dates_of_april
dates_of_april

## # A tibble: 30 x 4
##       year   month   day   date
##     <dbl>   <dbl>   <int> <date>
## 1   2019     4       1 2019-04-01
## 2   2019     4       2 2019-04-02
## 3   2019     4       3 2019-04-03
## 4   2019     4       4 2019-04-04
## 5   2019     4       5 2019-04-05
## 6   2019     4       6 2019-04-06
## 7   2019     4       7 2019-04-07
## 8   2019     4       8 2019-04-08
## 9   2019     4       9 2019-04-09
## 10  2019     4      10 2019-04-10
## # ... with 20 more rows
```

Kijk nog eens naar het type van deze variabele. Het heet een Datum.

Dit lost natuurlijk slechts een deel van het probleem op. Hoe gaan we van iets als "28-06-1991" naar 28 en 6 en 1991, en dan naar een datum? Stel dat we een tabel met datums hebben waar elke datum is opgeslagen als een karakter, zoals hieronder.²

dates_of_april_text

```
## # A tibble: 30 x 1
```

² Je kan deze tabel zelf maken door de jaar, maand, dag kolom van dates_of_april te verenigen met tidyR's unite.

```

##      date
##      <chr>
## 1 2019-4-1
## 2 2019-4-2
## 3 2019-4-3
## 4 2019-4-4
## 5 2019-4-5
## 6 2019-4-6
## 7 2019-4-7
## 8 2019-4-8
## 9 2019-4-9
## 10 2019-4-10
## # ... with 20 more rows

```

Kijk goed, deze datum kolom is geen “datum”, het is een character.

```

dates_of_april_text %>%
  glimpse()

## #> #> Rows: 30
## #> #> Columns: 1
## #> #> $ date <chr> "2019-4-1", "2019-4-2", "2019-4-3", "2019-4-4", "2019-4-5", "2...

```

Met behulp van separate zouden we deze tekens echter kunnen scheiden, en dan make_date toepassen.

```

library(tidyr)
dates_of_april_text %>%
  separate(date, into = c("year", "month", "day")) %>%
  mutate(date = make_date(year, month, day))

## #> #> # A tibble: 30 x 4
## #>   year  month  day    date
## #>   <chr> <chr> <chr> <date>
## #> 1 2019   4     1    2019-04-01
## #> 2 2019   4     2    2019-04-02
## #> 3 2019   4     3    2019-04-03
## #> 4 2019   4     4    2019-04-04
## #> 5 2019   4     5    2019-04-05
## #> 6 2019   4     6    2019-04-06
## #> 7 2019   4     7    2019-04-07
## #> 8 2019   4     8    2019-04-08
## #> 9 2019   4     9    2019-04-09
## #> 10 2019  4    10   2019-04-10
## #> # ... with 20 more rows

```

Dat is natuurlijk veel werk, wat we niet leuk vinden. We kunnen echter lubridate gebruiken. Het kan niet alleen data maken van zijn componenten, het kan deze componenten ook detecteren in een character! We hoeven het alleen de volgorde van de componenten te vertellen.

Aangezien het jaar eerst wordt opgeschreven, dan de maand, dan de dag, is de volgorde van de componenten *ymd*. Wel, er is een functie in lubridate met die naam. Laten we het proberen met onze eerdere datum.

```
date <- ymd("1991-06-28")
date
```

```
## [1] "1991-06-28"
```

Laten we een tweede datum maken voor vandaag.³

```
## date2 <- ymd("2021-03-21")
```

```
date2
```

```
## [1] "2021-03-21"
```

```
class(date2)
```

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

Er is eigenlijk een heel handige functie in lubridate om de datum van vandaag te krijgen. Het is gewoon om `today()`

```
today()
```

```
## [1] "2021-03-21"
```

Als de datumcomponenten in een andere volgorde staan, hoef je alleen maar de volgorde van de letters y, m en d te veranderen. De volgende functies worden allemaal geleverd door lubridate.

- dmy
- dym
- mdy
- myd
- ymd
- ydm

Hier zijn enkele voorbeelden.

```
ymd("2017-03-25")
```

```
## [1] "2017-03-25"
```

³ Dit is natuurlijk niet de dag waarop je de tutorial leest, maar de laatste dag dat de tutorial werd bijgewerkt.

```
dmy("25-03-2017")
## [1] "2017-03-25"

mdy("03-25-2017")
## [1] "2017-03-25"

ymd("17/03/25")
## [1] "2017-03-25"

ymd("20170325")
## [1] "2017-03-25"

ymd("170325")
## [1] "2017-03-25"
```

Merk op dat lubridate automatisch de componenten zal vinden zolang het weet in welke volgorde het moet zoeken - je hoeft je geen zorgen te maken over hoe het is gescheiden, of maand en dagen zijn geschreven met of zonder een nul (dus 1/4/2019 of 01/04/2019), of dat jaren zijn geschreven met 2 of 4 getallen. Lubridate is zelfs zo slim dat het zal proberen de juiste eeuw te raden. Stel we definiëren de datum die we eerder zagen, 28 juni 1991, maar geven lubridate alleen jaar 91.

```
ymd("910628")
```

```
## [1] "1991-06-28"
```

Het wordt 91 in 1991. Maar als we het jaar 19 geven (d.w.z. op dezelfde dag in 2019), wordt het 19 in 2019.

```
ymd("190628")
```

```
## [1] "2019-06-28"
```

Dit is natuurlijk erg handig (tenzij we het jaar 1919 zoeken). Echter, na de Y2K bug, zullen we meestal yyyy-datum vindens.⁴ Dus, laten we teruggaan naar onze tabel.

```
dates_of_april_text

## # A tibble: 30 x 1
##       date
##   <chr>
## 1 2019-4-1
## 2 2019-4-2
```

⁴ Fun fact for geeks: lubridate zal proberen een datum te vinden binnen 50 jaar vanaf nu. Dus, stel dat het jaartal 2019 is, alle nummers van 69 tot 99 zullen worden beschouwd als 1969 tot 1999, alle andere nummers zullen worden beschouwd als jaren in de 21e eeuw.

```
##  3 2019-4-3
##  4 2019-4-4
##  5 2019-4-5
##  6 2019-4-6
##  7 2019-4-7
##  8 2019-4-8
##  9 2019-4-9
## 10 2019-4-10
## # ... with 20 more rows
```

Gelukkig werkt ymd met vectoren, en ook met singuliere data. Dus, in plaats van dit

```
dates_of_april_text %>%
  separate(date, into = c("year", "month", "day")) %>%
  mutate(date = make_date(year, month, day))
```

kunnen we nu gewoon dit doen

```
dates_of_april_text %>%
  mutate(date = ymd(date))

## # A tibble: 30 x 1
##       date
##   <date>
## 1 2019-04-01
## 2 2019-04-02
## 3 2019-04-03
## 4 2019-04-04
## 5 2019-04-05
## 6 2019-04-06
## 7 2019-04-07
## 8 2019-04-08
## 9 2019-04-09
## 10 2019-04-10
## # ... with 20 more rows
```

wat natuurlijk een stuk makkelijker is.

Natuurlijk zijn er altijd momenten dat lubridate fouten kan maken. Denk aan de grappige eerste dag van april. In sommige delen van de wereld schrijven ze datums in een maand-dag-jaar volgorde, vooral in de Verenigde Staten.

Zo wordt de eerste dag van april geschreven als 04-01-2019. Als we deze dag zonder enige context zouden zien, zouden we ten onrechte denken dat 4 de dag is, 1 de maand en 2019 het jaar. Aldus,

```
dmy("04-01-2019")
```

```
## [1] "2019-01-04"
```

Lubridate zal geen problemen ondervinden omdat dit geen onjuiste datum is. Maar, we hebben het niet meer over de eerste april. We hebben het over 4 januari. De tweede april wordt 4 februari met deze code. Lubridate zal merken dat er iets fout gaat als we bij de 13e april komen.

```
dmy("04-13-2019")
```

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

Nu zal lubridate dit niet zien als een correcte dmy-date, en een NA teruggeven. Het is echter goed om te weten dat lubridate niet erg goed leert van zijn fouten. Stel dat we weer een hele dataset hebben met datums in dit formaat.⁵

```
dates_of_april_us
```

```
## # A tibble: 30 x 1
##   date
##   <chr>
## 1 4-1-2019
## 2 4-2-2019
## 3 4-3-2019
## 4 4-4-2019
## 5 4-5-2019
## 6 4-6-2019
## 7 4-7-2019
## 8 4-8-2019
## 9 4-9-2019
## 10 4-10-2019
## # ... with 20 more rows
```

Als we proberen is om te zetten in een datum met behulp van de verkeerde volgorde, dmy. Krijgen we het volgende.

```
## # A tibble: 30 x 1
##   date
##   <date>
## 1 2019-01-04
## 2 2019-02-04
## 3 2019-03-04
## 4 2019-04-04
## 5 2019-05-04
## 6 2019-06-04
## 7 2019-07-04
```

⁵ Je kunt deze tabel zelf maken door de kolommen data_van_april tabel te verenigen in de volgorde maand, dag, jaar.

```

##  8 2019-08-04
##  9 2019-09-04
## 10 2019-10-04
## 11 2019-11-04
## 12 2019-12-04
## 13 NA
## 14 NA
## 15 NA
## 16 NA
## 17 NA
## 18 NA
## 19 NA
## 20 NA
## 21 NA
## 22 NA
## 23 NA
## 24 NA
## 25 NA
## 26 NA
## 27 NA
## 28 NA
## 29 NA
## 30 NA

dates_of_april_us %>%
  mutate(date = dmy(date))

```

Merk op dat alle data tot de 13e (foutief) zijn geconverteerd. De overige zijn niet geconverteerd en vervangen door NA, waarvoor lubridate ons waarschuwt. Hij is echter niet zo slim om onze fout te herstellen en ook de eerste 12 dagen te vervangen door NA.

Laten we teruggaan naar de datums die we eerder hebben gedefinieerd.

```

date

## [1] "1991-06-28"

date2

## [1] "2021-03-21"

```

We kunnen nu met deze data rekenen.

```

date2 - date

## Time difference of 10859 days

```

Probleem opgelost. Laten we wat nieuwe problemen toevoegen.

12.4 There's always time

In plaats van datums kunnen we ook tijden hebben. In deze tutorial zullen we alleen de combinatie van datum en tijd bekijken, die we een *datetime* noemen (We gaan het niet hebben over *tijdstip* op zichzelf, wat weer een heel ander verhaal is). We kunnen een voorbeeld bekijken met de functie `now`. Terwijl `today` ons de datum van vandaag geeft, geeft `now` ons de datumtijd van nu.

```
current_time <- now()
current_time

## [1] "2021-03-21 20:52:54 CET"

class(current_time)

## [1] "POSIXct" "POSIXt"
```

De klasse van een datetime is niet langer een “date”, het is een “POSIXct” en “POSIXt”.⁶

Net als bij datums kunnen we datetimes op twee manieren maken.

1. Door `make_datetime` te gebruiken, en alle componenten (dag, maand, jaar, uren, minuten, seconden en *tijdzone*) mee te geven.
2. Met `ymd_hms`, en een karakter vector van *tijdstippen*.

Naast `ymd_hms` is er ook `dmy_hms`, `mdy_hms`, enz. Echter, uren, minuten en seconden moeten altijd in dezelfde volgorde staan. Er is geen `ymd_smh` bijvoorbeeld. Maar, we zullen ook nooit een datetime in dat formaat vinden, dus geen zorgen. Er *is* wel een `ymd_hm` en een `ymd_h` (en dus ook voor alle `ymd-orders`), die we kunnen gebruiken als we alleen het uur hebben, of het uur en de minuten, maar niet de meer gedetailleerde delen.

Je zult meestal `ymd_hms` gebruiken. Laten we enkele voorbeelden bekijken. Merk op dat de derde notatie, met een T tussen het datum- en *tijddeel*, heel gebruikelijk is in datasets, en lubridate zorgt daar zonder problemen voor.

```
dmy_hms("25-03-2017 10:30:00")

## [1] "2017-03-25 10:30:00 UTC"

dmy_hms("25032017103000")

## [1] "2017-03-25 10:30:00 UTC"

dmy_hms("25-03-2017T10:30:00")
```

⁶ Wat betekent dat in godsnaam? POSIXct is het aantal seconden sinds het begin van de eerste januari in 1970. R gebruikt dit aantal seconden om een *tijdstip* op te slaan. Een andere manier om het op te slaan is door de seconden, minuten, uren, enz. apart op te slaan. Dan is het een POSIXlt. Het verschil tussen die twee zal voor de gebruiker niet veranderen, alleen hoe de datum wordt opgeslagen. POSIXt is een bovenliggende klasse die er voor zorgt dat de meeste functies met beide soorten opslag kunnen werken. Het is niet nodig om deze namen te onthouden. Weet gewoon dat alles wat begint met POSI een datetime is, niet alleen een datum.

```
## [1] "2017-03-25 10:30:00 UTC"

ymd_hms("2017-03-25 10.30.00")

## [1] "2017-03-25 10:30:00 UTC"

dmy_hms("25-03-2017 10h30m00s")

## [1] "2017-03-25 10:30:00 UTC"

dmy_hm("25-03-2017 10:30")

## [1] "2017-03-25 10:30:00 UTC"
```

Nogmaals, je kunt dit gebruiken op volledige kolommen met behulp van mutate. Zorg er ook voor dat je controleert wat de werkelijke volgorde van componenten is. We zagen al dat lubridate ook fouten kan maken.

Als we eenmaal datetimes hebben, kunnen we er weer berekeningen mee doen.

```
datetime <- dmy_hms("25-03-2017 10h30m00s")
now() - datetime

## Time difference of 1457.391 days
```

Of ze vergelijken.

```
now() < datetime

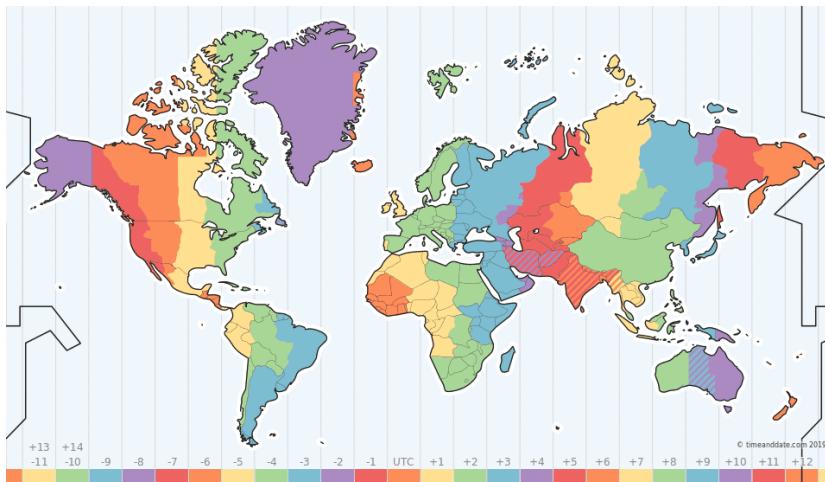
## [1] FALSE
```

We zullen nog veel meer leren wat we met data kunnen doen, maar er is eerst nog een horde te nemen.

12.5 *My time is not your time*

Dus, het hanteren van tijd is niet veel moeilijker dan het hanteren van datums. Voeg gewoon een paar extra letters toe aan de ymd functies. Dit is vooral waar als je op dezelfde plaats blijft. Zodra je je naar verschillende landen begeeft, wordt tijd moeilijker. Even het Noordzee kanaal oversteken en we moeten onze klokken verzetten.

Hieronder ziet u een kaart met alle verschillende tijdzones.



Wanneer je afstudeert zullen jullie zeker succesvolle internationale zakenleiders worden, dus een basiskennis van tijdzones is noodzakelijk. Laten we nog eens kijken naar de huidige tijd.

```
now()
```

```
## [1] "2021-03-21 20:52:55 CET"
```

Het laatste deel van deze tijd vertelt ons in welke tijdzone we ons bevinden. Op mijn pc staat er momenteel CET, dat is de Centraal Europese Wintertijd⁷. Je kunt ook controleren in welke tijdzone je zit met Sys.timezone().

```
Sys.timezone()
```

```
## [1] "Europe/Paris"
```

Op dit moment zegt mijn pc me dat het Europa/Parijs is in plaats van CET. Inderdaad, er zijn meerdere namen voor dezelfde zone. Het is allemaal deel van een groot masterplan om je in de war te brengen.

De functie olson_time_zones geeft je een lijst van alle tijdzones volgens de Olson tijd database. Dit is notatie van tijdzones met behulp van continent/stad. Het is genoemd naar Arthur David Olson.

```
olson_time_zones()
```

Probeer dit maar eens in je console. De lijst is iets te lang om hier te tonen.

Wanneer we een datum creëren met een ymd_hms-variant, zal deze automatisch UTC gebruiken als tijdzone.

```
ymd_hms("2017-03-25 10.30.00")
```

```
## [1] "2017-03-25 10:30:00 UTC"
```

⁷ Merk op dat als je deze tutorial leest, de tijdzones die je hier ziet uiteraard anders kunnen zijn t.o.v. de huidige tijdzone wanneer je het leest.

UTC is de Universele Gecoördineerde Tijd, d.w.z. de standaard, basislijn tijdzone. CET is UTC + 1, d.w.z. één uur voor op UTC. We kunnen dat veranderen door het tz argument in te stellen in de ymd_hms functie. Laten we eerst een meer betekenisvolle datetime nemen.

```
ymd_hms("2017-01-20 17:30:00")
```

```
## [1] "2017-01-20 17:30:00 UTC"
```

Op het moment gebeurde er een trieste gebeurtenis in Washington DC. Dus laten we de tijdzone op de juiste manier definiëren.

```
ymd_hms("2017-01-20 11:30:00", tz = "EST")
```

```
## [1] "2017-01-20 11:30:00 EST"
```

Merk op dat EST de Eastern Standard tijd is, de tijdzone die in de winter in DC wordt gebruikt. U kunt zien dat de tijd niet echt verandert, alleen de aanduiding van de tijdzone.

We kunnen nu zien hoe laat het was in België, door deze tijd in CET (Centraal Europese Tijd, onze wintertijd dus) weer te geven.

Eerst slaan we de tijd op als “doomsday”

```
doomsday <- ymd_hms("2017-01-20 11:30:00", tz = "EST")
```

Met behulp van with_tz kunnen we een punt in de tijd in een andere tijdzone laten zien.

```
with_tz(doomsday, "CET")
```

```
## [1] "2017-01-20 17:30:00 CET"
```

Dus, om 11u30 in Washington, was het 17u30 hier in België toen we de angstaanjagende gebeurtenissen live op onze tv-schermen zagen gebeuren.

Maar, beide verwijzen naar hetzelfde punt in de tijd. Het zijn gewoon verschillende voorstellingen van datzelfde punt.

```
doomsday_in_belgium <- with_tz(doomsday, "CET")
```

```
doomsday_in_belgium - doomsday
```

```
## Time difference of 0 secs
```

Als we echt de tijdzone willen veranderen, bv. verwijzen naar 11.30 in België, kunnen we in plaats daarvan force_tz gebruiken.

```
force_tz(doomsday, "CET")
```

```
## [1] "2017-01-20 11:30:00 CET"
```

Op dit punt hebben we het niet meer over hetzelfde tijdstip, maar over het tijdstip 6 uur voor doomsday.

```
force_tz(doomsday, "CET") - doomsday
```

```
## Time difference of -6 hours
```

We komen zo terug op de tijdzones, want zelfs in België verzetten we onze klokken af en toe. Laten we eerst eens kijken naar de dingen die we kunnen doen met datums en tijden.

12.6 Extract information from dates

Wanneer we een datum (of datetime) hebben, kunnen we er gemakkelijk informatie uit halen. Bijvoorbeeld, neem doomsday.

We kunnen zien welke dag het was.

```
day(doomsday)
```

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

of op welk uur

```
hour(doomsday)
```

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

of zelfs welke dag van de week

```
wday(doomsday)
```

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

Het was de 6e dag van de week. Maar als je zich doomsday net zo goed herinnert als ik, staat toch op je oogleden geprint dat het een verschrikkelijke vrijdag was?

Inderdaad, lubridate begint te tellen op zondag - een andere Amerikaanse eigenaardigheid. Dus, zondag is de eerste dag van de week. Zaterdag de zevende. En vrijdag de zesde.

Gelukkig heeft de wday functie een argument label, dat we op TRUE kunnen zetten als we de exacte betekenis vergeten zijn.

```
wday(doomsday, label = TRUE)
```

```
## [1] Fri
```

```
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

Vrijdag was het. Niet dat het automatisch een geordende factor is, wat wel handig is. Merk ook op dat het misschien niet in het Engels is op uw pc. Tijd is voor ieder van ons een beetje anders, afhankelijk van je PC instellingen.

We kunnen ook de maand van doomsday vragen als een getal en een label.

```
month(doomsday)

## [1] 1

month(doomsday, label = T)

## [1] Jan
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

Of natuurlijk, de eerste maand van het jaar is januari. Zelfs in de VS. (Maar misschien niet in China, maar lubridate werkt enkel met de Gregoriaanse kalender).

Met behulp van deze functies kunnen we tijdgegevens vanuit zeer verschillende invalshoeken bekijken. We kunnen maanden, dagen van de week, uren, enz. vergelijken. Het laat zien hoe complex tijd is, maar ook op hoeveel verschillende manieren we het kunnen gebruiken om iets te analyseren. Daarom is het zo belangrijk om het te kennen.

12.7 Time differences

Er zijn drie verschillende manieren om een tijdsverschil uit te drukken.

- Period
- Duration
- Interval

Een periode wordt uitgedrukt in componenten: uren, dagen, maanden, jaren. Het is hoe we tijd *waarnemen*

Een duration wordt uitgedrukt in een aantal seconden. Het is zoals een machine of klok de tijd telt.

Een interval is een *exact* tijdsinterval, met een specifieke begindatum (tijd) en een specifieke einddatum (tijd). Het is iets totaal anders dan perioden en durations. Laten we die eerst bekijken.

12.7.1 Period

We kunnen een periode creëren met de period functie, door het een aantal componenten te geven.

```
period(days = 2, minutes = 4, seconds = 70)
```

```
## [1] "2d 0H 4M 70S"
```

Merk op dat het ons gewoon de periode zal geven in gemakkelijk te begrijpen stukken tijd. 2 uur, 4 minuten en 70 seconden. Het zal het zelfs niet veranderen in 2 uur, 5 minuten en 10 seconden, want dat is niet wat we zeiden. Periodes hebben een menselijke kant.

Als onze tijd gemakkelijk kan worden uitgedrukt in een enkele component, kunnen we de componenten (in meervoud) gebruiken.

```
days(3)
```

```
## [1] "3d 0H 0M 0S"
```

```
weeks(3)
```

```
## [1] "21d 0H 0M 0S"
```

```
months(2)
```

```
## [1] "2m 0d 0H 0M 0S"
```

```
years(4)
```

```
## [1] "4y 0m 0d 0H 0M 0S"
```

We moeten het meervoud gebruiken (dagen, niet dag), want we hebben dag al gebruikt om de dag uit een datum te halen (weet je nog?). Wanneer we weken gebruiken, zal het elke week vertalen naar 7 dagen. Wanneer we maanden of jaren gebruiken, zal het extra componenten in zijn uitvoer creëren.

We kunnen deze functies ook bij elkaar optellen.

```
days(1) + hours(1)
```

```
## [1] "1d 1H 0M 0S"
```

We kunnen ook negatieve periodes maken.

```
days(-10)
```

```
## [1] "-10d 0H 0M 0S"
```

Of vectoren (reeksen) van periodes

```
period(days = 1:5, hours = 3:7)
```

```
## [1] "1d 3H 0M 0S" "2d 4H 0M 0S" "3d 5H 0M 0S" "4d 6H 0M 0S" "5d 7H 0M 0S"
```

```
days(0:6)
```

```
## [1] "OS"          "1d 0H 0M OS" "2d 0H 0M OS" "3d 0H 0M OS" "4d 0H 0M OS"
## [6] "5d 0H 0M OS" "6d 0H 0M OS"
```

En we kunnen periodes optellen bij een datum. Dus als het vandaag

```
today()
```

```
## [1] "2021-03-21"
```

is, dan zijn dit de volgende 6 datums

```
today() + days(1:6)
```

```
## [1] "2021-03-22" "2021-03-23" "2021-03-24" "2021-03-25" "2021-03-26"
## [6] "2021-03-27"
```

12.7.2 Durations

In plaats van deze voor mensen begrijpelijke componenten te gebruiken, vertalen durations alle tijdsverschillen naar seconden. Net als `period`, kunnen we `duration` gebruiken.

```
duration(days = 2, minutes = 4, seconds = 70)
```

```
## [1] "173110s (~2 days)"
```

Zoals u ziet, zal het altijd proberen een benadering te geven van de tijdsverschillen in begrijpelijke termen voor ons, mensen.

Inderdaad, we kunnen ook de wiskunde doen

```
2*24*3600+4*60+70
```

```
## [1] 173110
```

Op dezelfde manier kunnen we `ddays`, `dweek`, `dyears`, enz. gebruiken.⁸

```
ddays(3)
```

```
## [1] "259200s (~3 days)"
```

```
dweeks(3)
```

```
## [1] "1814400s (~3 weeks)"
```

```
dyears(4)
```

```
## [1] "126230400s (~4 years)"
```

We kunnen ze opnieuw combineren.

⁸ Uiteraard heeft dit niets te maken met D-Day. Het is gewoon dat day en days al genomen waren, en d staat voor duration.

```
ddays(1) + dhours(1)

## [1] "90000s (~1.04 days)"
```

We kunnen negatieve durations maken.

```
ddays(-10)

## [1] "-864000s (~-1.43 weeks)"
```

We kunnen opnieuw vectoren (reeksen) van durations maken.

```
duration(days = 1:5, hours = 3:7)

## [1] "97200s (~1.12 days)" "187200s (~2.17 days)" "277200s (~3.21 days)"
## [4] "367200s (~4.25 days)" "457200s (~5.29 days)"
```

```
ddays(0:6)

## [1] "0s"                  "86400s (~1 days)" "172800s (~2 days)"
## [4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"
## [7] "518400s (~6 days)"
```

En we kunnen rekenen met durations. Als het vandaag

```
today()

## [1] "2021-03-21"
```

is, zijn de volgende 6 dagen:

```
today() + ddays(1:6)

## [1] "2021-03-22" "2021-03-23" "2021-03-24" "2021-03-25" "2021-03-26"
## [6] "2021-03-27"
```

Merk op dat we dmonths niet kunnen berekenen!

```
dmonths(1)

## [1] "2629800s (~4.35 weeks)"
```

Er is geen vaste lengte voor maanden in aantal seonden, natuurlijk.⁹

Nu vraag je je af, waarom hebben we zowel perioden als durations? Wel, er zijn enkele belangrijke verschillen.

⁹ Je vraagt je misschien af dat er ook geen vaste lengte voor jaren is, afhankelijk van of we een schrikkeljaar hebben of niet? Dat is waar, maar dyears zullen impliciet de gangbare conventies gebruiken dat een jaar 365 dagen heeft, wat meer dan 75% van de tijd waar is. Voor maanden hebben we zo'n conventie niet.

12.7.3 Periods are not durations

Zoals ik reeds heb gezegd, zijn durations door machines interpreteerbare seconden, en zijn perioden door mensen interpreteerbare periodes.

Als wij, mensen, zeggen “volgende maand” op 3 februari, bedoelen we 3 maart. Een machine zou niet weten wat een maand is.

Laten we een paar voorbeelden bekijken om het duidelijk te maken.

Laten we de eerste dag van 2019 nemen, wat geen schrikkeljaar is.

```
ymd("2019-01-01")
```

```
## [1] "2019-01-01"
```

Als we een periode van een jaar toevoegen.

```
ymd("2019-01-01") + years(1)
```

```
## [1] "2020-01-01"
```

of een duration van een jaar

```
ymd("2019-01-01") + dyears(1)
```

```
## [1] "2020-01-01 06:00:00 UTC"
```

zijn beide hetzelfde, omdat de lengte van 2019 (d.w.z. 365 dagen), zoals wij die waarnemen, precies de duur van een jaar is.

Als we echter 2 jaar optellen, is er een verschil, want 2020 is een schrikkeljaar en hoe wij dat ervaren (366 dagen) is niet hetzelfde als de duur van een standaardjaar.

```
ymd("2019-01-01")
```

```
## [1] "2019-01-01"
```

Als we een periode van 2 jaar toevoegen.

```
ymd("2019-01-01") + years(2)
```

```
## [1] "2021-01-01"
```

Als we een duration van 2 jaar toevoegen.

```
ymd("2019-01-01") + dyears(2)
```

```
## [1] "2020-12-31 12:00:00 UTC"
```

“Volgend jaar” op Nieuwjaar 2020 betekent voor ons mensen Nieuwjaar 2021, ook al is het een schrikkeljaar. Voor durations betekent het oudjaar, omdat wordt aangenomen dat een jaar 365 dagen heeft.

Laten we een ander voorbeeld nemen. Op 31 maart 2019 zijn we van onze wintertijd (CET) naar onze zomertijd (CEST) overgegaan. Wat betekent het om “een dag” toe te voegen aan 16 uur op zaterdag.

```
ymd_hms("2019-03-30 16:00:00", tz = "CET") + days(1)

## [1] "2019-03-31 16:00:00 CEST"

ymd_hms("2019-03-30 16:00:00", tz = "CET") + ddays(1)

## [1] "2019-03-31 17:00:00 CEST"
```

Als we een menselijke periode toevoegen, zal het zeggen 16 uur op zondag, ook al is dat slechts 23 uur later. Als we een duration toevoegen, zal het ons vertellen dat “een dag later” zondag om 17 uur is, omdat er dan pas 24 uur zijn verstreken.

Natuurlijk is er geen verschil op een andere dag van het jaar.

```
ymd_hms("2019-04-30 16:00:00", tz = "CET") + days(1)

## [1] "2019-05-01 16:00:00 CEST"

ymd_hms("2019-04-30 16:00:00", tz = "CET") + ddays(1)

## [1] "2019-05-01 16:00:00 CEST"
```

Dus, het verschil is er alleen in speciale gevallen, maar toch een verschil.

12.8 Arithmetics

We hebben al heel wat berekeningen gezien die we kunnen doen. Hier zijn nog enkele voorbeelden.

```
doomsday >= now()

## [1] FALSE

now() - doomsday

## Time difference of 1521.141 days

as.period(now()-doomsday)

## [1] "1521d 3H 23M 0.908524990081787S"

doomsday + period(1,"week")

## [1] "2017-01-27 11:30:00 EST"

leap_year(2012)

## [1] TRUE

leap_year(now())
```

```
## [1] FALSE

now() + weeks(0:2)

## [1] "2021-03-21 20:53:00 CET"  "2021-03-28 20:53:00 CEST"
## [3] "2021-04-04 20:53:00 CEST"
```

Soms kunnen berekeningen fout gaan. Vooral met maanden.

```
ymd(20190331) + months(1)
```

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

Nou, 31 april bestaat niet echt. (Ik denk dat we allemaal wel een extra 24 uur zouden willen hebben, maar dat is niet zo.)

U kunt de speciale `%m+%` operator gebruiken om ervoor te zorgen dat je toevoegingen nooit over het einde van de maand gaan.

```
ymd(20190331) %m+% months(1)
```

```
## [1] "2019-04-30"
```

Hieronder hebben we beide geprobeerd een lijst te maken van alle maanden hun laatste dag in 2019.

```
ymd(20190131) + months(0:11)
```

```
## [1] "2019-01-31" NA          "2019-03-31" NA          "2019-05-31"
## [6] NA          "2019-07-31" "2019-08-31" NA          "2019-10-31"
## [11] NA          "2019-12-31"
```

```
ymd(20190131) %m+% months(0:11)
```

```
## [1] "2019-01-31" "2019-02-28" "2019-03-31" "2019-04-30" "2019-05-31"
## [6] "2019-06-30" "2019-07-31" "2019-08-31" "2019-09-30" "2019-10-31"
## [11] "2019-11-30" "2019-12-31"
```

12.9 Interval

Laten we tenslotte kijken naar interval. Durations en perioden hebben alleen een lengte. Een periode of een duur van een bepaalde lengte kan op elk moment voorkomen. Een interval is anders, want het verwijst naar een specifiek venster in de tijd. Het wordt niet gedefinieerd door zijn lengte, maar eerder door zijn begin- en eindpunt.

Victor Hugo leefde van 26 februari 1802 tot 22 mei 1885. We kunnen dus een interval definiëren aan de hand van deze twee data, die zijn leven vertegenwoordigen

```
hugo <- interval(mdy(02261802), mdy(05221885))
hugo
```

```
## [1] 1802-02-26 UTC--1885-05-22 UTC
class(hugo)

## [1] "Interval"
## attr(,"package")
## [1] "lubridate"
```

Merk op dat we natuurlijk ook tijdstippen kunnen gebruiken om een interval te definiëren. (We hebben alleen geen specifieke informatie over Hugo's geboorte- of sterfuur).

Oscar Wilde leefde van 16 oktober 1854 tot 30 november 1900. Dus,

```
wilde <- interval(mdy(10161854), mdy(11301900))
```

```
wilde
```

```
## [1] 1854-10-16 UTC--1900-11-30 UTC
```

We kunnen ook een interval maken met de speciale %-% operator. Dus de bovenstaande code is equivalent met

```
wilde <- mdy(10161854) %--% mdy(11301900)
wilde
```

```
## [1] 1854-10-16 UTC--1900-11-30 UTC
```

Het is korter om te schrijven, maar het gebruik van de functie is iets leesbaarder.

We kunnen controleren of een datum in een interval valt met de functie %within%. (Niet te verwarren met de %in% functie. Wat deed die weer?).

Charles Dickens is geboren op 7 februari 1812.

```
dickens_birthday <- dmy(7021812)
```

```
dickens_birthday %within% hugo
```

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

```
dickens_birthday %within% wilde
```

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

Hugo leefde op die dag, omdat deze dag binnen het interval van zijn leven valt. Wilde liep niet op de aarde op die dag.

We kunnen nagaan of twee intervallen elkaar overlappen. D.w.z. leefden Hugo en Wilde beiden op een bepaald moment in de geschiedenis.

```
int_overlaps(hugo, wilde)
```

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

Natuurlijk deden ze dat. We kunnen ook het exacte tijdsinterval creëren waarin beiden leefden.

```
intersect(hugo, wilde)
```

```
## [1] 1854-10-16 UTC--1885-05-22 UTC
```

Of het totale tijdsinterval dat ten minste één van hen leefde

```
union(hugo, wilde)
```

```
## [1] 1802-02-26 UTC--1900-11-30 UTC
```

Merk op dat union en intersect functies zijn die je ook kan gebruiken op bijvoorbeeld vectoren:

```
intersect(1:10, 5:15)
```

```
## [1] 5 6 7 8 9 10
```

Het zijn base-R functies die aangepast zijn voor vectoren door dplyr, en voor tijdstippen door lubridate. Als je `library(dplyr)` doet en vervolgens `library(lubridate)`, zie je de volgende boodschap:

```
> library(dplyr)
```

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

```
filter, lag
```

The following objects are masked from ‘package:base’:

```
intersect, setdiff, setequal, union
```

```
> library(lubridate)
```

Attaching package: ‘lubridate’

The following objects are masked from ‘package:base’:

```
date, intersect, setdiff, union
```

De boodschap *The following objects are masked from* hebben we voorheen genegeerd, maat het wijst ons erop dat de packages functies

van andere packages *overschrijven* of *aanvullen*. In de meeste gevallen vormt dit geen probleem, en zal R erin slagen de juiste functie aan te spreken. Zo kan het hierboven zonder problemen de intersectie tussen twee numerieke vectoren en tussen twee intervallen op de correcte manier.

Maar, het is toch het vermelden waard dat je op een punt gekomen bent dat er conflicten starten te ontstaan tussen verschillende packages. Welkom in de dagelijkse wereld van een echte programmer (of toch, een beetje). Proficiat met het vervolledigen van deze laatste tutorial, waardoor je nu een stevige basis hebt wat betreft data visualisatie, manipulatie, cleaning, transformatie en tijdsdata! Super!

— The End —