Inhoudsopgave

1

Introductie

1.1 Vooraf

Dit boek (An Introduction to Statistical and Data Sciences via Data van Chester Ismay en Albert Y. Kim) is geïnspireerd door drie boeken:

- "Mathematical Statistics with Resampling and R" (Chihara and Hesterberg, 2011),
- "Intro Stat with Randomization and Simulation" (Diez et al., 2014), and
- "R for Data Science" (Grolemund and Wickham, 2016).

Het eerste boek, voor hoger niveau bachelor- en masterstudenten ontwikkeld, is een goede bron dat zicht geeft op statistische concepten zoals normaal verdelingen via het gebruik van computers in plaats van de nadruk op het herinneren van formules. De andere twee boeken zijn een gratis alternatief voor de traditioneel dure introductieboeken statistiek. Als je het grote aantal van deze introductieboeken overziet, zie je dat er niet een is die de nieuwe R-pakketten in de tekst incorporeert. Daarnaast was er geen open-source, gratis tekstboek beschikbaar dat de nieuwe studenten het volgende biedt:

1. hoe is R te gebruiken om data te exploreren en te visualiseren?;
2. hoe zijn randomisatie en simulatie te gebruiken is om inferentiële ideeën op te bouwen?;
3. hoe zijn verhalen effectief op te bouwen om de informatie op een lekenpubliek over te brengen?

We zullen soms ingewikkelde statistische concepten met datavisualisatie introduceren. Vandaag de dag worden we gebombardeerd met grafieken om ideeen over te brengen. We zullen uitzoeken wat iets tot een goede grafiek maakt en wat standaard manieren zijn om relaties binnen data aan te tonen. Je zult ook het gebruik van visualisatie zien om concepten als gemiddelde, mediaan, standaard deviatie, verdelingen en dergelijke te introduceren. In het algemeen zullen we visualisatie gebruiken als een manier om bijna alle ideën in dit boek op te bouwen.

Aanvullend legt dit boek de nadruk op de triade van computer denken, data denken en inferentieel denken. We zullen door het hele boek heen zien dat deze drie manieren van denken effectieve manieren kunnen zijn om mee te werken, zaken te beschrijven en statistische

kennis op te bouwen. Om dat te doen zul je het belang van programmeren leren inzien om goed data wetenschap te kunnen uitvoeren. Met andere woorden, je zult zien hoe je codes en beschrijvingen moet gebruiken waar niet alleen de computer mee uit de voeten kan maar waarmee de lezers ook precies begrijpen wat een statistische analyse doet en hoe het werkt. Van Hal Abelson is de uitspraak die we in dit boek volgen:

> "Programma's moeten voor mensen zijn geschreven om te lezen en alleen incidenteel voor machines om uit te voeren."

1.2 Drie basis data bronnen

Inplaats van de ene dataset naar de andere te hoppen in het boek, hebben we besloten om ons in dit boek te richten op drie verschillende databronnen:

- vluchten die New York City in 2013 verlaten;
- profielen van OKCupid gebruikers in San Francisco;
- IMDB film scores.

Door juist drie grote datasets te gebruiken, hopen we dat je in staat bent te doorzien hoe de hoofdstukken met elkaar zijn verbonden. Je zult zien hoe opgeschoonde dataset leiden naar datavisualisatie en manipulatie in het exploreren van data-analyse en hoe deze concepten weer verbonden zijn met inferentie en regressie (hoewel dit deel niet over regressie en inferentie gaat).

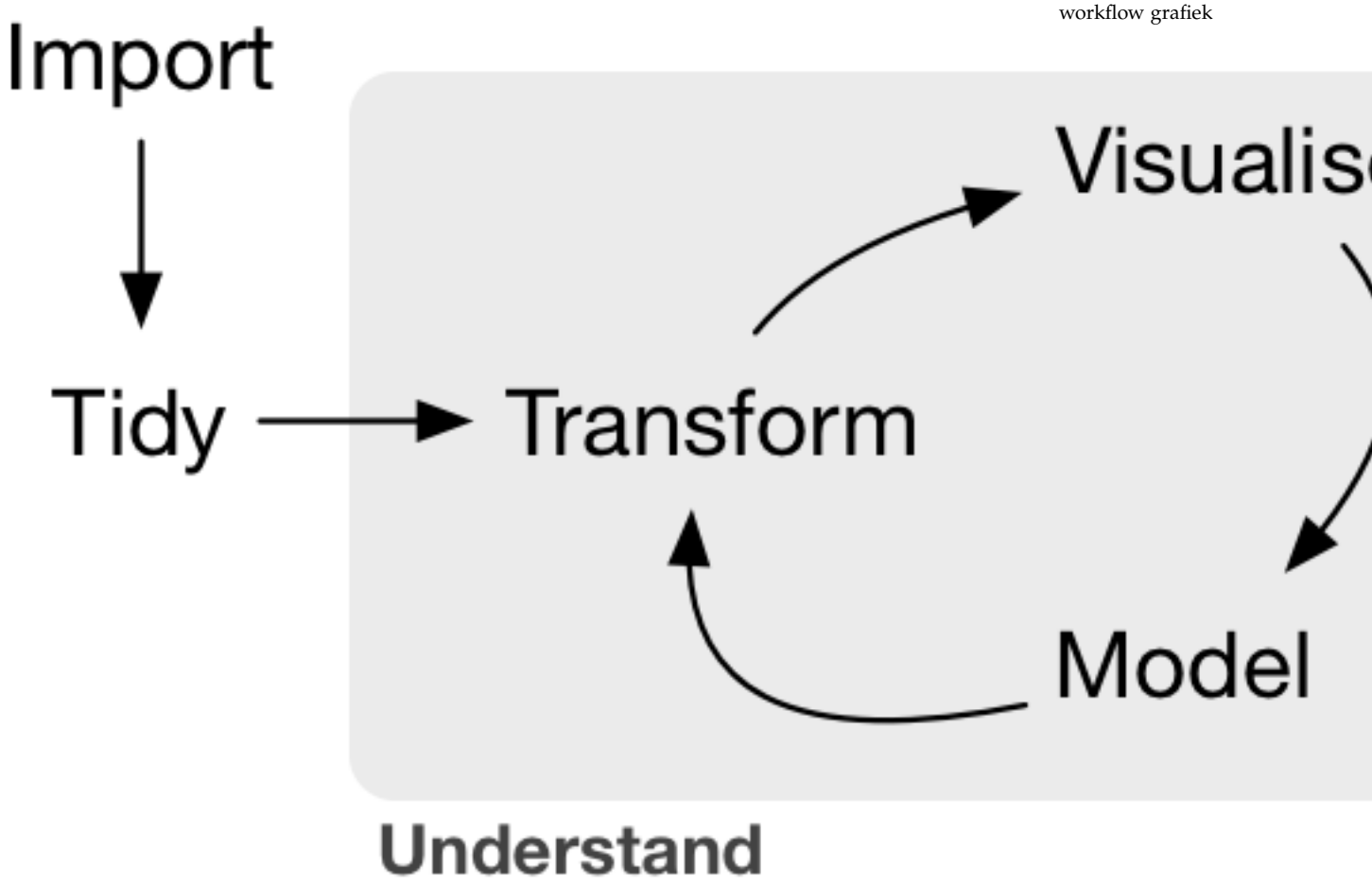
1.3 De pijplijn van dataanalyse

Misschien denk je dat het bij statistiek alleen maar gaat om een aantal getallen. In het algemeen horen we over 'statistiek' wanneer we naar sportevenementen luisteren. Statistiek (vooral data analyse), in aanvulling op het beschrijven van slaggemiddelden bij honkbal, speelt een vitale rol in alle wetenschap. Je hoort heel vaak over "statistisch significant" in the media. Je ziet daar dingen als "Wetenschap laat nu zien dat chocolade goed voor jou is." Het onderschrijven van dit soort claims is ook data analyse. Aan het einde van het boek zul je beter begrijpen of deze uitspraken vertrouwd of juist gewantrouwd moeten worden. In de data analyse zijn er nog vele sub-velden die we in het boek zullen bespreken zoals (maar niet noodzakelijk in deze volgorde):

1. data verzameling;
2. data manipulatie;
3. data visualisatie;
4. data modeleren;
5. inferentie;
6. correlatie en regressie;
7. interpretatie van resultaten;

8. verhalen vertellen aan de hand van data.

Dit kan worden samengevat in a grafiek die gewoonlijk door Hadley Wickham wordt gebruikt:



Figuur 1.1: Hadley's workflow grafiek

We zullen beginnen met een discussie over wat opgeschoonde data betekent en graven dan dieper in het grijze **Begrijpen** deel van de cirkel en daarna hebben we het over interpretatie en discussie van de resultaten van onze modellen via **Communicatie**. Deze stappen zijn essentieel voor elke statistische analyse. Maar waarom is statistiek überhaupt belangrijk? "Waarom moest ik deze lessen volgen?"

Er is een reden waarom zoveel terreinen een statistiek cursus vereisen. Wetenschappelijke kennis groeit door het begrijpen van statistische significantie en data-analyse. Maar je moet je niet laten intimideren door statistiek. Het is niet dat beest wat het was en, gecombineerd met de computer, zul je zien hoe vooral met reproduceerbaar onderzoek wetenschappelijke kennis toeneemt.

1.4 Reproduceerbaarheid

> "Het meest belangrijke gereedschap is de *mindset* dat, als je start, het eindproduct reproduceerbaar is." Keith Baggerly

Een ander belangrijk doel is de lezers het belang van reproduceerbare analyses te begrijpen. Het is te hopen dat de lezers de gewoonte eigen maken dat hun onderzoek reproduceerbaar is vanaf het begin. Dit betekent dat wij jou helpen met het eigen maken van deze nieuwe gewoontes. Dat vereist oefening en dat zal moeilijk zijn bij tijd en wijle. Je zult zien waarom het zo belangrijk is dat je grip houdt op jouw code en dat je het goed documenteert zodat je jezelf later kunt helpen en iedere andere persoon met wie je samenwerkt ook.

Kopiëren en resultaten plakken van het ene programma in een tekstprogramma is niet de manier waarop efficiënt en effectief wetenschappelijk onderzoek wordt uitgevoerd. Het is veel belangrijker om tijd te besteden aan dataverzameling en data-analyse en niet al je tijd te besteden aan het heen en weer kopiëeren en plakken van grafieken bijvoorbeeld van het ene naar het andere programma.

Wanneer er in de traditionele data-analyse er een fout was gemaakt in de orgninele data, moesten we weer door het hele proces heen: opnieuw de grafieken maken en deze kopiëren en plakken in jouw tekstdocument evenals de hele statistische analyse zelf. Zo zijn er makkelijk fouten te maken en het is een frustrerende manier van tijdbesteding. We zullen zien hoe we hier R Markdown voor kunnen gebruiken en dat we meer tijd over hebben om aan wetenschap te besteden.

> "We praten over *computational* reproduceerbaarheid." - Yihui Xie

Reproduceerbaarheid betekent veel voor verschillende wetenschappelijke velden. Zijn de experimenten uitgevoerd op een manier dat een andere wetenschapper de stappen kan volgen en dezelfde resultaten krijgt? In dit document leggen we de nadruk op wat er bekend is als we de focus leggen op **computational reproduceerbaarhied**. Dit refereert ernaar dat je in staat bent alles van de dataanalyse, data sets en conclusies aan iemand anders te geven en dat hij of zij dan dezelfde resultaten krijgt op zijn of haar machine. Zo heb je meer tijd op aan wetenschap, interpreteren van resultaten en maken van assumpties te besteden in plaats van helemaal opnieuw te moeten beginnen of een lijst van stappen te volgen die voor iedere machine anders is.

1.5 Voor wie is dit boek?

Dit boek is voor studenten die een traditionele introductiecursus statistiek volgen in een kleine omgeving en daarbij RStudio of RStudio Server gebruiken. Er wordt geen voorkennis verondersteld: geen algebra, rekenvaardigheden of programmeerkennis. Dit is bedoeld als een prettige en aardige introductie in de statistiek waar het gaat over hoe data wetenschappers, statistici, data journalisten en andere wetenschappers data analyseren en verhalen schrijven over data. Met opzet vermijden we het gebruik van formules zoveel mogelijk. In

plaats daarvan leggen we de nadruk op statistische concepten zoals datavisualisatie en statistisch computer gebruik. Wij hopen dat dit een meer intuïtieve ervaring oplevert dan de manier waarop traditioneel statistiek in het verleden werd onderwezen (en hoe er van de buitenkant nog naar wordt gekeken). Aanvullend hopen we dat je de waarde in gaat zien van reproduceerbaar onderzoek met R verderop in jouw studie. We begrijpen dat je in het begin tegen problemen oploopt als je het programmeren eigen wilt maken maar we zijn hier om jou te helpen en je moet weten dat er een grote gemeenschap van R-gebruikers is die ook graag nieuwkomers helpt.

Laten we nu beginnen met het leren over hoe goede verhalen zijn te maken over en met data!

(DEEL) Data Exploratie

2

Opgeschoonde data

Hier zullen we het hebben over het belang van opgeschoonde data. Jij denkt misschien dat het dan enkel gaat om hoe je de data in een spreadsheet krijgt, maar je zult zien dat het meer inhoudt. Data komen naar ons toe in verschillende formaten van plaatjes naar tekst naar enkel getallen. Wij zullen ons hier richten op datasets die opgeslagen kunnen worden in een spreadsheet omdat dat de meest normale wijze is waarop data in wetenschap worden opgeslagen.

Met opgeschoonde data kunnen we makkelijker onze data visualiseren zoals we zullen zien in Hoofdstuk 3. Het ondersteunt ons ook bij het manipuleren van data zoals we zien Hoofdstuk 4 en ook in alle hoofdstukken die gaan over statistische inferentie maar die hier niet aan de orde komen. Misschien dat je niet meteen het belang van **opgeschoonde data** begrijpt maar dat komt wel als we verder in het boek komen.

Benodigde pakketten

Bij het begin hiervan en al de volgende hoofdstukken hebben we steeds een lijst van pakketten nodig die je moet installeren en laden. We hebben het `nycflights13` pakket nodig dat we kort zullen bespreken en het `dplyr` pakket voor datamanipulatie, het onderwerp van Hoofdstuk 4. We laden ook het `tibble` pakket hier, dat de bruikbare `glimpse` functie omvat.

```
library(nycflights13)
library(dplyr)
library(tibble)
```

2.1 Wat betekent opgeschoonde data?

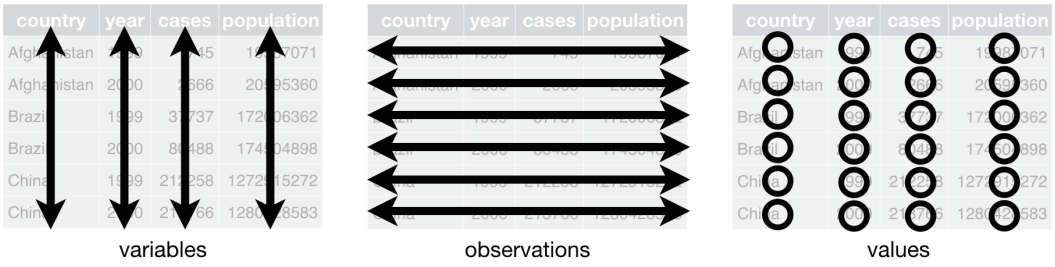
Je hebt vast van het woord "opgeschoond (tidy in het Engels)" gehoord zoals:

- "Maak je kamer schoon!"
- Marie Kondo's goed verkopende boek *The Life-Changing Magic of Tidying Up: The Japanese Art of Decluttering and Organizing*

Dus wat betekent het als je data **opgeschoond** moeten zijn? Eenvoudig gezegd betekent het dat jouw data georganiseerd moeten zijn. Maar het is meer dan enkel dat. Het betekent dat jouw data een standaard format volgen die het voor anderen makkelijk maakt om delen van jouw data te vinden, te manipuleren en om te vormen, en ook om jouw data makkelijker te visualiseren en de relaties tussen verschillende variabelen in jouw data te zien.

We zullen hier Hadley Wickham's definitie van **opgeschoonde data** volgen (**?**):

> Een dataset is een verzameling waarden, meestal of getallen (kwantitatief) of woorden (kwalitatief). Waarden zijn op twee manieren georganiseerd. Elke waarde behoort tot een variabele en een observatie. Een variabele omvat alle waarde die hetzelfde onderlinge attribuut meet (zoals hoogte, temperatuur, duur) over eenheden. Een observatie omdat alle waarden gemeten over dezelfde eenheid (zoals een persoon, een dag, een wedstrijd) over attributen.
>
> Opgeschoonde data is een standaard manier om de betekenis van een dataset in relatie tot de structuur uit te drukken. Een dataset is een rotzooitje of opgeschoond afhankelijk van hoe rijen, kolommen en tabellen corresponderen met observaties, variabelen en typen. In **opgeschoonde data**:
>
> 1. Vormt elke variabele een kolom.
> 2. Vormt elke observatie een rij.
> 3. Vormt elke type observatieeenheid een tabel.



Figuur 2.1: Opgeschoonde data grafiek van http://r4ds.had.co.nz/tidy-data.html

Als je deze definitie leest, kun je gaan nadenken over datasets die niet deze duidelijke opbouw kennen.

2.2 *Datasets in het `nycflights13` pakket*

Als je op een vliegveld rondloopt, zie je overal vluchtinformatie. En je ziet dan ook dat regelmatig dat bepaalde vluchten vertraagd zijn vanwege verschillende redenen. Zijn er manieren waarmee we dit kunnen voorkomen?

We willen waar mogelijk allemaal op tijd op onze bestemmingen aankomen. In dit boek zullen we data analyseren die gerelateerd zijn aan de vluchten in het `nycflights13`-pakket dat we eerder hebben geladen (**?**). Dit pakket omvat informatie over alle vluchten die van NYC vertrokken (b.v. EWR, JFK en LGA) in 2013 in 5 data sets:

- `flights`: informatie over alle 336,776 vluchten
- `weather`: meterologische data van elk uur en elke luchhaven
- `planes`: constructieinformatie over elk vliegtuig
- `airports`: namen van luchthavens en locaties
- `airlines`: vertaling van twee letter codes en namen

We zullen eerst de `flights` dataset laden en krijgen dan een idee van de structuur. Draai het volgende in jouw console

```
data(flights)
```

Deze code laadt de `flights` dataset die is opgeslagen in het `nycflights13` pakket. Deze dataset, en de meeste andere die in dit boek worden gepresenteerd, staan in het "data-frame" format van R. Data-frames zijn essentiële spreadsheets en stellen ons in staat naar een verzameling variabelen te kijken die aan elkaar gekoppeld zijn.

De beste manier om een gevoel te krijgen van een data-frame is via het gebruik van de `View` functie in RStudio. Dit commanda wordt het hele boek door gegeven als een herinnering, maar de werkelijke output zullen we niet laten zien. Run `View(flights)` in R en bekijk de data-frame. je zult langzaam de gewoonte krijgen om altijd alle data-frames te `View`-en die jouw kant op komen.

Door het runnen van `View(flights)` zien we de verschillende **variabelen** die in de kolommen staan en zien we dat er verschillende variabelen te onderscheiden zijn. Enkele variabelen zoals `distance`, `day` en `arr_delay` en dat zijn wat we noemen **kwantitatieve** variabelen. Deze variabelen verschillen op numerieke wijze. Andere variabelen hier zijn **categoriale variabelen**.

Wanneer je naar de meest linkse kolom van de `View(flights)` output kijkt, zie je een kolom met getallen. Dit zijn de rijnummers van de dataset. Als je een blik werpt op een rij met hetzelfde nummer, zeg rij 5, krijg je een idee waar elke rij voor staat. Met andere woorden, zo kun je zien waarnaar een bepaalde rij refereert. Dit wordt vaak de **observationele eenheid** genoemd. De **observationele eenheid** in dit voorbeeld is een individuele vlucht die in 2013 vanuit New York City vertrekt.

Opgelet: Het eerste dat je vaak moet doen wanneer je een dataset onder ogen krijgt is:

- Identificeren van de observationele eenheid;
- Specificeren van de variabelen;
- Het vaststellen van de typen variabelen.

Het `glimpse()` commando in het `tibble` pakket biedt ons deze informatie en nog veel meer:

```
glimpse(flights)
```

```
## Observations: 336,776
## Variables: 19
```

```
## $ year           <int> 2013, 2013, 2013,...
## $ month          <int> 1, 1, 1, 1, 1, 1,...
## $ day            <int> 1, 1, 1, 1, 1, 1,...
## $ dep_time       <int> 517, 533, 542, 54...
## $ sched_dep_time <int> 515, 529, 540, 54...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, ...
## $ arr_time       <int> 830, 850, 923, 10...
## $ sched_arr_time <int> 819, 830, 850, 10...
## $ arr_delay      <dbl> 11, 20, 33, -18, ...
## $ carrier        <chr> "UA", "UA", "AA",...
## $ flight         <int> 1545, 1714, 1141,...
## $ tailnum        <chr> "N14228", "N24211...
## $ origin         <chr> "EWR", "LGA", "JF...
## $ dest           <chr> "IAH", "IAH", "MI...
## $ air_time       <dbl> 227, 227, 160, 18...
## $ distance       <dbl> 1400, 1416, 1089,...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5,...
## $ minute         <dbl> 15, 29, 40, 45, 0...
## $ time_hour      <dttm> 2013-01-01 05:00...
```

We zien dat `glimpse` jou de eerste informatie over elke variabele geeft in een rij. Daarnast zie je informatie over het type variabele dat direct gegeven wordt naar de variabelenaam in `< >`. Hier refereren de `int` en `num` naam kwantitatieve variabelen en `chr` refereert naar categoriale variabelen. Een ander type variabele wordt hier gegeven in de `time_hour` variabele: **dttm**. Zoals je mag verwachten correspondeert deze variabele met een specifieke datum en tijd van de dag.

Een ander aardig kenmerk van R is het helpsysteem. Je kunt hulp van R vragen door simpel een vraagteken te plaatsen voor de naam van een functie of een object en jou wordt een pagina voorgeschoteld met deze informatie. Omdat `glimpse` een functie is binnen het `tibble` pakket, kun je dat benadrukken dat je hier hulp zoekt door twee keer een dubbele punt te plaatsen tussen de naam van het pakket en de functie. Let op dat je hier niet de outputhelpfiles ziet maar de `flights` hulp kun je krijgen via hier op pagina 3 van het PDF document.

```
?tibble::glimpse
?flights
```

Een ander aspect van opgeschoonde data is een beschrijving van wat en waar elke variabele van in een dataset voor is en staat. Dat helpt anderen te begrijpen wat jouw variabelennamen betekenen en waar ze voor staan. Als we naar de output voor `?flights` kijken, zien we een beschrijving van elke variabelenaam.

Iets anders wat belangrijk is om **ALTIJD** in jouw data te includeren is de juiste meeteenheid. We zullen dit verder zien als we met de `dep_delay` variabele in Hoofdstuk 3 werken.

2.3 *Hoe is `flights` op te schonen?*

We zien dat `flights` een rechthoekige vorm heeft met elke rij die naar een verschillende vlucht refereert en elk kolom naar een karakteristiek van die vlucht. Dat komt precies overeen met hoe Hadley Wickham opgeschoonde data definieert:

1. Elke variabele vormt een kolom;
2. Elke observatie vormt een rij;

Maar wat betekent dat voor de derde eigenschap?

3. Elk type observationele eenheid vormt een tabel.

We hebben eerder opgemerkt dat een observationele eenheid in de `flights` dataset een individuele vlucht is. En we hebben laten zien dat deze dataset bestaat uit 336,776 vluchten met 19 variabelen. Met andere woorden, sommige rijen van deze dataset refereren niet naar een meting van een luchtvaartmaatschappij of een luchthaven. Ze refereren naar karakteristieken of metingen van een gegeven **vlucht** vanaf New York City in 2013.

In het `nycflights13` pakket zitten ook datasets met andere observationele eenheden (**?**):

- `weather`: meteorologische data van uur tot uur voor elk vliegveld;
- `planes`: constructieinformatie over elk vliegtuig;
- `airports`: namen van vliegvelden en locaties;
- `airlines`: vluchtlijnen, overzetten van twee lettercodes en namen.

Je mag je afvragen waar `carrier` naar refereert in de `glimpse(flights)` output hierboven. De `airlines` dataset geeft een beschrijving hiervan met luchtvaartmaatschappij als observationele eenheid:

```
data(airlines)
airlines
```

```
## # A tibble: 16 x 2
##    carrier                       name
##      <chr>                      <chr>
##  1      9E           Endeavor Air Inc.
##  2      AA      American Airlines Inc.
##  3      AS        Alaska Airlines Inc.
##  4      B6             JetBlue Airways
##  5      DL        Delta Air Lines Inc.
##  6      EV    ExpressJet Airlines Inc.
##  7      F9      Frontier Airlines Inc.
##  8      FL AirTran Airways Corporation
##  9      HA      Hawaiian Airlines Inc.
```

```
## 10      MQ                  Envoy Air
## 11      OO       SkyWest Airlines Inc.
## 12      UA       United Air Lines Inc.
## 13      US             US Airways Inc.
## 14      VX              Virgin America
## 15      WN      Southwest Airlines Co.
## 16      YV          Mesa Airlines Inc.
```

Zoals je kunt zien als een naam van een object in R intikt, zal het standaard de inhoud van dat object laten zien op het scherm. Wees voorzichtig! Het is meestal beter om de `View()` functie in RStudio te gebruiken omdat grotere objecten een tijd duren om op het scherm geprint te krijgen en het is ook niet handig voor jou om honderden regels output te krijgen.

2.3.1 Identificatie variabelen

Er is soms een subtiel verschil in variabelen van de data frames. Het `airports` data-frame waar je hierboven mee werkte bevat verschillende soorten data. Laten we ze eens uit elkaar trekken door het gebruik van de `glimpse` functie:

```
glimpse(airports)
```

```
## Observations: 1,458
## Variables: 8
## $ faa   <chr> "04G", "06A", "06C", "06N"...
## $ name  <chr> "Lansdowne Airport", "Moto...
## $ lat   <dbl> 41.13047, 32.46057, 41.989...
## $ lon   <dbl> -80.61958, -85.68003, -88....
## $ alt   <int> 1044, 264, 801, 523, 11, 1...
## $ tz    <dbl> -5, -6, -6, -5, -5, -5, -5...
## $ dst   <chr> "A", "A", "A", "A", "A", "...
## $ tzone <chr> "America/New_York", "Ameri...
```

De variabelen `faa` en `name` zijn wat we noemen *identificatie variabelen*. Ze worden meestal gebruikt om een naam te geven aan de observationele eenheid. Hier is de observationele eenheid een luchthaven en de `faa` geeft de code die door FAA is gegeven voor die luchthaven terwijl de `name` variabele de langere, natuurlijke naam van de luchthaven geeft. Deze ID variabelen verschillen van de andere variabelen, meestal de *meet* of *karakteristieke* variabelen. De andere variabelen in `airports` (dus buiten `faa` en `name`) zijn van dit type. Ze identificeren de observationele eenheid niet in de eerste plaats, maar beschrijven de eigenschappen van de observationele eenheid veel meer. Vanwege organisatorische doelen is het aan te bevelen jouw id-variabelen in de meest linkse kolommen van jouw data-frame te plaatsen.

2.4 *Normale vormen van data*

De datasets van het `nycflights13` pakket zijn zo opgeslagen dat de overtolligheid van data beperkt is. We zullen zien dat het makkelijk is om de verschillende tabellen samen te voegen () *merge / join*. We zijn daartoe in staat omdat de tabellen sleutels hebben die gerelateerd zijn aan elkaar. Dit is een belangrijke eigenschap van **normale vormen** van data. Dit proces om minder overtollige data te krijgen zonder informatie te verliezen wordt **normalisatie** genoemd. Meer informatie hierover vind je op Wikipedia.

We zag hierboven hiervan een voorbeeld met de `airlines` dataset. Terwijl het `flights` data frame met namen van de vluchtlijnen kon omvatten in plaats van de 'carrier code', zou dit herhaling betekenen omdat er een unieke koppeling is van 'carrier code' met de naam van de vluchtlijn.

Hieronder zie je een voorbeeld van hoe de vluchtlijnen data-frame (`airlines` data)samen is te voegen (**join**) met het vlucht data-frame (`flights` data frame). Dat gebeurt door de twee datasets te koppelen via een gemeenschappelijke **sleutel** van `"carrier"`. Deze "samengevoegde" data frame wordt opgeslagen als een nieuw dataframe dat `joined_flights` wordt genoemd. The **sleutel** variable waar we de sets regelmatig mee samenvoegen is een van de *identificatie variabelen* die we hierboven noemden.

```
library(dplyr)
joined_flights <- inner_join(x = flights, y = airlines, by = "carrier")
```

```
View(joined_flights)
```

Als we deze dataset bekijken (`View`) zien we een nieuwe variabele die naam heet (`name`). (We zullen in de paragraaf 4.4.2 manieren zien om deze naam te veranderen in een meer beschrijvende variabele naam.) Meer informatie over het samenvoegen van verschillende data-frames vind je in Hoofdstuk 4. We zullen daar zien dat de namen van de kolommen die gelinkt moeten niet overeenkomen zoals hier met `"carrier"`.

2.5 *Wat komt er nu?*

In Hoofdstuk 3 zullen we de verdeling van een variabele in een aan `flights` gerelateerde dataset bezien: de `temp` variabele in de `weather` dataset. We willen dan begrijpen hoe deze variabele varieert in relatie tot de waarden van andere variabelen in de dataset. We zullen zien dat visualisatie vaak een krachtig gereedschap is om te zien wat er omgaat in een dataset. Het is een nuttige aanvulling op de `glimpse` functie die we hier hebben gepresenteerd om de data op te schonen.

3
Data Visualisatie via ggplot2

In het vorig hoofdstuk spraken we over het belang dat datasets netjes zijn **opgeschoond**. Je zult in deze voorbeelden hierna zien waarom een opgeschoonde dataset ons enorm helpt bij het plotten van onze data. Met het plotten van onze data zijn we in staat veel meer in onze data te zien, veel meer dan we in de ruwe data konden zien. We zullen vooral Hadley Wickham's `ggplot2` gebruiken, dat is ontwikkeld om met **opgeschoonde** datasets te werken. Het biedt ons een eenvoudige manier om onze plots aan te passen zoals we willen en in gebaseerd op de datavisualisatietheorie zoals naar voren gebracht in *The Grammar of Graphics* (Wilkinson, 2005).

Vanaf het meest eenvoudige basisniveau af gezien bieden grafieken/plots/figuren ons een aardige manier om een gevoel te krijgen voor hoe kwantitatieve variabelen in elkaar zitten wat hun centrum en spreiding betreft. Dat wat je echt moet weten over grafieken is dat ze gemaakt zijn om jouw publiek te laten zien wat jij ze wilt laten zien. Dat vraagt een balans tussen niet te veel in een jouw plots willen stoppen, maar ook genoeg zodat de relaties en interessante bevindingen kunnen worden opgemerkt. Zoals we zullen zien, helpen plots/grafieken ons om patronen en uitschieters te ontdekken in onze data. We zullen zien dat een uitbreiding van deze ideeën is om de **verdeling** van een kwantitatieve variabele te zien.

Pakketten die we nodig hebben

Voordat we verder gaan met dit hoofdstuk, laden we al de benodigde pakketten die hier nodig zijn (deze moeten wel eerder zijn geïnstalleerd, zoals je ondertussen weet).

```
library(ggplot2)
library(nycflights13)
library(knitr)
library(dplyr)
```

3.1 *De grammatica van grafieken*

We beginnen met een discussie over een theoretisch raamwerk voor datavisualisatie dat we kennen als "De Grammatica van Grafieken", dat de basis vormt voor het `ggplot2`-pakket. Net zoals we zinnen vormen in elke taal door linguïstische grammatica te gebruiken (met zelfstandige naamwoorden, werkwoorden, subjecten, objecten, etc), stelt het theoretisch raamwerk zoals dat is opgesteld door Leland Wilkinson (Wilkinson, 2005) ons in staat componenten van een statistische grafiek te specificeren.

3.1.1 *Componenten of Grammatica*

Samengevat vertelt ons deze grammatica dat:

> **Een statistische grafiek `data`variabelen in kaart brengt langs esthetische attributen (`aes`thetic) van `geom`etrische objecten.**

Specifiek kunnen we een grafiek in de volgende drie essentiele componenten indelen:

1. `data`: de dataset die bestaat uit bepaalde variabelen die we in kaart willen brengen.
2. `geom`: het geometrisch object waar het om gaat. Dit refereert naar het type of objecten die we in onze plot kunnen observeren. Bijvoorbeeld punten, lijnen, staaf, ect.
3. `aes`: esthetisch ('aes-thetic') attribuut van het geometrisch object dat we aan de grafiek kunnen toevoegen. Bijvoorbeeld, x/y positie, kleur, vorm en omvang. Elk toegekend attribuut kan worden toegekend aan een variabele in onze dataset. Als ze niet worden toegekend aan variabelen, gelden de standaard waarden.

3.1.2 *Napoleons Mars naar Moskou*

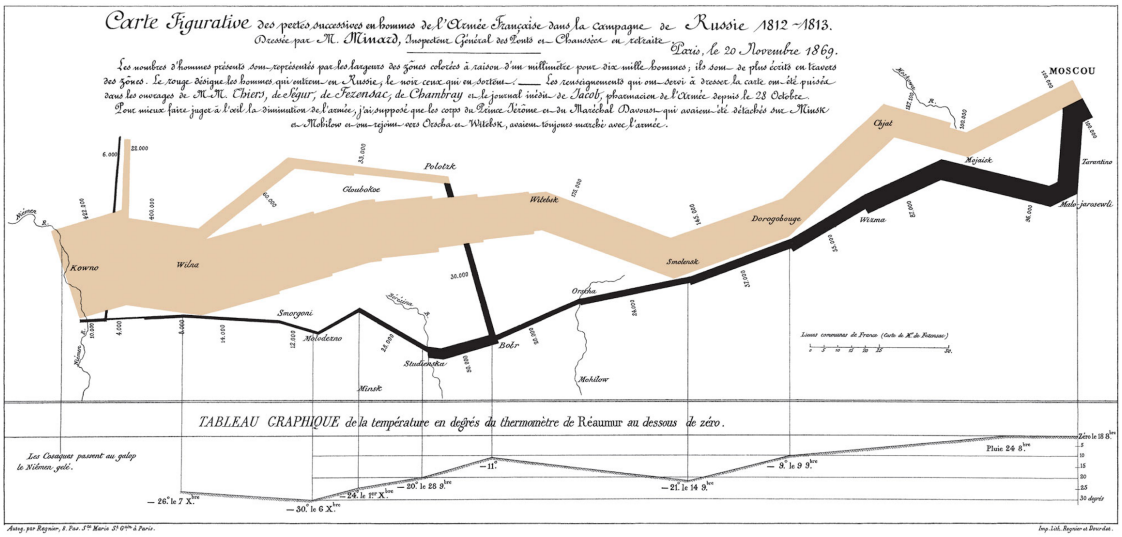
In 1812 leidde Napoleon een Franse invasie in Rusland door op Moskou te marcheren. Het was een van de grootste militaire rampen, ook vanwege de Russische winter. In 1869 publiceerde een Franse ingenieur (Charles Joseph Minard) misschien wel een van de grootste statistische visualisaties van alle tijden, die deze mars samenvatte:

Dit werd gezien als een revolutie in statistische grafieken omdat de kaart 6 dimensies van informatie geeft (bijvoorbeeld variabelen) die op een 2-dimensionale pagina worden afgebeeld. Laten we eens naar de grafiek kijken vanuit het perspectief van de Grammatica van Grafieken:

data	aes	geom
longitude	x	point
latitude	y	point
army size	size	path
army direction	color	path

data	aes	geom
date	x	line & text
temperature	y	line & text

Tabel 3.1: Grammatica van de kaart (Top) en lijngrafiek (Onderkant) in Minard's Grafiek van Napoleans Mars



Figuur 3.1: Minard's Visualizatie van de mars van Napolean

Bijvoorbeeld de datavariabele `lengtegraad` krijgt een `x` aesthetisch van punt- geometrische objecten voor de kaart terwijl de the annoteerde lijn-grafiek de datum- (`date`) en temperatuur- (`temperature`)-variabeleinformatie op de kaart toewijst aan via `x` en `y` aesthetiek van de geometrische lijnobject.

3.1.3 Andere Componenten van de Grammatica

Er zijn andere componenten van de Grammatica van Grafieken die we kunnen controleren:

- `facet`: waarmee een plot in subsets is op te breken
- statistische transformaties: bv samenbrengen tot waarden in een histogram.
- `scales` zowel
  - converteren van **data eenheden** naar **fysieke eenheden** waarmee de computer kan werken
  - legenda en/af assen tekenen, waarmee de orginele data in de grafiek goed zijn te lezen.
- `coordinatiesysteem` voor x/y waarden: vooral `cartesian`, maar dat kan ook `polar` of `map` zijn
- `position` (positie)aanpassingen

Hier richten we ons alleen op de eerste twee: `faceting` (geïntroduceerd in Sectie Facetten en `statistische` transformaties (in beperkte mate, wanneer we het hebben over Staafdiagrammen in Sectie Geombar; de andere componenten komen in een meer geavanceerde tekst aan de orde. Dat is geen probleem voor het maken van een grafiek om die andere componenten in standaard setting worden uitgevoerd.

Er zijn natuurlijk ook nog andere mogelijkheden die ons ter beschikking staan zoals de titel van de grafiek, labels voor de assen en aanvullende thema's voor de grafiek. In het algemeen

kunnen we met de Grammatica van de Grafieken alles aanpassen zoals we willen. Het heeft steeds een consistent raamwerk dat we kunnen toepassen als we onze creaties willen aanpassen.

3.1.4 Het ggplot2 Pakket

Nu zullen we Hadley Wickham's `ggplot2` pakket introduceren, dat een implementatie is van de Grammatica van Grafieken voor R. Misschien is het je opgevallen dat veel van de tekst tot nu toe hier in computerlettertype is geschreven. Dat is omdat verschillende componenten van de Grammatica van Grafieken zijn gespecificeerd door het gebruik van de `ggplot` functie, dat minimaal verwacht van de argumenten

- het dataframe waarin de variabelen bestaan (het `data` argument) en
- de namen van de variabelen die moeten worden geplot (het `mapping` argument).

De namen van de variabelen staan als argumenten in de `aes` functie waar `aes` staat voor estethica ("aesthetics").

3.2 Vijf benoemde grafieken - De 5 BG

Voor onze doelen zullen we ons beperken tot het bekijken van vijf verschillende typen grafieken (in deze tekst gebruiken we de termen "grafieken", "plotten" en "figuren" door elkaar heen). We praten hier over de **5BG** (vijf benoemde grafieken):

1. puntgrafieken
2. lijngrafieken
3. doosdiagrammen
4. histogrammen
5. staafdiagrammen

Met dit repertoire aan grafieken kun je een breed scala van datavariabelen aan waarmee je wordt geconfronteerd. We zullen hier wat variaties van bespreken, maar met de 5BG's in jouw gereedschapkist kun je veel aan! Iets waar we ook nog de nadruk op zullen leggen is dat bepaalde grafieken alleen werken voor categoriale/logische variabelen en andere alleen voor kwantitatieve variabelen. Je zult jezelf vaak willen testen op de vraag welke grafiek voor een bepaald probleem het meest geschikt is.

3.3 5BG#1: Puntgrafieken

De meest simpele van de 5BG zijn de **puntgrafieken** (ook wel bivariate grafieken genoemd); ze stellen jou in staat de relatie te onderzoeken tussen twee continue variabelen. Misschien ben je wel bekend met deze grafieken, maar laten we er toch eens naar kijken vanuit het perspectief van de Grammatica van Grafieken. Vanuit dat perspectief onderzoeken we grafisch de relatie tussen de volgende twee continue variabelen in het `flights` dataframe:

1. `dep_delay`: vertrekvertraging (departure delay) op de horizontale "x" as en
2. `arr_delay`: aankomstvertraging (arrival delay) op de verticale "y" as

voor de Alaska Airlines vluchten die NYC in 2013 verlaten. Daarvoor moeten we het `flights` dataframe versmallen tot `all_alaska_flights` dat alleen bestaat uit Alaska Airlines (met code "AS") vluchten.

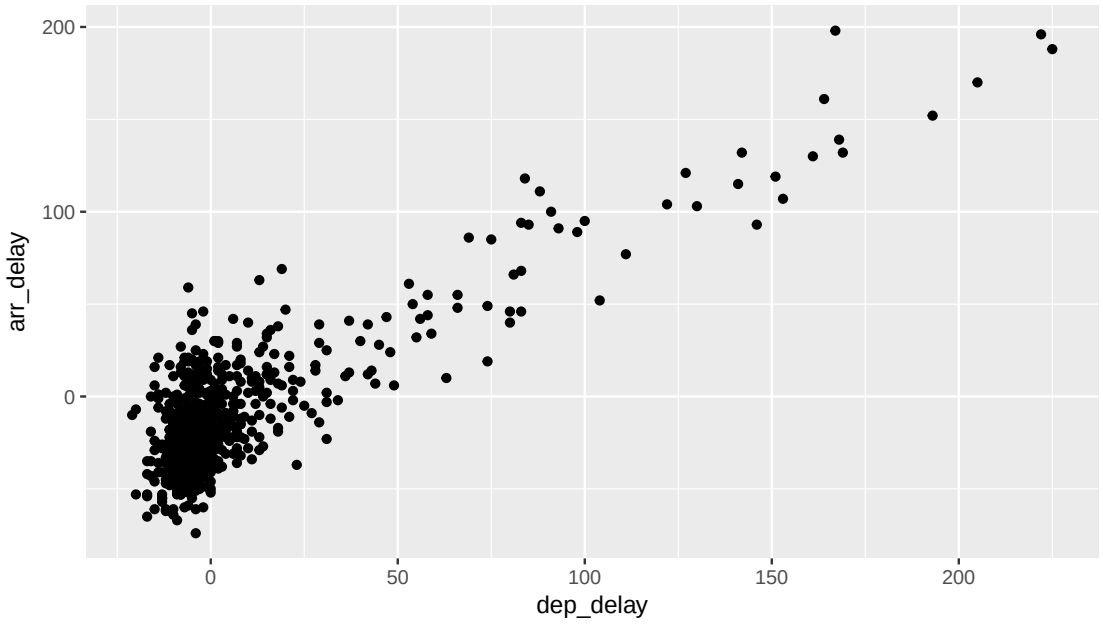
```
data(flights)
all_alaska_flights <- flights %>%
  filter(carrier == "AS")
```

Deze code maakt gebruik van functies in het `dplyr`-pakket voor datamanipulatie waarmee we ons doel kunnen bereiken: het neemt het `flights`-dataframe en filtert het (`filter`) tot enkel de rijen voldoen aan de `carrier == "AS"` voorwaarde (nogmaals, gelijk aan wordt gespecificeerd met == en niet =). Verschillende andere functies worden gebruikt in het Hoofdstuk Manipulatie.

3.3.1 Puntgrafieken via geom_point

We gaan verder met het maken van puntgrafieken via het gebruik van de `ggplot()` functie:

```
ggplot(data = all_alaska_flights, aes(x = dep_delay, y = arr_delay)) +
  geom_point()
```



Figuur 3.2: Aankomstvertragingen vs Vertrekvertragingen bij Alaska Airlines vluchten vanaf NYC in 2013

We raden je aan de **Returnknop** op jouw toetsenbord te gebruiken na de +. Omdat we meer en meer elementen toevoegen, is het handig om ze ingesprongen te gebruiken zoals we hieronder zullen zien. Let op dat dit niet werkt als je de regel met de + begint.

Laten we hierop terugkomen in de discussie van Sectie Grammatica van Grafieken:

- Binnen de`ggplot()`functie roepen we twee componenten van de grammatica op:
  1. Het `data`-frame `all_alaska_flights` via `data = all_alaska_flights`
  2. De `aesthetische` mapping via `aes(x = dep_delay, y = arr_delay)`. Daarbij brengt
     - `dep_delay` de `x` positie in kaart
     - `arr_delay` de `y` positie in kaart
- We voegen een **laag** toe aan de `ggplot()`-functie door het gebruik maken van het + teken
- De laag waar het hier om gaat specificeert de derde component van de grammatica: het`geometrisch` object waar het om gaat. Hier gaat het om het geometrisch object punten (`points`), benoemd door `geom_point()`

In het Figuur zien we dat er een positieve relatie bestaat tussen de twee variabelen `dep_delay` en `arr_delay`: als de vertrekvertragingen toenemen, hebben de aankomstvertragingen ook de neiging om toe te nemen. De meerderheid van de punten vallen in de omgeving van het punt (0, 0). Een groot deel van de punten clusteren hier samen.

3.3.2 *Over-Plotten*

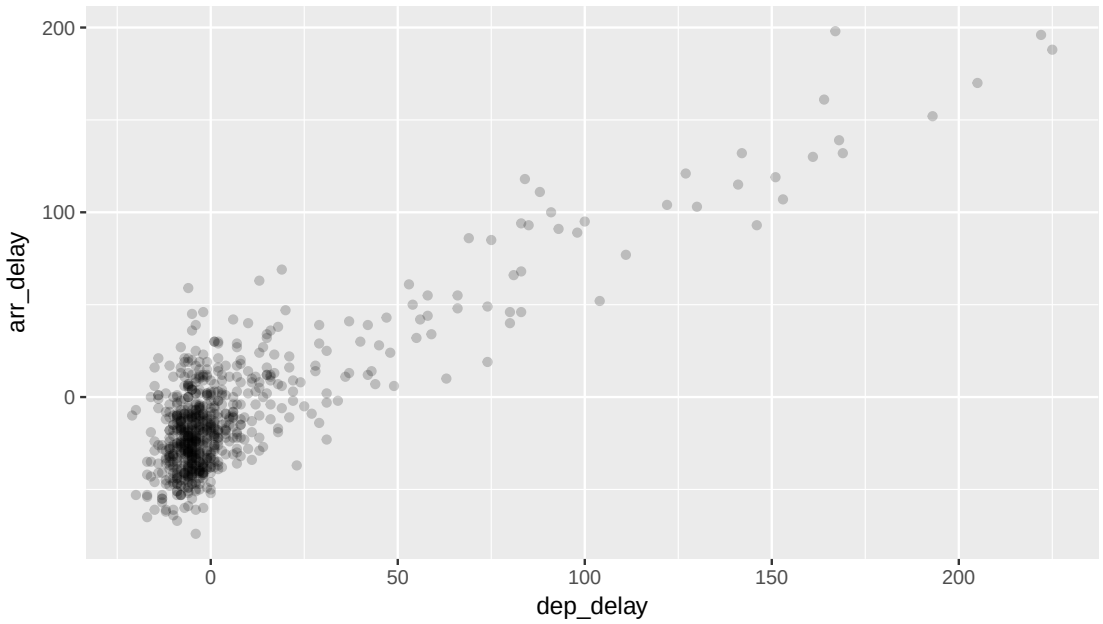
Het merendeel van de punten zit in de buurt van (0, 0) en dat veroorzaakt enige verwarring. Dat is het resultaat van een fenomeen dat **over-plotting** wordt genoemd. Zoals je mag veronderstellen heeft dat weer te maken met waarden die *over* en *over* elkaar heen worden afgedrukt. Het is dan moeilijk om vast te stellen hoeveel waarden hier zijn afgedrukt door enkel te kijken naar een basis puntengrafiek zoals we die hier hebben. Er zijn twee manieren om dit aan te pakken:

1. Door de transparantie van de punten aan te pakken via het `alpha` argument
2. Door het jitteren van punten via `geom_jitter()`

De eerste manier om met over-plotting om te gaan is door het aanpassen van het `alpha` argument in `geom_point()` die de transparentie van de punten regelt. De standaard- (of default-waarde) staat op `1`. Hier kunnen we iets kleiners van maken (maar wel groter dan 0) om de transparantie van de punten in de plot aan te passen:

```
ggplot(data = all_alaska_flights, aes(x = dep_delay, y = arr_delay)) +
  geom_point(alpha = 0.2)
```

De tweede manier om de over-plotting wat aan te passen is door te punten wat te **jitteren**. Met andere woorden, we voegen wat random ruis toe aan de punten om ze beter te zien en wat over-plotting weg te halen. Bij “jitteren” kun je misschien het beste denken aan een beetje



Figuur 3.3: Puntgrafiek Vertraging met alpha=0.2

schudden van de punten. In plaats van `geom_point` te gebruiken, gebruiken we `geom_jitter` om dat schudden te veroorzaken en bepalen we hoeveel we eraan toe moeten voegen met de breedte (`width`) en hoogte (`height`) argumenten. Dit hangt ook weer af van hoe hard je wilt schudden aan de horizontale en verticale variabelen (in dit geval minuten).

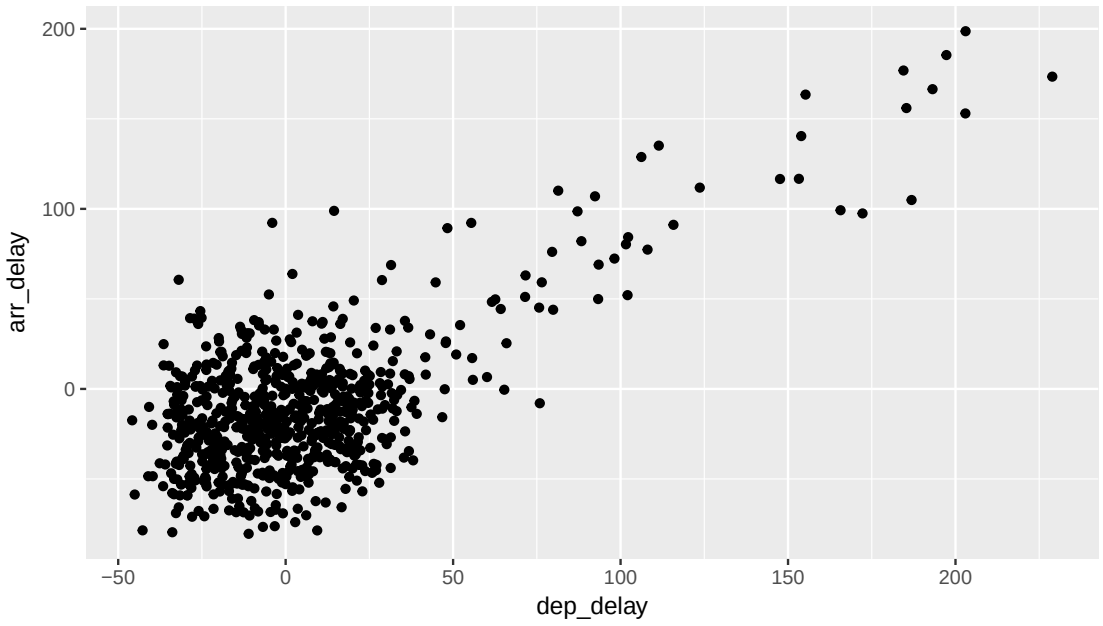
```
ggplot(data = all_alaska_flights, aes(x = dep_delay, y = arr_delay)) +
  geom_jitter(width = 30, height = 30)
```

Maar allicht is het veranderen van de transparantie van de punten door het aanpassen de `alpha`-waarde meer effectief.

3.3.3 Samenvatting

Puntengrafiek laat de relatie zien tussen twee continue variabelen en is waarschijnlijk de meest gebruikte grafiek vandaag de dag. Hij laat meteen de trend zien van een variabele ten opzichte van een andere variabele. Als je een puntengrafiek wilt maken van variabelen waarvan de ene niet continue is krijg je vreemde resultaten. Wees voorzichtig!

Met middelgrote en grote datasets wil je allicht het `geom_jitter` of het `alpha`-argument uitproberen om een goed gevoel te krijgen voor de relaties in jouw data. Dit aanpassen is vaak het aardige deel van datavisualisatie omdat je de kans hebt de relaties beter te zien als je deze subtiele veranderingen in jouw grafiek doorvoert.



Figuur 3.4: Gejitterde vertraging in de puntgrafiek

3.4 *5BG#2: Lijngrafieken*

De volgende 5BG is een lijngrafiek. Deze wordt vooral gebruikt wanneer de x-as tijd representeert en de y-as een of andere numerieke variabele; deze grafieken staan bekend als **tijd series**. Tijd representeert een variabele die is verbonden met elke dag die op de vorige dag volgt. Met andere woorden, tijd heeft een natuurlijke ordening. Lijngrafieken moeten vermeden worden wanneer er geen duidelijke opeenvolgende ordening zit in de verklarende variabele, de x-variable of de *voorspellende* variabele.

Onze nadruk ligt op de temperatuur (`temp`) variabele in dit weer (`weather`)-dataset. Door

- te kijken naar de `weather`-dataset via het intikken van `View(weather)` in de console of
- door het runnen van `?weather` in de helpfile

zien we dat de `temp`-variabele correspondeert met de temperatuur (in Fahrenheit hier) die ieder uur vastgelegd is door weerstations in de buurt van de vliegvelden van New York City. In plaats van de nadruk te leggen op alle uren in 2013 voor alle drie de luchtvelden in NYC, leggen we de nadruk op de uurtemperatuur van de luchthaven in Newark (`origin` code "EWR") voor de eerste 15 dagen in januari 2013. Het `weather`-dataframe in het `nycflights13`-pakket omvat deze data, maar we moeten de set eerst zo filteren dat het alleen die rijen omvat die corresponderen met Newark in de eerste dagen van januari 2013.

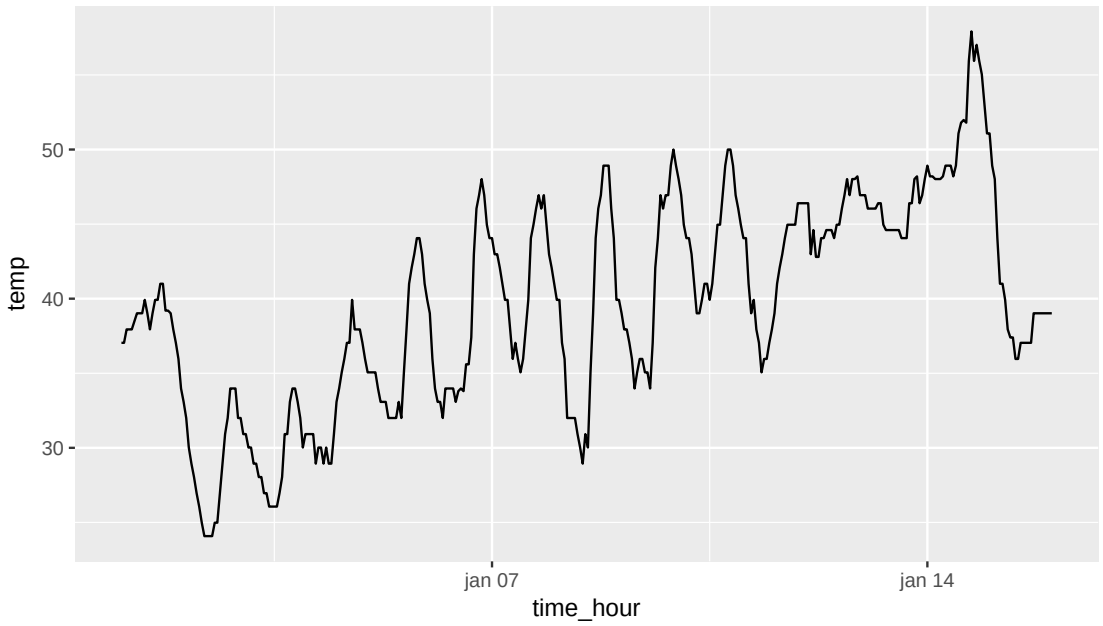
```
data(weather)
early_january_weather <- weather %>%
  filter(origin == "EWR" & month == 1 & day <= 15)
```

Dit is hetzelfde als het eerder gebruik van het `filter`-commanda in Sectie Puntgrafieken, echter nu gebruiken we het & teken. Het commando hierboven selecteert alleen die rijen in `weather` waar `origin == "EWR"` **en** `month = 1` **en** `day <= 15`.

3.4.1 Lijngrafieken via geom_line

We plotten een lijngrafiek van uurtemperatuur via het gebruik van `geom_line()`:

```
ggplot(data = early_january_weather, aes(x = time_hour, y = temp)) +
  geom_line()
```



Figuur 3.5: Uurtemperatuur in Newark voor 1-15 januari 2013

Net zoals het `ggplot()` gebruik in Sectie 3.3.1, specificeren we de componenten van de Grammatica van Grafieken:

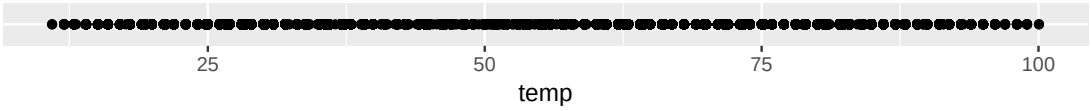
- Binnen de `ggplot()`-functie die we oproepen, specificeren we twee componenten van de grammatica:
  1. Het `data`-frame wordt op `early_january_weather` gezet door `data = early_january_weather`
  2. Het `aesthetiek` specificatie wordt op `aes(x = time_hour, y = temp)` gezet. Iets preciezer:
     - `time_hour` (bv de tijdvariabele) komt overeen met de `x` positie
     - `temp` komt overeen met de `y` positie
- We voegen een **laag** toe aan de `ggplot()`-functie door het gebruik van het `+` teken
- De laag in kwestie specificeert de derde component van de grammatica. In dit geval is het geometrisch object een lijn, door het instellen van `geom_line()`

3.4.2 *Samenvatting*

Lijngrafieken laten, net als puntgrafieken, de relaties zien tussen twee continue variabelen. Echter de variabele op de x-as (verklarende variabele) moet een natuurlijke ordening hebben, zoiets als notie van tijd. We kunnen ons publiek misleiden als dat niet het geval is.

3.5 *5BG#3: Histogrammen*

Laten we nog eens naar de temperatuur (`temp`)-variable in het weer (`weather`)-dataframe kijken. Maar nu zijn we niet zoals bij de lijngrafiek in de Sectie Lijngrafieken geinteresseerd in de relatie tussen temperatuur en tijd, maar hebben we meer oog voor de **(statistische) verdeling** van de temperatuur. We kunnen punten zetten waar op een soort lijn bepaalde waarden verschijnen:



Figuur 3.6: Gestripte grafiek van de uurtemperatuur die vastgelegd is in NYC in 2013

Dit geeft ons een algemeen idee van hoe de temperatuur (`temp`) verschilt per uur. We zien dat de temperatuur verschilt van de 11 tot 100 graden Fahrenheit. Het gebied tussen de 40 en 60 graden lijkt meer punten af te drukken dan het gebied dat er buiten ligt.

3.5.1 *Histogrammen via geom_histogram*

De **histogram** laat zien hoeveel van de elementen van een enkele numerieke variabelen binnen gespecificeerd **bakjes** vallen. In dit geval corresponderen deze **bakjes** met waarden tussen 0-10°F, 10-20°F, etc. We maken een histogram van de uurtemperaturen op alle drie NYC luchthavens in 2013:

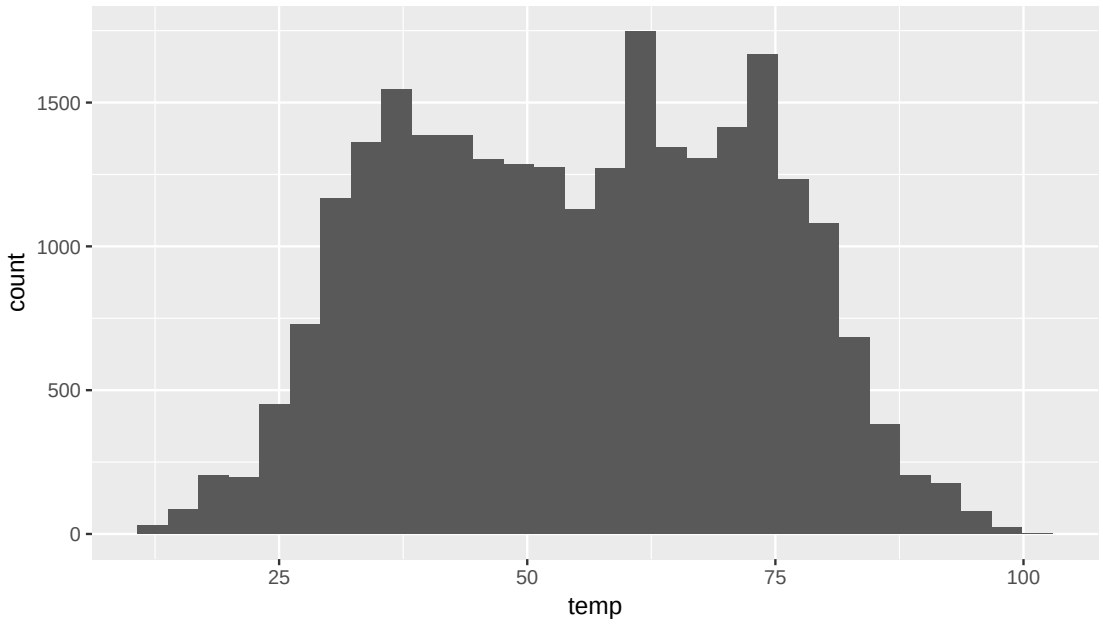
```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram()
```

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

```
## Warning: Removed 1 rows containing non-finite
## values (stat_bin).
```

Let hier op dat:

- Dat er alleen maar een variabele in kaart wordt gebracht in `aes()`: alleen maar de continue variabele temperatuur (`temp`). Jij hoeft hier niet de y-aes-thetiek in te stellen: deze wordt hier automatisch aangemaakt.



Figuur 3.7: Histogram van de uurtemperatuur van NYC in 2013

- We stellen het geometrisch object in op `geom_histogram()`
- We krijgen een waarschuwingsteken dat er een rij is verwijderd (`1 rows containing non-finite values`). Dat komt omdat er een van de waarden van temperatuur wordt gemist. R maakt er ons attent op dat dit is gebeurd.

3.5.2 *Aanpassen van de bakjes*

We kunnen het aantal/omvang van de bakjes op twee manieren aanpassen:

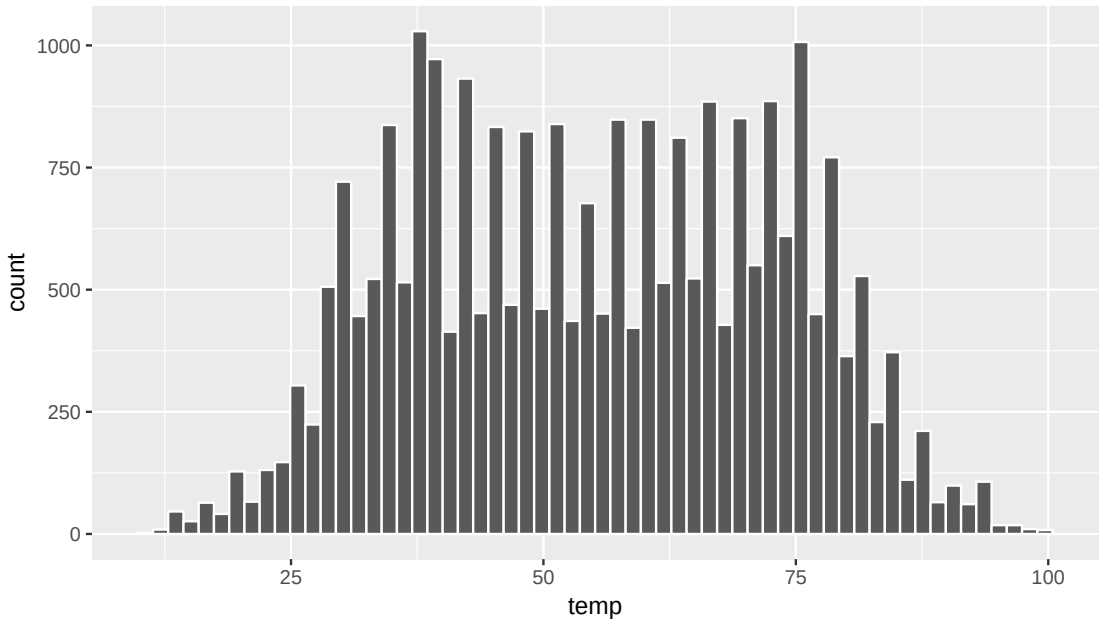
1. Door het aantal bakjes via the `bins` argument aan te passen
2. Door de breedte van de bakjes via `binwidth` argument aan te passen

We kunnen het aantal bakjes dat we willen krijgen specificeren door dit in de `geom_histogram` -functie mee te nemen. Standaard is er gekozen voor 30 maar dat is wat arbitrair; we kregen een waarschuwing boven onze plot dat dit was gebeurd.

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(bins = 60, color = "white")
```

Kijk allereerst naar de toevoeging van het kleur (`color`)-argument. Als je de bakjes duidelijk en makkelijk wilt onderscheiden van elkaar, kun je de kleur van de omheining specificeren zoals hierboven is gedaan.

Zie ook dat in plaats van het specificeren van het aantal bakjes, we de breedte van de bakjes kunnen bepalen door gebruik te maken van het `binwidth` argument in de `geom_histogram` functie.



Figuur 3.8: Histogram van de uurtemperatuur van NYC in 2013 - 60 Bins

```
ggplot(data = weather, mapping = aes(x = temp)) +
  geom_histogram(binwidth = 10, color = "white")
```

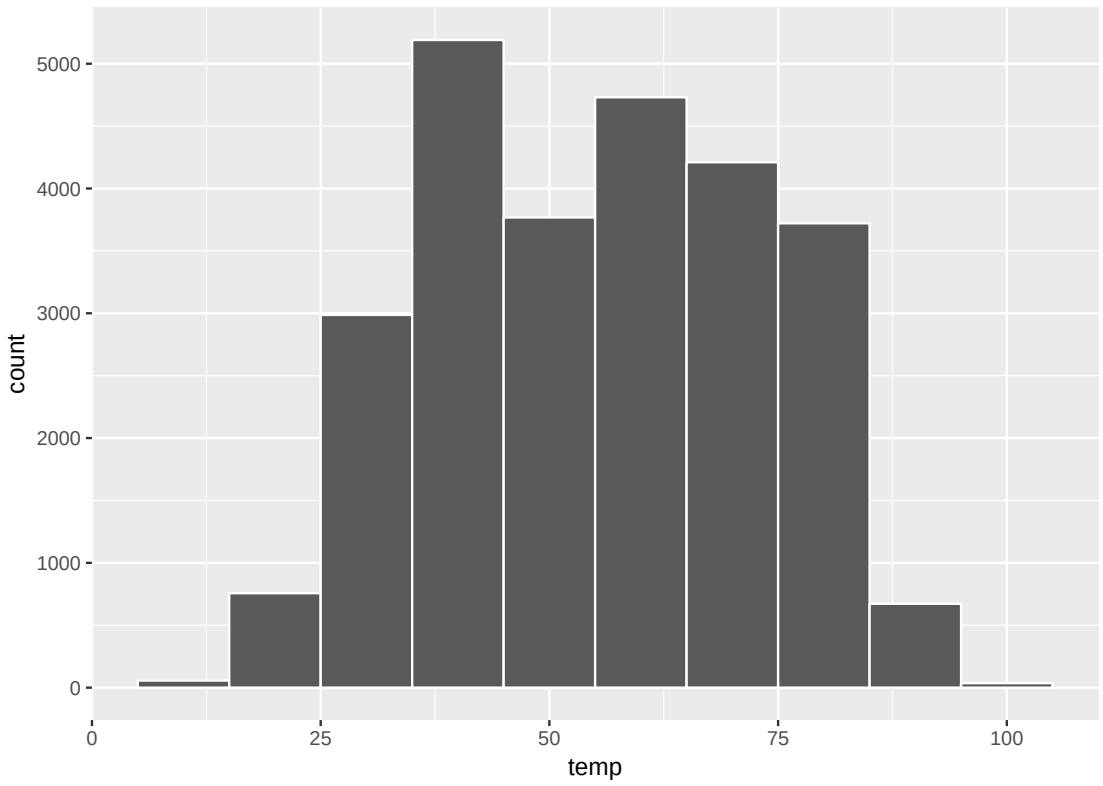
3.5.3 *Samenvatting*

In tegenstelling tot punt- en lijngrafieken geven histogrammen alleen informatie over een enkele continue variabele. Het zijn vooral visualisaties van de (statistische) verdeling van waarden.

3.6 *Facetten*

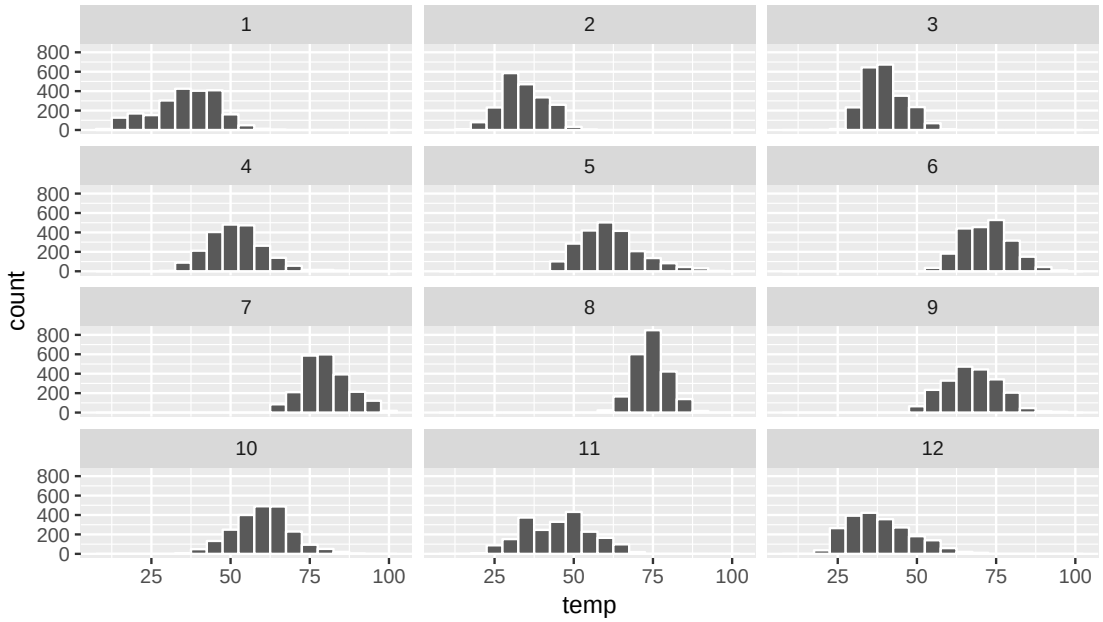
Voordat we verder gaan met de 5BG introduceren we kort een functie die **facetten** wordt genoemd. Facetten wordt gebruikt als we een klein aantal van dezelfde grafiek over een aantal categoriale variabelen willen afdrukken. Standaard hebben al deze kleine grafiekjes dezelfde verticale as.

Bijvoorbeeld, stel dat we geïnteresseerd zijn in hoe de temperatuur histogram waarover we het hadden in Sectie Histogrammen verschilt per maand. Dat is wat we noemen "de verdeling van een variabele over een andere variabele": temperatuur (`temp`) is de ene variabele en maand (`month`) is de andere variabele. Om naar de maand-histogrammen van temperatuur (`temp`) te kijken, voegen we een laag toe via `facet_wrap(~month)`. Je kunt ook het aantal rijen die je van de grafieken wilt zien benoemen door `nrow` te gebruiken in `facet_wrap`.



Figuur 3.9: Histogram van de uurtemperatuur van NYC in 2013 - Binwidth = 10

```
ggplot(data = weather, aes(x = temp)) +
  geom_histogram(binwidth = 5, color = "white") +
  facet_wrap(~ month, nrow = 4)
```



Figuur 3.10: Gefacette histogram

Zoals we mogen verwachten hangt de temperatuur samen met de seizoenen die we kunnen onderscheiden.

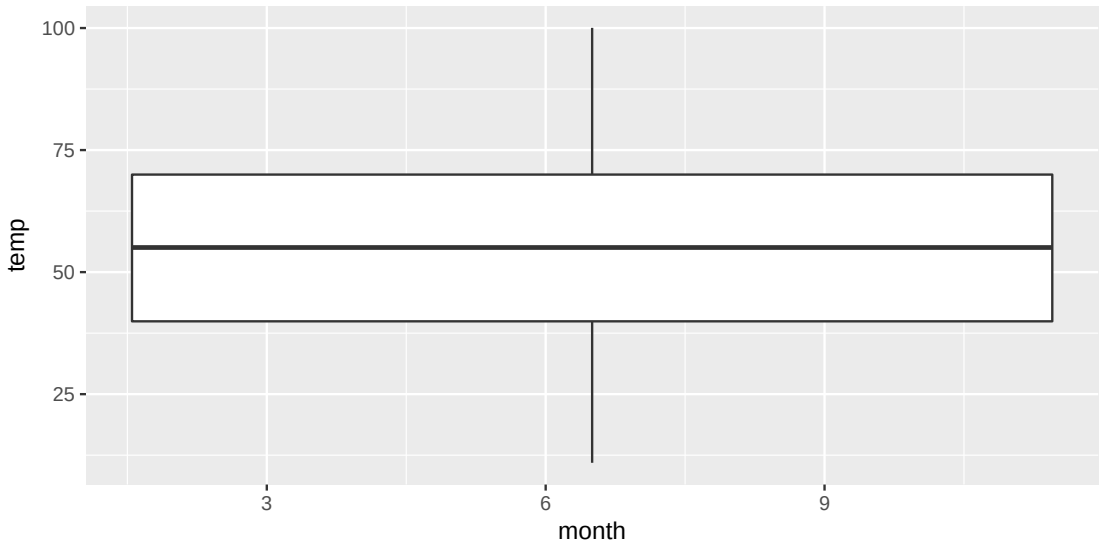
3.7 5BG#4: Doosdiagrammen

Terwijl het gebruik van gefacette histogrammen een manier kan zijn om verdelingen van een continue variabele te leveren verdeeld over groepen van een categoriale variabele zoals in Hoofdstuk Facetten, is er een alternatieve grafiek die een **doosdiagram** wordt genoemd (ook wel een **zijde-aan-zijde doosdiagram** genoemd) kan dezelfde taak uitvoeren and wordt vaak geprefereerd.

3.7.1 Doosdiagrammen via geom_boxplot

Laten we eens een doosdiagram maken om de maandelijkse temperaturen te vergelijken zoals we hierboven deden in de gefacette histogrammen.

```
ggplot(data = weather, aes(x = month, y = temp)) +
  geom_boxplot()
```



Figuur 3.11: Niet goede specificatie van doosdiagram

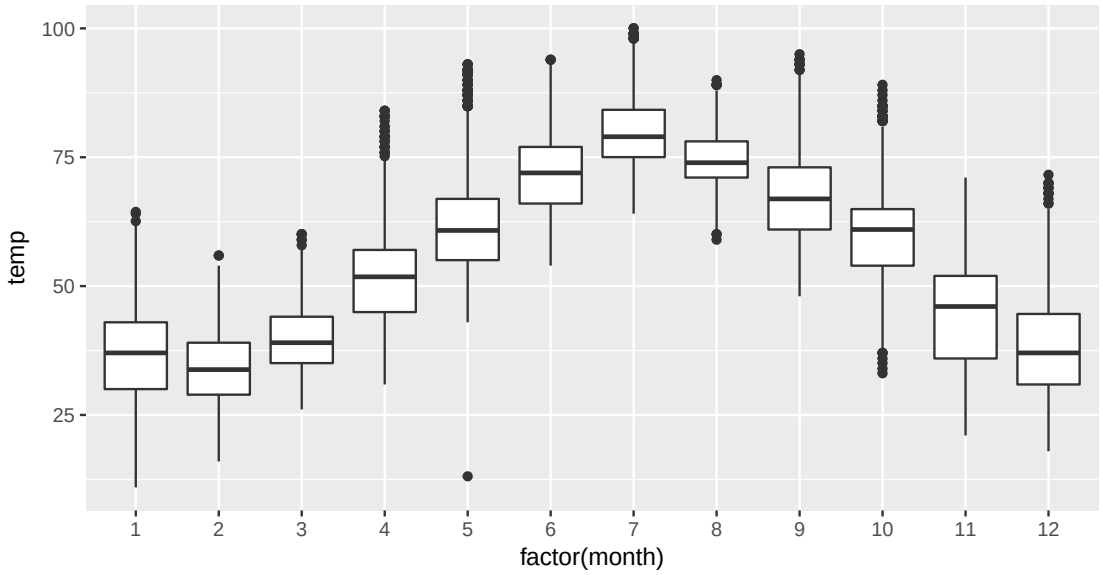
Let op de eerste waarschuwing die hier is gegeven. (De tweede heeft te maken met de missende waarden in het dataframe en die is uitgezet bij de volgende grafieken.) Merk op dat deze grafiek er niet zo uitziet zoals we verwachtten. We verwachtten de verdeling van temperaturen van iedere maand te zien (dus 12 verschillende doosdiagrammen). Dit geeft ons de samenvattende doosdiagram te zien zonder de indeling in groepen. Hier kunnen we uitkomen door een nieuwe functie te introduceren voor onze x variabele:

```
ggplot(data = weather, mapping = aes(x = factor(month), y = temp)) +
  geom_boxplot()
```

We hebben hier een nieuwe functie geintroduceerd die `factor()` wordt genoemd. Een van de dingen die deze functie doet is omzetten van een discrete waarde zoals maand (`month`) (1, 2, . . . , 12) in een categoriale variabele. Het "doos" deel van deze grafiek representeert het 25^{ste} percentiel, de mediaan (50^{ste} percentiel) en het 75^{ste} percentiel. De punten hangen samen met **uitschieters**. De lijnen laten zien hoe de data varieren die niet in het centrum van 50% liggen zoals gedefinieerd door het eerste en derde kwartiel. Langere lijnen corresponderen met meer variatie en kortere lijnen corresponderen met minder variatie.

3.7.2 Samenvatting

Doosdiagrammen bieden ons een manier om de verdeling van een kwantitatieve variabele te vergelijken en te contrasteren over verschillende niveaus van die ene categoriale variabele. Je kunt makkelijk zien waar de mediaan ligt bij de verschillende groepen door naar de middelste lijn in de doos te kijken. Je kunt ook zien hoe de spreiding per groep eruit ziet door naar de breedte van de doos te kijen en hoe ver de lijnen buiten de doos vallen. Als lijnen ver buiten de doos vallen maar de doos heeft een smalle breedte, dan is de variaties van de waarden



Figuur 3.12: Doosdiagram met maandtemperaturen

dichter bij het centrum veel smaller dan de variatie aan het uiteinde van de variabele. Tenslotte kunnen uitschieters eenvoudiger geidentificeerd worden als we naar een doosdiagram kijken dan als we naar een histogram kijken.

3.8 5BG#5: Staafdiagrammen

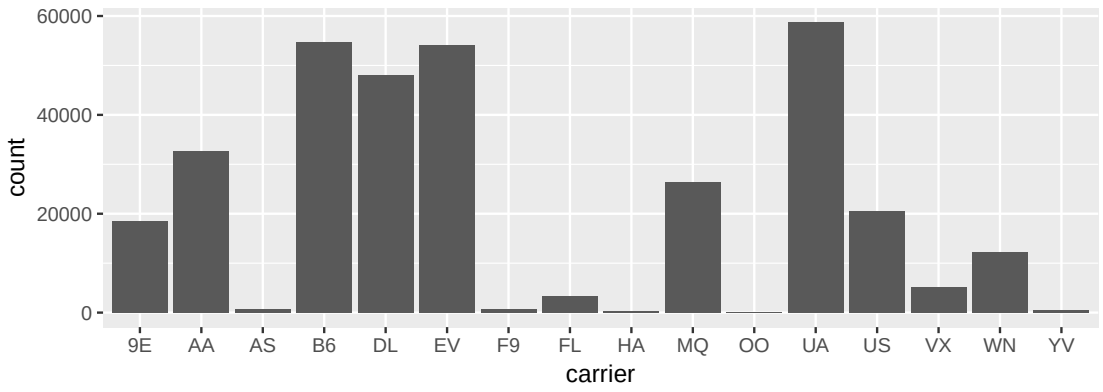
Zowel histogrammen als doosdiagrammen representeren manieren om de variatie van continue variabelen te laten zien. Maar we moeten ook de verdeling van een categoriale variabele in kaart kunnen brengen. Dat is een relatief eenvoudige taak omdat we geïnteresseerd zijn in hoeveel elementen van onze data in verschillende categorieën van de categoriale variabele vallen.

3.8.1 Staafdiagrammen via geom_bar

Vaak is de beste manier om verschillende aantallen (ook wel bekend als **frequenties**) te visualiseren via een staafdiagram. Denk aan de verdeling van de vluchten die in 2013 uit New York City vertrokken. Hier onderzochten we het aantal vluchten van iedere luchtvaartmaatschappij/`carrier`. Dit kan geplot worden door de `geom_bar`-functie in `ggplot2` te gebruiken:

```
ggplot(data = flights, mapping = aes(x = carrier)) +
  geom_bar()
```

Om een gevoel te krijgen met welke namen van deze luchtvaartmaatschappijen deze `carrier`-codes corresponderen, kunnen we kijken naar het luchtvaartmaatschappij (`airlines`)-dataframe in het `nycflights13` pakket. Gebruik hier de `kable`-functie die in het `knitr` pakket zit en die



Figuur 3.13: Aantal vluchten vertrokken van NYC in 2013 per luchtvaartmaatschappij

een aardig vormgegeven tabel van waarden produceert van de waarden in het betreffende dataframe.

```
data(airlines)
kable(airlines)
```

carrier	name
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air
OO	SkyWest Airlines Inc.
UA	United Air Lines Inc.
US	US Airways Inc.
VX	Virgin America
WN	Southwest Airlines Co.
YV	Mesa Airlines Inc.

Terug naar de staafdiagram zien we dat United Air Lines, JetBlue Airways en ExpressJet Airlines de meest vluchten hadden die in 2013 van New York City vertrokken. Om het goede aantal vluchten voor elk van deze maatschappijen te krijgen gebruiken we de aantal (`count`)-function in het `dplyr` pakket op de `carrier`-variabele in vluchten (`flights`), dat we verder introduceren in Hoofdstuk Manipulatie.

```
flights_table <- flights %>% dplyr::count(carrier)
knitr::kable(flights_table)
```

carrier	n
9E	18460
AA	32729
AS	714
B6	54635
DL	48110
EV	54173
F9	685
FL	3260
HA	342
MQ	26397
OO	32
UA	58665
US	20536
VX	5162
WN	12275
YV	601

Technische opmerking: Gebruik `::` in beide codelijnen hierboven. Dit is een andere manier om er zeker van de zijn dat de goede functie wordt opgeroepen. Een aantal (`count`) bestaat in een aantal verschillende pakketten en somst ontvang je een vreemde fout als het in een andere functie wordt gebruikt. Dit is een goede manier om R te vertellen "Ik wil deze!". Je specificeert de naam van het pakket direct voor de `::` en dan de naam van de functie direct na `::`.

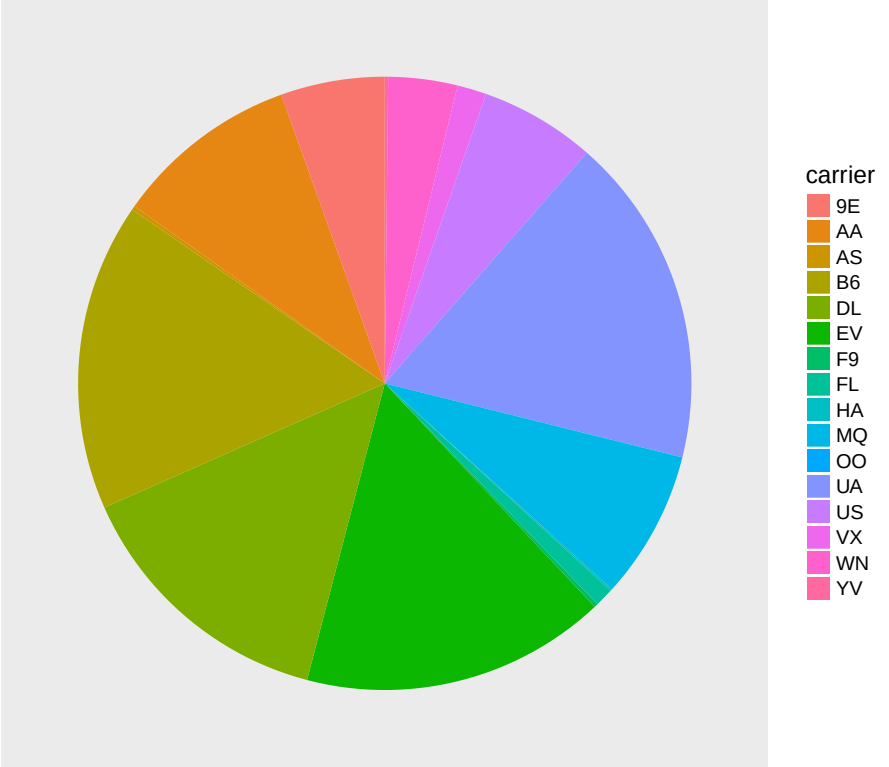
3.8.2 Voorkomen van taartgrafieken!

Jammer genoeg zien we dat de taartgrafiek een van de meest voorkomende grafieken is voor categoriale variabelen. Ook al lijkt hier niets aan de hand, ze laten een probleem zien waar wij mensen mee te maken hebben namelijk dat wij hoeken niet zo goed kunnen inschatten. Zoals Naomi Robbins in haar boek "Creating More Effective Graphs" (Robbins, 2013) beschrijft, overschatten we hoeken die groter zijn dan 90 graden en onderschatten we hoeken die kleiner zijn dan 90 graden. Met andere woorden, het is moeilijk om de relatieve omvang van een stuk taart met een ander stuk te vergelijken.

Laten we nog eens kijken naar ons vorige voorbeeld van de staafdiagram die het aantal vluchten per maatschappij verbeelde dat NYC verliet. Nu gebruiken we de taartgrafiek. Als je hier naar kijkt, probeer dan vast te stellen

- hoeveel groter het deel van de taart is voor ExpressJet Airlines (`EV`) vergeleken met US Airways (`US`),
- welke maatschappij op de derde plaats komt wat vertrekvluchten betreft, en

- hoeveel maatschappijen minder vluchten hebben dan United Airlines (`UA`).



Figuur 3.14: De gevreesde taartgrafiek

Daar waar het met staafdiagrammen vrij makkelijk is om antwoorden te geven op dit soort vragen, is het redelijk moeilijk hier met taartgrafieken antwoorden op te geven. Staafdiagrammen kunnen informatie makkelijker voor het oog verwerken. Er is een uitzondering volgens Nathan Yau van www.FlowingData.com. Het oordeel daarover laten we graag bij de lezer liggen:

3.8.3 Gebruiken van barplotten om twee variabelen te vergelijken

Staafdiagrammen zijn de manier om de frequentie van verschillende categorieen van een categoriale variabele te visualiseren. Ze maken het makkelijk om aantallen te orderen en een frequentie van de ene groep met die van andere groepen te vergelijken. Een ander gebruik van staafdiagrammen (misschien wat verwarrend) is om twee categoriale variabelen samen te vergelijken. Laten we de verdeling van uitgaande vluchten van NYC vergelijken langs luchtvaartmaatschappij (`carrier`) en luchthaven (`airport`).

We halen eerst de namen van luchthavens in NYC te voorschijn die er in de `flights`-dataset zitten. Van Hoofdstuk Opschonen weet je nog dat dit kan worden gedaan met de `inner_join` functie (en meer daarover in het volgende Hoofdstuk Manipulatie).

Figuur 3.15: De enige goede taartgrafiek

```
flights_namedports <- flights %>%
  inner_join(airports, by = c("origin" = "faa"))
```

Na het runnen van `View(flights_namedports)`, zien we nu dat de naam (`name`) correspondeert met de naam van de luchthavennaam zoals we dat zagen in de `origin` variabele. We drukken nu de grafiek af met `carrier` als de horizontale variabele. Als we `geom_bar` gebruiken, specificeert `count` het als de verticale variabele. Een nieuwe aanvulling hier is `fill = name`. Kijk even naar de grafiek om te zien wat dit argument jou levert.

Merk op dat `fill` een `aesthetiek` is net zoals `x` een `aesthetiek` is. We moeten dezelfde naam (`name`)-variabele van deze `aesthetiek`. Iedere keer als je een variabele als deze gebruikt, moet je er zeker van zijn dat deze opgenomen is in de `aes`-functie. **Dit is een veel voorkomende fout!** Laat dit tot jou doordringen zodat je later niet meer deze fout maakt.

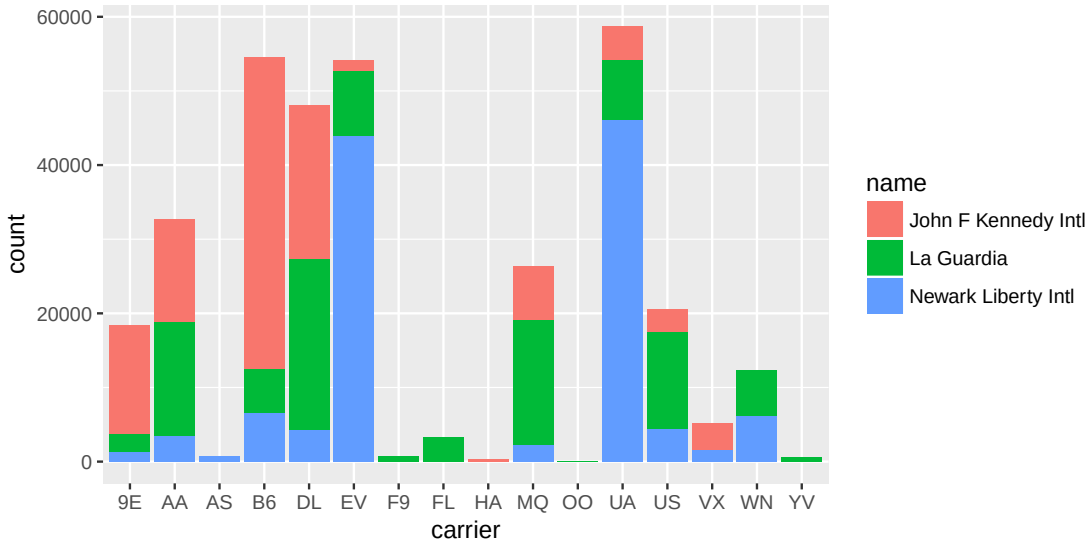
```
ggplot(data = flights_namedports, mapping = aes(x = carrier, fill = name)) +
  geom_bar()
```

Deze grafiek is wat we kennen als een **gestapelde staafdiagram**. Makkelijk te maken, maar het levert vaak problemen op.

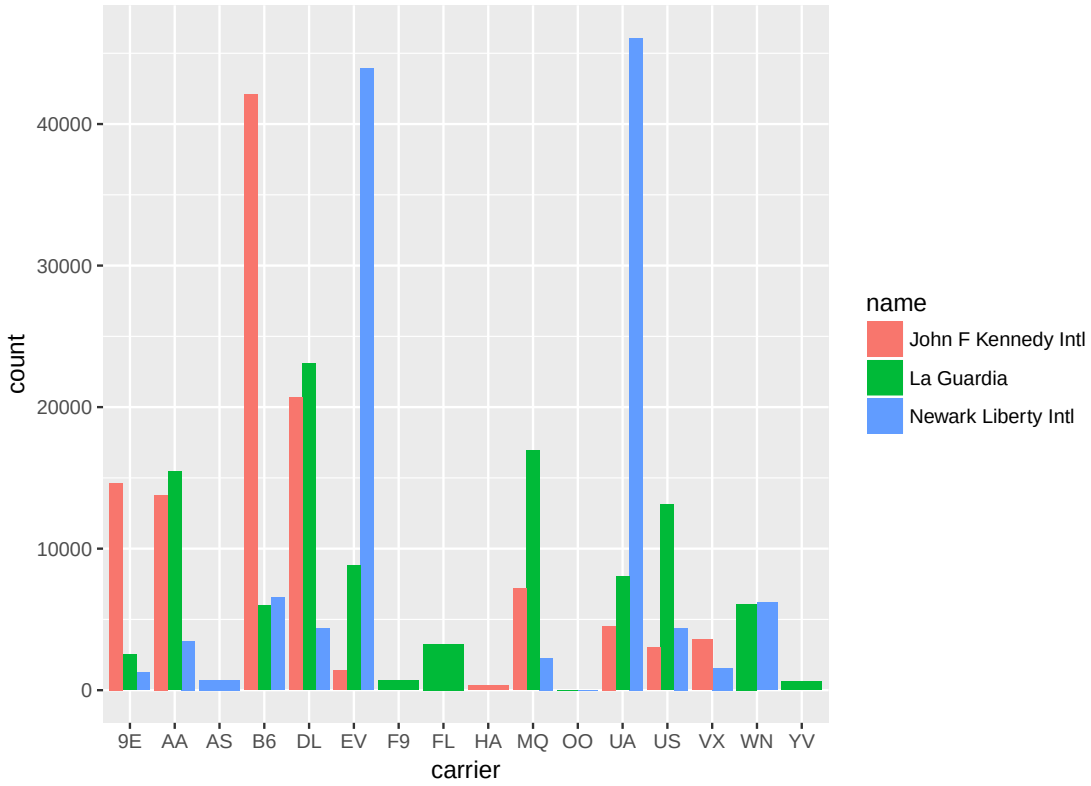
Een variant van de **gestapelde staafdiagram** is the **zijde-naast-zijde staafdiagram**.

```
ggplot(data = flights_namedports, mapping = aes(x = carrier, fill = name)) +
  geom_bar(position = "dodge")
```

Tenslotte komt de **gefacette staafdiagram** vaak voor. We zagen al eerder de gefacette of kleine, meer voorkomende grafieken in Sectie Facetten. Dit geeft ons een betere manier om de verdelingen langs zowel maatschappij (`carrier`) als luchthaven/`name` zichtbaar te maken.

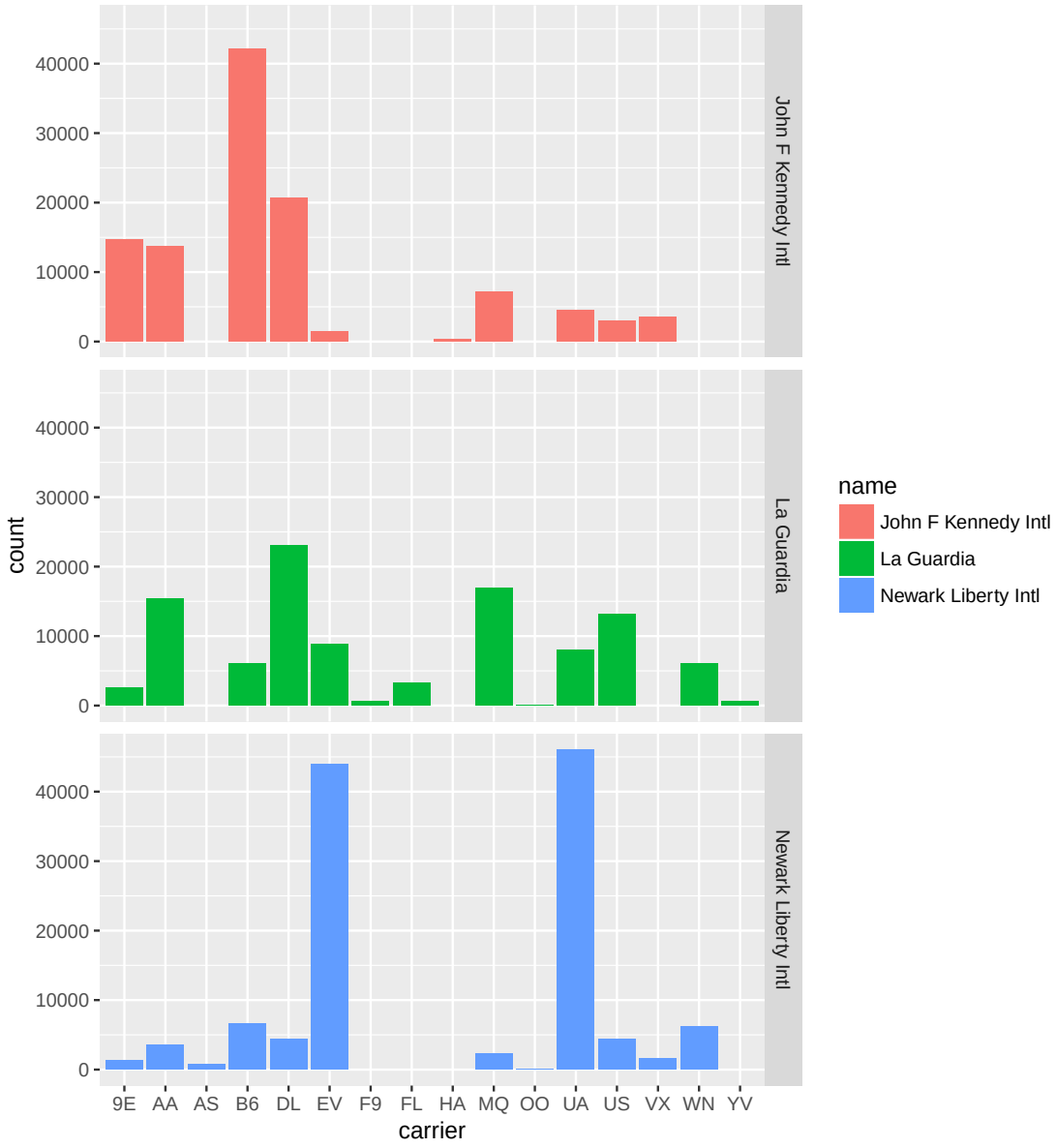


Figuur 3.16: Staafdiagram die het aantal vluchten van maatschappij en luchthaven met elkaar vergelijkt



Figuur 3.17: Zijde-naast-zijde staafdiagram die de vluchten van maatschappijen en luchthavens met elkaar vergelijkt

```
ggplot(data = flights_namedports, mapping = aes(x = carrier, fill = name)) +
  geom_bar() +
  facet_grid(name ~ .)
```



Figuur 3.18: Gefacette staafdiagram met vluchten langs maatschappij als luchthaven

Kijk naar de `facet_grid`-functieargumenten die hier geschreven zijn. We willen de namen van de luchthavens verticaal en die van de maatschappijen horizontaal. Zoals je kunt raden staat dit argument en andere van dit soort *formules* in R in `y ~ x` volgorder.

3.8.4 Samenvatting

Staafdiagrammen worden geprefereerd als het op categoriale variabelen aankomt. Ze zijn makkelijk te begrijpen en er kunnen vergelijkingen worden gemaakt tussen de groepen van een categoriale variabele. Als je te maken hebt met meer dan een categoriale variabele, kun je beter gefacette staafdiagrammen gebruiken dan zijde-over-zijde of gestapelde staafdiagrammen. Gestapelde staafdiagrammen zijn soms aardig om naar te kijken, maar het soms behoorlijk moeilijk om zaken te vergelijken boven een bepaald niveau omdat de omvang van de staven van elkaar verschillen. Zijde-over-zijde staafdiagrammen kunnen dat iets beter aan, maar het probleem van het vergelijken over groepen blijft hier bestaan.

3.9 Conclusie

3.9.1 Bronnen

Een hele goede bron om grafieken maken met het `ggplot2`-pakket is een cheatsheet die RStudio samen heeft gesteld onder de titel "Datavisualisatie met ggplot2" en beschikbaar door

- te klikken hier of
- te klikken op de RStudio Menu Bar -> Help -> Cheatsheets -> "Data Visualization with `ggplot2`"

Dit omvat meer dan we in dit hoofdstuk hebben besproken en laat mooie visuele beschrijvingen zien van wat elke functie doet.

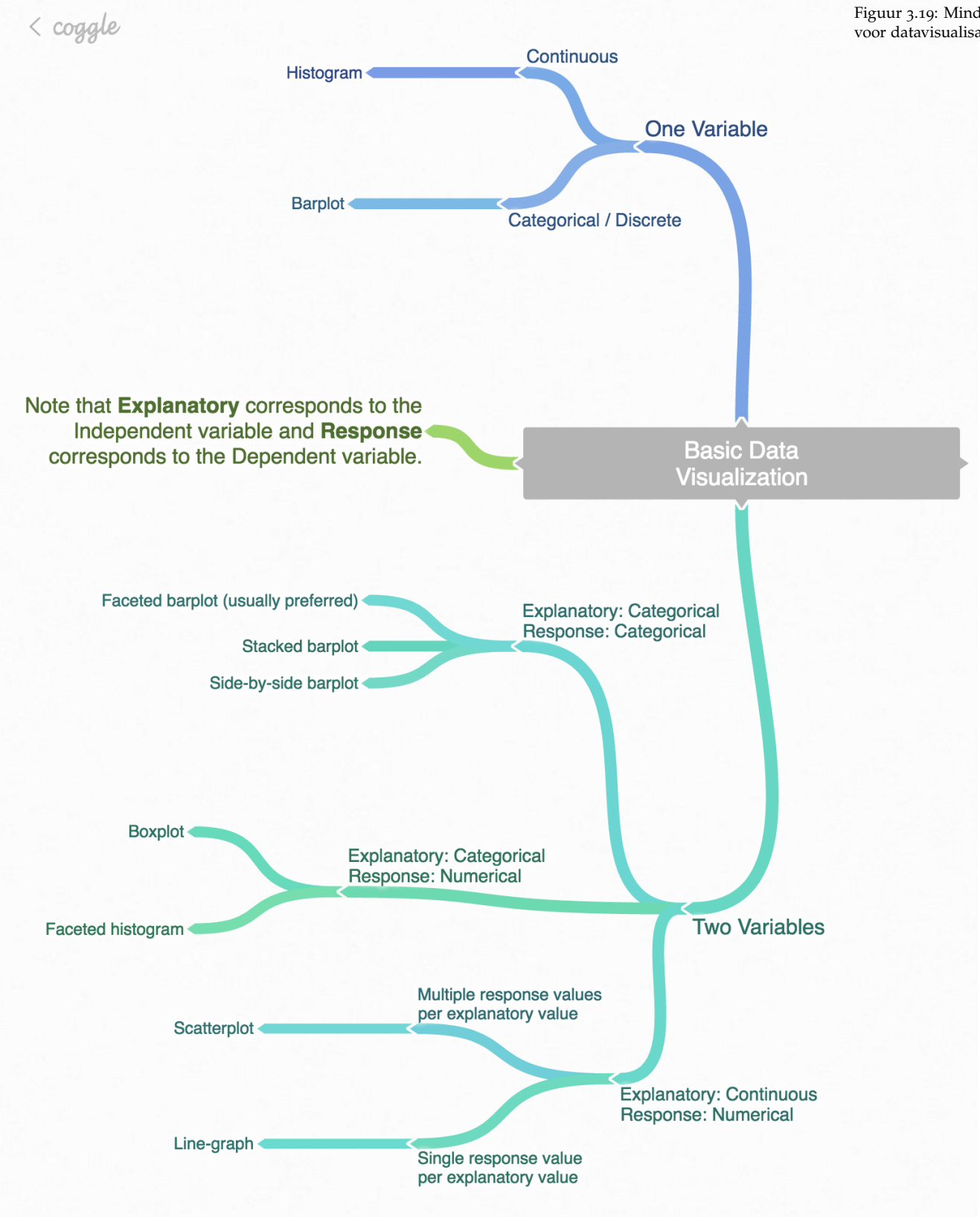
In aanvulling hebben we een mindmap gemaakt om jou eraan te herinneren welke typen grafieken het meest geschikt zijn in een bepaalde situatie afhankelijk van de typen variabelen waar je in het probleem mee te maken hebt. Het is beschikbaar hier en hieronder.

3.9.2 Script van R code

Een R-scriptfile of alle R codes die in dit hoofdstuk zijn gebruikt vind je hier.

Vragen over het `fivethirtyeight` R-pakket (**?**) met linken naar de hiermee corresponderende FiveThirtyEight.com artikelen in onze gratis DataCamp cursus **Effectief Dataverhalenvertellen Storytelling via het gebruik van `tidyverse`**. Het materiaal in dit hoofdstuk wordt gedekt voor de hoofdstukken van de DataCamp cursus die hieronder beschikbaar is:

- Punt- & Lijngrafieken
- Histogrammen & Doosdiagrammen
- Staafdiagrammen
- Een **ggplot2** DataCamp cursus is op dit moment in ontwikkeling.



Figuur 3.19: Mindmap voor datavisualisatie

3.9.3 Wat komt er nu?

In het volgende Hoofdstuk Manipulatie onderzoeken we de data verder door de data te groeperen, samenvattingen te maken gebaseerd op deze groepering, filteren van onze data zodat ze aan onze voorwaarden voldoen en andere manipulaties van onze data, inclusief het definiëren van nieuwe kolommen/variabelen. Deze procedures van datamanipulatie gaan goed hand-in-hand met de procedures van datavisualisatie waar we het in dit hoofdstuk over hadden.

4
Data Manipulatie via dplyr

Laten we even kort herhalen wat we hebben gedaan en waar we naar toe gaan. In Hoofdstuk Opschonen, bespraken we wat het voor data betekent om opgeschoond te zijn. We zagen dat dit refereerde naar observationele eenheden die corresponderen met rijen en variabelen die in kolommen zijn opgeslagen (een variabele voor elke kolom). De entries in de data-frame corresponderen met verschillende combinaties van observationele eenheden en variabelen. In het `flights` dataframe zien we bijvoorbeeld dat elke rij correspondeert met een bepaalde vlucht vanaf New York City. Met andere woorden, de observationele eenheid van dat opgeschoonde data-frame is een vlucht. De variabelen zijn als kolommen weergegeven en voor `flights` houden ze wel kwantitatieve variabelen in als `dep_delay` en `distance` maar ook categoriale variabelen als `carrier` en `origin`. De entry in de tabel correspondeert met een bepaalde vlucht op een bepaalde dag en een bepaalde waarde van een bepaalde variabele representeert die vlucht.

In het Hoofdstuk Visualisatie zagen we dat het organiseren van data op deze opgeschoonde wijze het ons makkelijker maakt om grafieken te maken. We kunnen dan simpel specificeren welke variabele we op een bepaalde as willen, welke variabele we op de andere as willen en wat voor een soort grafiek we willen maken. We kunnen ook dingen doen als het veranderen van de kleur door een andere variabele of de omvang van punten veranderen volgens een andere variabele in de opgeschoonde dataset. Verder wezen we in Hoofdstuk Visualisatie op enkele manieren om data samen te vatten en te manipuleren om te voldoen aan wat we willen.

Dit hoofdstuk Manipuleren gaat daar verder op door. We gebruiken een aantal voorbeelden van wat we hier maar even noemen *Vijf Belangrijke Woorden* in het `dplyr` pakket. Er zijn veel meer geavanceerde handelingen dan enkel deze vijf. Aan het einde van het hoofdstuk zul je enkele voorbeelden daarvan zien.

Op verschillende momenten wijzen we op het gebruik van het `View()`-commando waarmee een bepaald dataframe is te onderzoeken. Voel je vrij om dat commando waar dan ook te gebruiken. In feite moet je de gewoonte ontwikkelen om dat voor *elke* data-frame te doen waarmee je ook werkt.

Pakketten die nodig zijn

Voordat we verder gaan met dit hoofdstuk, laten we eerst eens de benodigde pakketten zien (wel eerst binnenhalen natuurlijk).

```
library(dplyr)
library(ggplot2)
library(nycflights13)
library(knitr)
```

4.1 *De pijp* `%>%`

Voordat we de vijf belangrijke woorden introduceren, introduceren we eerst de pijp-functie (`%>%`). Net als het + teken dat werd gebruikt om lagen toe te voegen aan een plot wanneer je `ggplot()` gebruikt, stelt de pijpfunctie ons in staat om `dplyr` datamanipulatie-functies samen te voegen. Deze functie kan worden gelezen als "*dan*". De `%>%` functie stelt ons in staat om makkelijk een stap verder te gaan in `dplyr` zodat we bijvoorbeeld:

- `filter` , ons data frame kunnen filteren door op enkele rijen te richten en *dan*
- `group_by`, een andere variabele kunnen maken om groepen te vormen en *dan*
- `summarize`, deze data groeperen om het gemiddelde voor elk niveau van de groep te bepalen.

Op de pijpsyntax zal de nadruk liggen in de rest van het boek en je zult zien dat je snel hieraan verslaafd raakt. Als je meer voorbeelden wilt zien in het gebruik van `dplyr` kun je er Hoofdstuk 5 van Hadley en Garrett's boek erop naslaan (Hadley & Garrett, 2016).

4.2 *Vijf belangrijkste woorden - 5BW*

De `d` in `dplyr` staat voor data-frames. De functies hier werken als je met objecten van het dataframe werkt. Het is voor jou vooral van belang om je te richten op de 5BW (vijf belangrijkste woorden): de vijf meest voorkomende functies die jou helpen bij het manipuleren en samenvatten van data. Een beschrijving van wat deze woorden inhouden volgt. De belangrijkste functies zijn

- `filter`: Haalt er rijen uit gebaseerdd op de voorwaarden van hun waarden;
- `summarize`: Maakt een samenvatting van de variabelen
  - over de hele data-frame
  - of over groepen van observaties van variabelen door `group_by` te gebruiken;
- `mutate`: Maakt een nieuwe variabele in de data-frame door de bestaande te muteren;
- `arrange`: Arrangeren/sorteren van de rijen gebaseerd op een of meer variabelen.

Net als in het vorige hoofdstuk waarin we het hadden over de 5BG (de Vijf Benoemde Grafieken in Hoofdstuk Visualisatie via het gebruik van `ggplot2`) voor datavisualisatie, hebben we het hier over de 5BW (de Vijf Belangrijke Woorden in `dplyr`) voor datamanipulatie. Elk van de 5BW's volgt dezelfde syntax met hetzelfde argument voor de pijp `%>%`, wat de naam is van het dataframe en dan naar van het woord van andere argumenten die specificeren met welke criteria je het woord dat tussen haakjes staat wilt laten werken.

4.2.1 5BW#1: Filter observaties door het gebruik van 'filter'

Figuur 4.1: Filterdiagram van Data Wrangling met de dplyr- en tidyr-cheatsheet

De `filter` functie hier werkt vergelijkbaar als de "Filter" -optie in Microsoft Excel; het stelt jou in staat de criteria van de variabelewaarden in jou dataset te specificeren en dan alleen die rijen te kiezen die met die criteria matchen. We beginnen met ons alleen te richten op vluchten van New York City naar Portland, Oregon. De `dest` code (aankomst of te wel luchthavencode) van Portland, Oregon is `"PDX"`. Run het volgende en kijk naar de resultaten in de spreadsheet om jou ervan te overtuigen dat alleen de vluchten naar Portland zijn gekozen:

```
portland_flights <- flights %>%
  filter(dest == "PDX")
View(pdx_flights)
```

Let op het volgende:

- De ordening van de commando's:
  - Neem het data-frame `flights` *dan*
  - `filter` het data-frame zo dat alleen de `dest` met `"PDX"` erin zitten.
- Het dubbele gelijkteken ==. Je maakt makkelijk de fout dat je alleen het enkelvoudige =teken gebruikt. Laten we eens zien welke fout we dan maken:

```
portland_flights <- flights %>%
  filter(dest = "PDX")
```

```
Error: filter() takes unnamed arguments. Do you need '=='?
```

Je kunt ook verschillende criteria tegelijk gebruiken:

- `|` correspondeert met "of"
- `&` correspondeert met "en"

We kunnen het gebruik van `&` ook vaak voor lief laten en tussen de voorwaarden een komma plaatsen. Dat zul je in het voorbeeld hieronder zien.

Ter aanvulling, je kunt ook andere mathematische tekens gebruiken (vergelijkbaar met ==):

- `>` correspondeert met "groter dan"
- `<` correspondeert met "kleiner dan"
- `>=` correspondeert met "groter of gelijk aan"
- `<=` correspondeert met "kleiner of gelijk aan"
- `!=` correspondeert met "niet gelijk aan"

Om al deze tekens in actie te zien selecteren we alle vluchten die de JFK-luchthaven hebben verlaten richting Burlington, Vermont (`"BTV"`) of Seattle, Washington (`"SEA"`) in de maanden october, novembe of december. Run het volgende

```
btv_sea_flights_fall <- flights %>%
  filter(origin == "JFK", (dest == "BTV" | dest == "SEA"), month >= 10)
View(btv_sea_flights_fall)
```

Let op dat je misschien wel praat over "alle vluchten naar Burlington, Vermont *en* Seattle, Washington", maar in termen van computerhandelingen bedoelen we echt "alle vluchten naar Burlington, Vermont *of* Seattle, Washington", omdat voor de gegeven rij de dataset, `dest`, het: "BTV", "SEA" of iets anders is, maar niet "BTV" en "SEA" op hetzelfde moment.

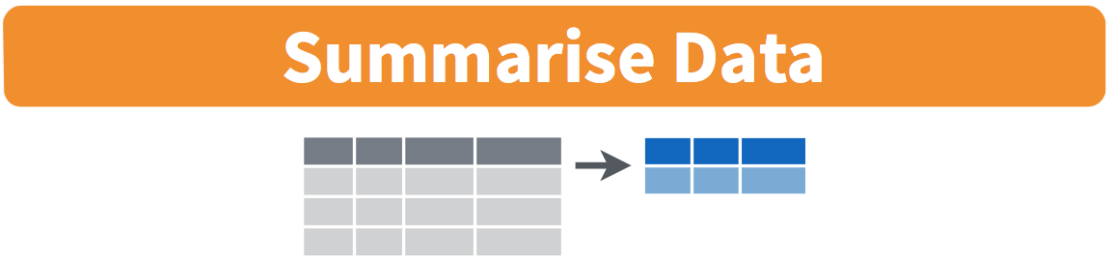
Een ander voorbeeld gebruikt `!` om er rijen uit te pikken **DIE NIET** matchen met een voorwaarde. Hieronder selecteren we rijen die corresponderen met vluchten die niet naar Burlington, VT of Seattle, WA gaan.

```
not_BTV_SEA <- flights %>%
  filter(!(dest == "BTV" | dest == "SEA"))
View(not_BTV_SEA)
```

Een laatste opmerking waar we op willen wijzen is dat `filter()` niet als eerste woord moet worden gebruikt dat je toepast op jouw data. Dit schoont jouw dataset op naar de rijen waar je zorg voor draagt of, om het anders te zeggen, het vernauwt de scope naar een bepaalde groep observationele eenheden.

4.2.2 5BW#2: Samenvatten van variabelen door het gebruik van 'summarize'

We kunnen de standard deviatie en het gemiddelde van de temperatuurvariabele `temp` uitrekenen in het `weather` data-frame van `nycflights13` door het gebruik van de `summarize`-functie in `dplyr`:



Figuur 4.2: Samenvatdiagram van Data Wrangling met dplyr en tidyr-cheatsheet

Figuur 4.3: Een ander samenvatdiagram van Data Wrangling met dplyr en tidyr-cheatsheet

```
summary_temp <- weather %>%
  summarize(mean = mean(temp), std_dev = sd(temp))
kable(summary_temp)
```

mean	std_dev
NA	NaN

We hebben een klein data-frame gemaakt dat we `summary_temp` noemen en dat zowel het gemiddelde (`mean`) en de standaardeviatie (`std_dev`) van de `temp` variabele omvat in `weather`. Let op, zoals we dat eerder zagen, dat het data-frame `weather` van veel rijen naar een enkele rij ging met alleen maar samenvattende waarden in het data-frame `summary_temp`. Maar waarom missen dan de gemiddelden en de standaarddeviatie, zoals we zien aan `NA`? Zoals je je misschien herinnert slaan de `mean` en `sd` functies de missende waarden niet over. We moeten het argument `na.rm=TRUE` benoemen (`rm` is een afkorten van "remove", "weghalen"):

```
summary_temp <- weather %>%
  summarize(mean = mean(temp, na.rm = TRUE), std_dev = sd(temp, na.rm = TRUE))
kable(summary_temp)
```

mean	std_dev
55.20351	17.78212

Als we direct toegang willen hebben tot een van deze waarden kunnen we het `$`-teken gebruiken om een kolom in het data-frame te specificeren. Bij voorbeeld:

```
summary_temp$mean
```

```
## [1] 55.20351
```

Je komt vaak zaken met missende waarden `NA` tegen. In feite is er een heel veld van de statistiek dat zich met missende waarden bezig houdt. Echter het is niet aan te raden om `na.rm = TRUE` standaard in jouw samenvattende codes op te nemen; je moet proberen het te runnen zonder dit argument. Het idee hierachter is dat je er in ieder geval alert bent op de aanwezigheid van missende waarden en dat je door hebt wat de impact op de analyse is als je deze waarden niet meeneemt. Met andere woorden, `na.rm = TRUE` moet alleen gebruikt worden waar dat nodig is.

Welke andere samenvattende functies kunnen we gebruiken binnen het `summarize()` woord? Elke functie in R heeft een vector van waarden en geeft er slechts ??n terug. Hier zijn er een paar:

- `min()` en `max()`: minimum- en maximumwaarden respectievelijk
- `IQR()`: interkwartielbereik
- `sum()`: de som
- `n()`: het aantal rijen/observaties in elke groep. Deze samenvattende functie komt meer tot zijn recht in het `group_by` hoofdstuk.

4.2.3 5BW#3: Groepen op rij door het gebruik van 'group_by'

Figuur 4.4: Groepen volgens en samenvattende-diagram van Data Wrangling met dplyr en tidyr cheatsheet

Echter het is vaak makkelijker om een variabele samen te vatten gebaseerd op het groeperen van een andere variabele. Laten we zeggen dat net als in de vorige paragraaf we geinteresseerd zijn in het gemiddelde en de de standaard deviatie van temperaturen maar *gegroepeerd volgens maand*. Dit concept kan net zo goed worden uitgedrukt als: we willen het gemiddelde en standaarddeviatie van temperaturen

1. verdeeld over maanden.
2. versneden over maanden.
3. geaggregeerd over maanden.
4. in maanden uiteengevallen.

We denken dat je verrast zult zijn hoe eenvoudig dit is. Run de volgende code:

```
summary_monthly_temp <- weather %>%
  group_by(month) %>%
  summarize(mean = mean(temp, na.rm = TRUE),
            std_dev = sd(temp, na.rm = TRUE))
kable(summary_monthly_temp)
```

month	mean	std_dev
1	35.64127	10.185459
2	34.15454	6.940228
3	39.81404	6.224948
4	51.67094	8.785250
5	61.59185	9.608687
6	72.14500	7.603357
7	80.00967	7.147631
8	74.40495	5.171365
9	67.42582	8.475824
10	60.03305	8.829652
11	45.10893	10.502249
12	38.36811	9.940822

Deze code is identieek aan de vorige code die `summary_temp`vormde, maar er is een extra `group_by(month)` opsplitsing tussengeplaatst. Door simpel de `weather` dataset eerst te groeperen volgens maand (`month`) en dan dit nieuwe dataframe samen te vatten (`summarize`) krijgen we een data-frame dat het gemiddelde en de standaarddeviatie laat zien voor elke maand in New York City. Omdat elke rij in `summary_monthly_temp` een samenvatting representeert van verschillende rijen in `weather`, zijn de observationele eenheden veranderd.

Het is belangrijk op te merken dat het `group_by` -commando niet het data-frame aanpast. Het maakt als het ware *meta-data* (data over de data) aan, gespecificeerd voor de groepsstructuur van de data. Alleen nadat we de `summarize` functie hebben toegevoegd verandert het data-frame. Als we deze groepsstructuur willen weghalen, kunnen we de `ungroup()` functie toepassen.

We komen nu terug bij de `n()` samenvattende telfunctie terug die we in het vorige hoofdstuk hebben geintroduceerd. Bijvoorbeeld, stel dat we een gevoel willen krijgen van hoeveel vluchten er van de drie luchthavens van New York City vertrekken:

```
by_origin <- flights %>%
  group_by(origin) %>%
  summarize(count = n())
kable(by_origin)
```

origin	count
EWR	120835
JFK	111279
LGA	104662

We zien dat van Newark (`"EWR"`) de meeste vluchten in 2013 vertrokken, gevolgd door `"JFK"` en tot slot door LaGuardia (`"LGA"`). Let op dat er een subtiel maar ook belangrijk verschil is tussen de functies `sum()` en `n()`. Terwijl `sum()` optelt tot een groot aantal, telt `n()` het aantal keren per bepaalde waarde.

Je bent niet beperkt in het groeperen volgens een variabele! Stel dat je het aantal vluchten wilt weten dat elk van de drie luchthavens van NYC *iedere maand* verlaat. In dat geval kunnen we langs een tweede variabele groeperen `month`: `group_by(origin, month)`.

```
by_monthly_origin <- flights %>%
  group_by(origin, month) %>%
  summarize(count = n())
kable(by_monthly_origin)
```

4.2.4 *5BW#4: Maken van nieuwe variabelen/veranderen van oude variabelen door het gebruik van 'mutate'*

Figuur 4.5: Mutate diagram from Data Wrangling with dplyr and tidyr cheatsheet

Als we naar de `flights` dataset kijken zijn er enkele duidelijke aanvullende variabelen die kunnen worden berekend gebaseerd op de waarden van de variabelen die al in de dataset zitten. Passagiers zijn vaak gefrustreerd als hun vluchten laat vertrekken maar hun gemoed verandert enigzins als piloten erin slagen om wat van de vliegtijd af te snoepen en op hun plaats van bestemming aankomen in de buurt van de verwachte aankomsttijd. Dat wordt vaak gezien als "winst" en we zullen deze variabele maken via het gebruik van de `mutate` functie. Let op dat we het `flights` dataframe hebben overschreven met wat het hiervoor was en met de aanvullende variabele winst (`gain`) hier.

```
flights <- flights %>%
  mutate(gain = arr_delay - dep_delay)
```

Waarom overschrijven we `flights` in plaats van het nieuwe dataframe als een nieuw object opslaan, zoiets als `flights_with_gain`? Als een grove regel geldt, dat als je niet informatie

verliest die je later mogelijk nodig hebt, het acceptabel is om de dataframes te overschrijven. Echter als je bestaande variabele overschrijft en bestaande variabelen en/of observationele eenheden verandert, wordt het moeilijk om de orginele informatie weer boven tafel te krijgen. In dat geval is het allicht beter een nieuw dataobject te maken.

Laten we eens naar de samenvattende uitkomsten van deze `gain`-variabele kijken en deze zelfs plotten in de vorm van een histogram:

```
gain_summary <- flights %>%
  summarize(
    min = min(gain, na.rm = TRUE),
    q1 = quantile(gain, 0.25, na.rm = TRUE),
    median = quantile(gain, 0.5, na.rm = TRUE),
    q3 = quantile(gain, 0.75, na.rm = TRUE),
    max = max(gain, na.rm = TRUE),
    mean = mean(gain, na.rm = TRUE),
    sd = sd(gain, na.rm = TRUE),
    missing = sum(is.na(gain))
  )
kable(gain_summary)
```

min	q1	median	q3	max	mean	sd	missing
-109	-17	-7	3	196	-5.659779	18.04365	9430

We hebben de samenvattingfunctie (`summary`) aangepast als we deze vergelijken met de functie in Hoofdstuk Visualisatie door de `summarize`-functie van `dplyr`pakket te gebruiken.

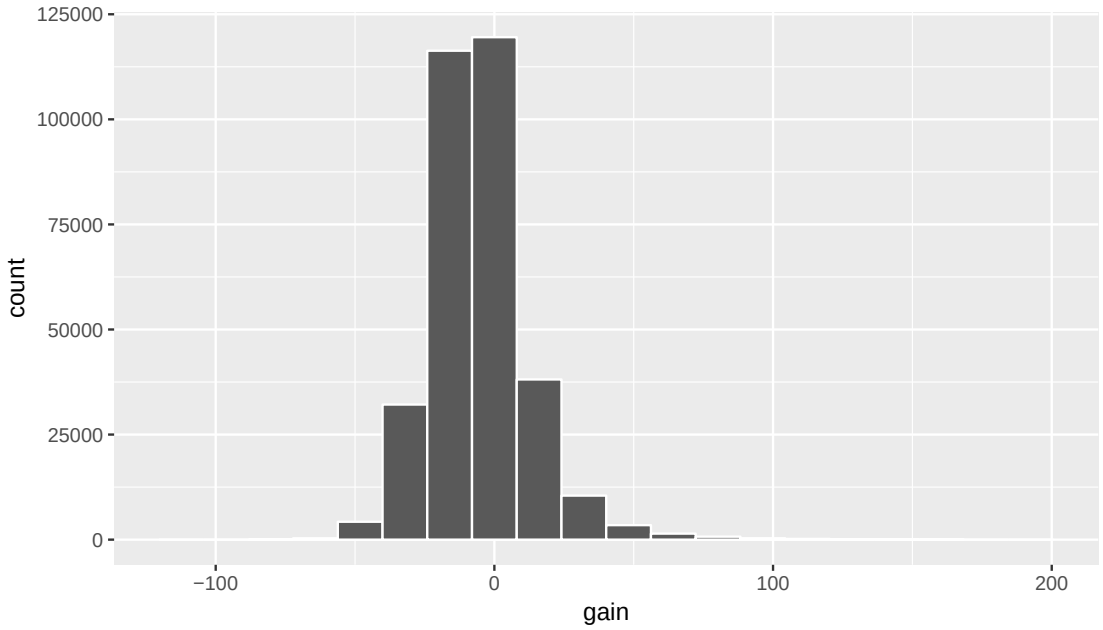
```
ggplot(data = flights, mapping = aes(x = gain)) +
  geom_histogram(color = "white", bins = 20)
```

We kunnen ook een aantal kolommen tegelijkertijd maken en zelfs naar deze deze kolommen refereren die net in een nieuwe kolom zijn gezet. In Hadley Wickhams Hoofdstuk 5 van "R for Data Science" (Wickham, 2016) zien we hier een voorbeeld van:

```
flights <- flights %>%
  mutate(
    gain = arr_delay - dep_delay,
    hours = air_time / 60,
    gain_per_hour = gain / hours
  )
```

4.2.5 5BW#5: Herordenen van het data-frame door het gebruik van 'arrange'

Misschien heb je tijdens het werken met de dataframes in dit boek tot nu toe zelf al gedacht, dat een van de meest voorkomende zaken die je moet doen is het sorteren van een dataframe via een specifieke variabele in een kolom. Is jou wel eens gevraagd om een mediaan met de



Figuur 4.6: Histogram of gain variable

hand uit te rekenen? Dit vraagt van jou om de waarden van de data in de volgorde van klein naar groot te zetten. Het `dplyr`-pakket heeft een functie die arrangeren (`arrange`) wordt genoemd dat we zullen gebruiken voor het sorteren/heroderen van de data volgens de waarden van de gespecificeerde variabele. Dit wordt vaak toegepast nadat we de functies `group_by` en `summarize`-functie hebben gebruikt, zoals we zullen zien.

Stel dat we geinteresseerd zijn in het vaststellen van de vaakst voorkomende bestemmings luchthavens van New York City in 2013:

```
freq_dest <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n())
freq_dest
```

```
## # A tibble: 105 x 2
##     dest num_flights
##    <chr>       <int>
##  1   ABQ         254
##  2   ACK         265
##  3   ALB         439
##  4   ANC           8
##  5   ATL       17215
##  6   AUS        2439
##  7   AVL         275
##  8   BDL         443
```

```
##  9   BGR         375
## 10   BHM         297
## # ... with 95 more rows
```

Standaard worden de waarden van de bestemming (`dest`) in alfabetische volgorde geplaatst. We zijn geinteresseerd in de luchthavens die het meest voorkomen:

```
freq_dest %>% arrange(num_flights)
```

```
## # A tibble: 105 x 2
##     dest num_flights
##    <chr>       <int>
##  1   LEX           1
##  2   LGA           1
##  3   ANC           8
##  4   SBN          10
##  5   HDN          15
##  6   MTJ          15
##  7   EYW          17
##  8   PSP          19
##  9   JAC          25
## 10   BZN          36
## # ... with 95 more rows
```

Dit geeft het tegendeel van waarin we geinteresseerd zijn; het geeft ons de minst voorkomende bestemmingsluchthavens eerst. Om de volgorde van toenemend naar afnemend te veranderen gebruiken we de `desc` functie:

```
freq_dest %>% arrange(desc(num_flights))
```

```
## # A tibble: 105 x 2
##     dest num_flights
##    <chr>       <int>
##  1   ORD       17283
##  2   ATL       17215
##  3   LAX       16174
##  4   BOS       15508
##  5   MCO       14082
##  6   CLT       14064
##  7   SFO       13331
##  8   FLL       12055
##  9   MIA       11728
## 10   DCA        9705
## # ... with 95 more rows
```

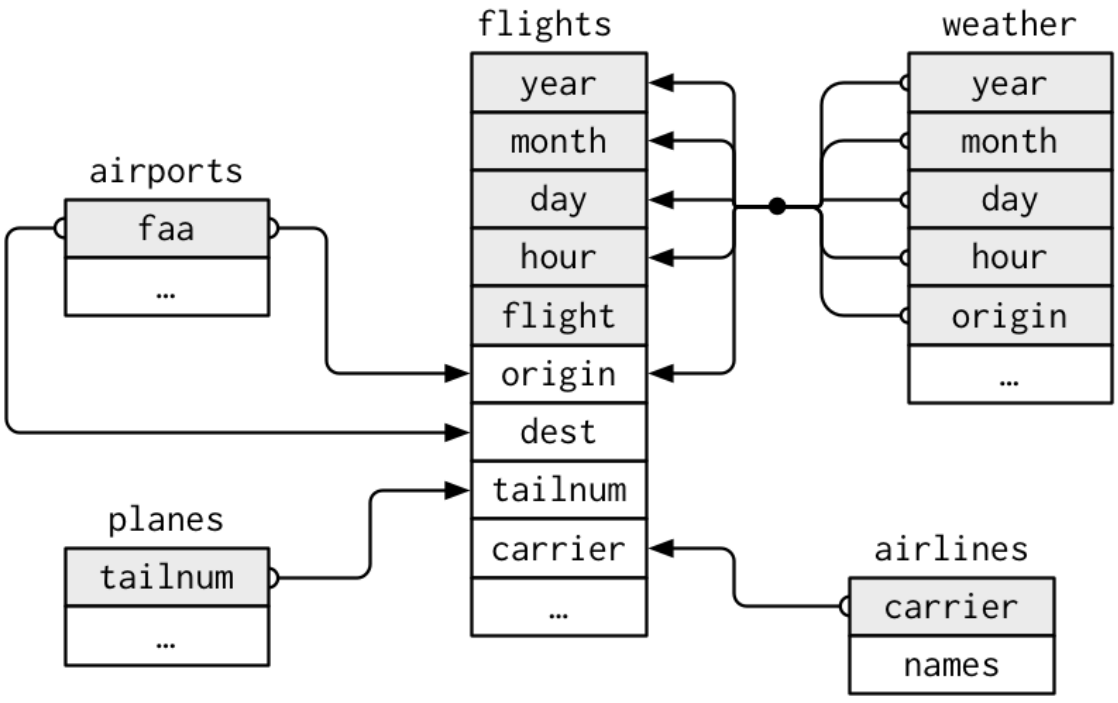
4.3 *Samenvoegen data-frames*

Een andere veel voorkomende taak is het combineren/mergen van twee verschillende datasets. In de `flights` data bijvoorbeeld staat de variabele `carrier` voor de carriercode van de verschillende verschillende vluchten. Terwijl`"UA"` en `"AA"` misschien nog relatief makkelijk zijn voor sommigen om te raden (United en American Airlines), is dat voor "VX", "HA" en en "B6" een stuk moeilijker. Deze informatie vind je in een aparte dataframe `airlines`.

```
View(airlines)
```

We zien in `airports` dat `carrier` de carriercode is terwijl `name` de volledige naam is van de vluchtmaatschappij. Door deze tabel te gebruiken kunnen we zien dat "VX", "HA" en "B6" respectievelijk corresponderen met Virgin America, Hawaiian Airlines en JetBlue. Maar wil je wel de hele tijd voor elke vlucht naar de naam van de carrier kijken in de `airlines` dataset? Nee! In plaats van dit steeds met de hand te moeten doen, laten we R dat automatisch voor ons doen.

Merk op dat de waarden in de variabele `carrier` van de `flights`-dataset matcht met de waarden van de variabele `carrier` in `airlines`. In dit geval gebruiken we de variabele `carrier` als een *sleutelvariabele* om twee dataframes te combineren/matchen. Hadley en Garrett (Wickham & Garrett,2016) maakten het volgende diagram om ons te laten begrijpen hoe de twee datasets zijn gelinkt:



Figuur 4.7: Data relaties in nycflights13 uit 'R for Data Science'

4.3.1 *Samenvoegen op basis van sleutelvariabelen*

In zowel`flights` als `airlines` heeft de sleutelvariabele waarmee we de twee dataframes willen combineren/matchen dezelfde naam in beide datasets: `carriers`. We maken gebruik van de `inner_join()` functie om het via de variabele `carrier` samen te brengen.

```
flights_joined <- flights %>%
  inner_join(airlines, by = "carrier")
View(flights)
View(flights_joined)
```

We zien dat `flights` en `flights_joined` identiek zijn behalve dat de `flights_joined` een aanvullende variabele heeft met de naam `name` waarvan de waarden van`airlines` komen.

Er zijn meer complexere functies voor het samenvoegen beschikbaar, maar de `inner_join`-functie zal bijna alle problemen waarmee je geconfronteerd wordt op is onze ervaring.

4.3.2 *Samenvoegen door sleutelvariabelen met verschillende namen*

Stel dat je geinteresseerd bent in alle vluchtbestemming van NYC in 2013 en je stelt je de volgende vragen: - “In welke steden zitten deze luchthavens?”
- “Betekent `"ORD"` Orlando?”
- “Waar is `"FLL"`?

Het `airports`-dataframe bevat luchthavencodes:

```
View(airports)
```

Echter als we zowel naar `airports` als `flights` kijken als naar de visuele representatie kijken van de relaties, zien we dat in:

- `airports` de luchthavencode de variable `faa` is
- `flights` de luchthavenconde de variable `origin` is

Om deze twee datasets samen te voegen, heeft de `inner_join` operatie een aanvullende `by` argument nodig dat rekening houdt met de verschillende namen:

```
flights %>%
  inner_join(airports, by = c("dest" = "faa"))
```

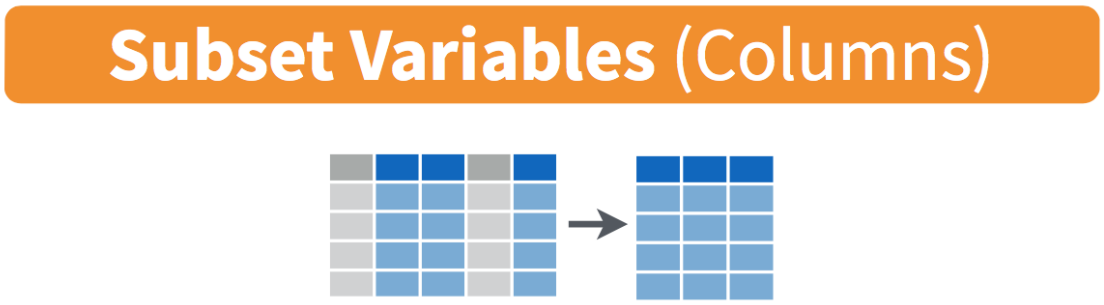
Laten deze achtereenvolgende commando’s eens maken die ons het aantal vluchten van NYC naar verschillende bestemmingen geeft maar ons ook de informatie geeft over elke bestemmingsluchthaven:

```
named_dests <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n()) %>%
  arrange(desc(num_flights)) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  rename(airport_name = name)
View(named_dests)
```

Voor het geval je het niet wist, `"ORD"` is de luchthavencode van de Chicago O'Hare luchthaven en `"FLL"` staat voor de luchthaven in Fort Lauderdale, Florida, zoals we nu ook kunnen zien in ons `named_freq_dests` dataframe.

4.4 *Optioneel: andere woorden*

4.4.1 *Selecteer variabelen door het gebruik van 'select'*



Figuur 4.8: Selecteer diagram van Data Wrangling met dplyr tidyr cheatsheet

We hebben gezien dat het `flights` dataframe in het `nycflights13` pakket heel veel verschillende variabelen omvat. De `names`-functie geeft een overzicht van alle kolommen in een dataframe; in ons geval zou je `names(flights)` runnen. Je kunt deze variabelen ook identificeren door het runnen van de `glimpse`-functie in het `dplyr` pakket:

```
glimpse(flights)
```

Stel dat het je om twee variabelen gaat, zeg `carrier` en `flight`. Dan kun je deze selecteren (select'):

```
flights %>%
  select(carrier, flight)
```

Een andere variabele is `year`. Als jij je de orginele beschrijving van het `flights` dataframe kunt herinneren (of door het runnen van `?flights`), zul jij je herinneren dat deze data corresponderen met de vluchten die in 2013 van New York City vertrokken. De `year` variabele is

niet echt een variabele die verandert hier... `flights` komt eigenlijk van een grotere dataset die verschillende jaren omvat. Misschien dat we de `year`-variabele uit onze dataset willen verwijderen omdat deze in ons geval geen extra informatie oplevert voor onze analyses. We kunnen `year` deselecteren door het gebruik van het `-` teken:

```
flights_no_year <- flights %>%
  select(-year)
names(flights_no_year)
```

Of we kunnen het bereik van de kolommen specificeren:

```
flight_arr_times <- flights %>%
  select(month:day, arr_time:sched_arr_time)
flight_arr_times
```

De selecteerfunctie (`select`) kan ook gebruikt worden voor het herordenen van de kolommen in combinatie met de `everything` -helpfunctie. Stel dat we de `hour`, `minute` en `time_hour` variabelen, die aan het einde van de `flights`-dataset verschijnen, direct willen zien na de `day`-variable:

```
flights_reorder <- flights %>%
  select(month:day, hour:time_hour, everything())
names(flights_reorder)
```

In dit geval neemt `everything()` alle overige variabelen mee. Tenslotte kunnen de helpfuncties `starts_with`, `ends_with` en `contains` worden gebruikt om de kolomnamen te kizen die met de volgende voorwaarden corresponderen:

```
flights_begin_a <- flights %>%
  select(starts_with("a"))
flights_begin_a
```

```
flights_delays <- flights %>%
  select(ends_with("delay"))
flights_delays
```

```
flights_time <- flights %>%
  select(contains("time"))
flights_time
```

4.4.2 Hernoemen van variabelen door het gebruik van 'rename'

Een andere bruikbare functie is `rename`, wat, zoals je mag verwachten, de naam van de ene kolom in een andere naam voor die kolom verandert. Stel dat we van`dep_time` en `arr_time` in het `flights_time` dataframe `departure_time` en `arrival_time` willen maken:

```
flights_time_new <- flights %>%
  select(contains("time")) %>%
  rename(departure_time = dep_time,
         arrival_time = arr_time)
names(flights_time)
```

Het is makkelijk om te vergeten of de nieuwe naam voor of na het gelijkteken komt. Ik herinner met dat veelal met "Nieuw Voor, Oud Na" oftewel NVON. Je krijgt een foutmelding als je het omdraait:

```
Error: Unknown variables: departure_time, arrival_time.
```

4.4.3 *Vind de hoogste waarde door het gebruik van 'top_n'*

We kunnen ook de `top_n`-functie gebruiken die ons automatisch vertelt wat het meest voorkomende aantal vluchten van `num_flights` is. We specificeren de top 10 luchthavens hier:

```
named_dests %>%
  top_n(n = 10, wt = num_flights)
```

We moeten dit nog steeds arrangeren door `num_flights`:

```
named_dests  %>%
  top_n(n = 10, wt = num_flights) %>%
  arrange(desc(num_flights))
```

Let op: Onthoud dat ik niet eerder iets vertelde over de `n` en `wt` argumenten. Informatie daarover kun je vinden door de `?`-functie op `top_n`.

We kunnen ook nog een stap verder gaan en de `group_by` en `summarize`-functies samen te brengen die we hebben gebruikt om de meest frequente vluchten te vinden:

```
ten_freq_dests <- flights %>%
  group_by(dest) %>%
  summarize(num_flights = n()) %>%
  top_n(n = 10) %>%
  arrange(desc(num_flights))
View(ten_freq_dests)
```

4.5 *Conclusie*

4.5.1 *Bronnen*

Zoals we zagen met de RStudio cheatsheet op data visualization, heeft RStudio ook a cheatsheet voor data manipulatie gemaakt met de titel "Data Transformation with dplyr".

- Daar kom je door hier te klikken.

In de rest van het boek (dat hier vooralsnog geen plek krijgt) wordt er meer gebruik gemaakt van de `dplyr`-functies. Verder moedigen we jou aan ook `tidyr` verder te onderzoeken voor als je te maken krijgt met data die niet in een opgeschoond format staan waar onze voorkeur naar uitgaat.

4.5.2 Script van de R code

Een R-scriptfile van alle codes die in dit hoofstuk staan is beschikbaar hier.

4.5.3 Wat komt er nu?

Hiermee sluiten we het **Data exploratie** deel van dit boek af. Nu moet je aardig handig zijn geworden in zowel het plotten van variabelen (of verschillende variabelen tegelijkertijd) in verschillende datasets en het manipuleren van data zoals we dat in dit hoofdstuk hebben gedaan. Het is goed als je nog een keer door de code van eerdere hoofdstukken heen gaat en veranderingen aanbrengt op basis van de kennis die je nu hebt opgedaan.

In de rest van het boek (dat hier niet aan de orde komt) leer je te begrijpen hoe dit deel over **Data Exploratie** kan worden gekoppeld aan statistische inferentie. Onthoud dat de nadruk daar in het algemeen ligt op datavisualisatie en dat zullen we zien als we het vervolgens hebben over sampling, resampling en bootstrapping. Dat leidt ons vervolgens weer naar hypothese testen en betrouwbaarheidsintervallen. Maar dat alles misschien een andere keer of je moet de oorspronkelijke tekst erop naslaan.

books