

# (BIG) DATA RETRIEVAL & TEXT MINING - WEEK 1

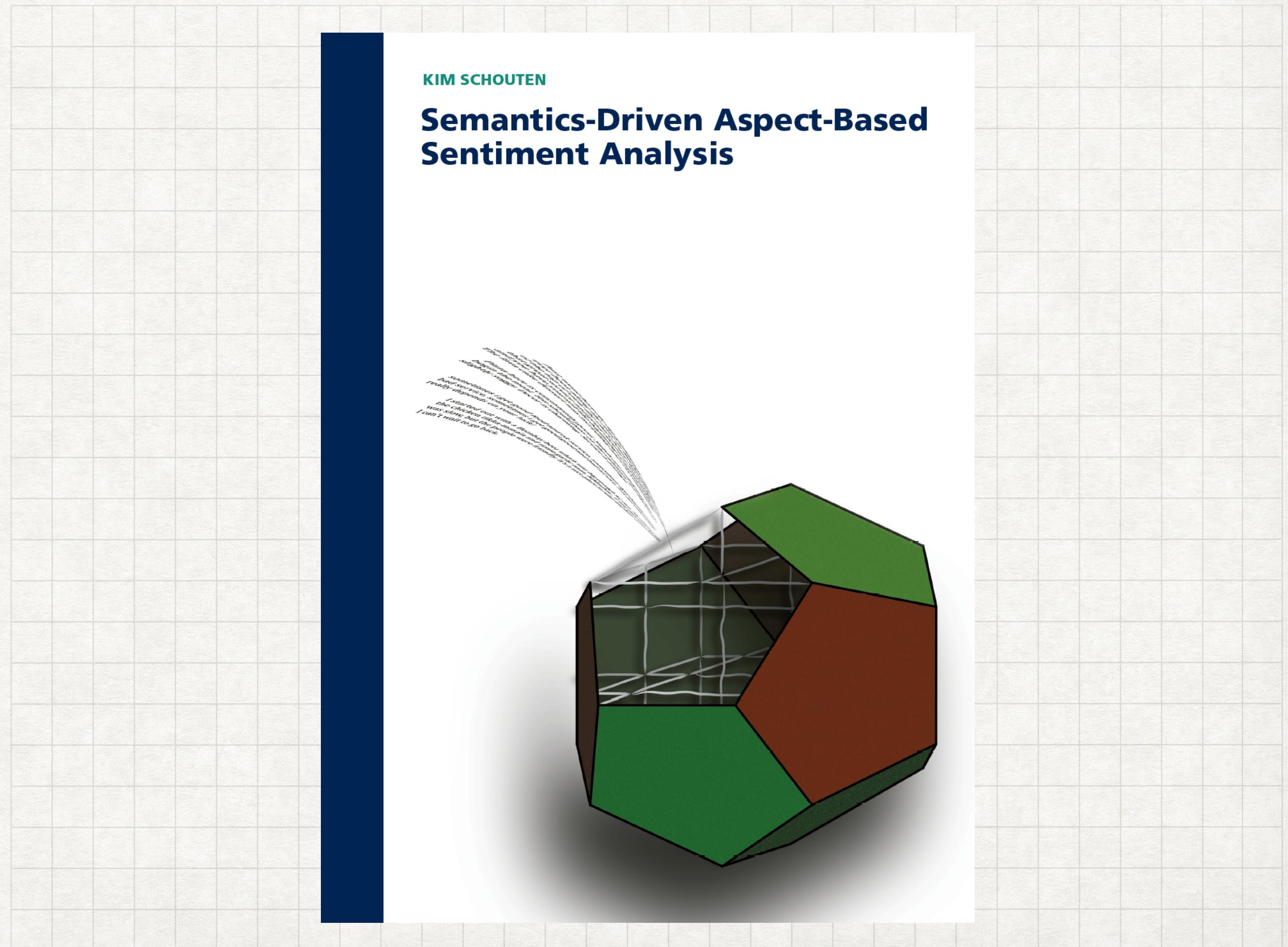
# PROGRAMMING A DATA ANALYTICS PROJECT



# INTRODUCTIE

## DOCENT

- Kim Schouten
- Data scientist bij Erasmus Q-Intelligence
- Interesses:
  - Sentiment analyse
  - Natural Language Processing
  - Onderwijs



# INTRODUCTIE

## DOEL

- Vak heeft vooral een praktische insteek
- Theorie waar nodig
- Veel zelf doen in R
- Al veel onderdelen van data analytics zijn behandeld (analyse, machine learning, vizualisatie, etc.)
- Dit vak richt zich vooral op de beginfase van een data analytics project

# INTRODUCTIE ONDERWERPEN

- Programming a Data Analytics Project
- Web scraping
- Web APIs
- Databases, SQL, Big Data
- Text Mining, Natural Language Processing
- Sentiment Analysis, Text Analytics
- Machine Learning with Textual Data

# INTRODUCTIE

## SYSTEEM VEREISTEN

- Gedurende het vak gebruiken we veel verschillende packages
- Sommige packages hebben andere software nodig die geïnstalleerd moet worden
- Zorg dat je installatie/administrator rechten hebt op je laptop

# INTRODUCTIE

## OPDRACHT

- Opdracht in drie delen
- Opdrachten komen beschikbaar na college 2, 4, en 6
- Deadlines zijn 2 feb, 16 feb en 8 maart
- Twee weken de tijd voor deel 1 en 2, en drie weken de tijd voor deel 3

# INTRODUCTIE

## OPDRACHT

- Opdracht in drie delen
- Opdrachten komen beschikbaar na college 2, 4, en 6
- Deadlines zijn 2 feb, 16 feb en 8 maart
- Twee weken de tijd voor deel 1 en 2, en drie weken de tijd voor deel 3
- Analyse zoals <https://www.nrc.nl/nieuws/2019/12/27/154109-woorden-zonder-spiekbriefje-a3985089>

# PROGRAMMING A DATA ANALYTICS PROJECT

# ZEVEN STAPPEN



# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 1: BEGRIJP DE BUSINESS / VRAAG

- Voordat je los gaat op een data set is het belangrijk te weten wat je wilt bereiken
- Maak geen oplossing voor een probleem dat niemand heeft  
*(tenzij je, net als Apple, mensen kan overtuigen dat ze dit probleem hebben)*
- Praat met mensen, stel vragen, scherp je begrip van de situatie
- Maak mensen enthousiast (incl. jezelf!) zodat je later in het proces op support kan rekenen
- Resultaat: concreet geformuleerde doelen waarmee je kan bepalen of het project een succes is geworden

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 2: DATA VERZAMELEN

- Als het doel duidelijk is kan je gaan nadenken welke data je allemaal nodig hebt
- Hoe meer hoe beter, zowel in datapunten als in databronnen
- Verschillende bronnen combineren geeft toegevoegde waarde
  - Databases (lokale/interne data)
  - APIs (cloud data)
  - Open data (overheidsdata, web scraping, publieke datasets)
- Resultaat: (verschillende) ruwe data bestanden en (bij voorkeur) de code om ze te verzamelen

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 3: DATA VERKENNEN EN OPSCHONEN

- Begrijp je data (wat zijn de variabelen/kolommen, mogelijke waarden, etc.)
- Transformeer de data vanuit verschillende formaten in iets werkbaars (bijv. dataframe)
- Deze stap met de volgende stap kosten samen vaak ~80% van de tijd!
- Resultaat: (verschillende) schone datasets (+ code)

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 4: VERRIJK DE DATA

- Combineer de verschillende data sets
- Maak afgeleide variabelen  
(bijv. maandkolommen, feestdage, verschil/som kolommen, leads/lags)
- Wees waakzaam voor *bias* in de data
- Resultaat: één geïntegreerde dataset met alle informatie die je nodig hebt (+ code)
- Opmerking: afhankelijk van je doel kan je alsnog meerdere dataframes hebben, maar deze zijn dan wel te combineren

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 5: VISUALIZEER

- Vizualizaties zijn nuttig om inzicht te krijgen in je data en om je data verder te verrijken. (stap 3 en 4)
- Ze zijn ook nuttig om het probleem waar je aan werkt in kaart te brengen en dit aantrekkelijk te communiceren naar een klant of werkgever
- Soms is een goede (interactive) visualisatie het eindproduct op zich
- Resultaat: een verzameling plots, of een deliverable met visualisaties en wat uitleg, of extra kolommen in data set (+ code)

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 6: VOORSPEL

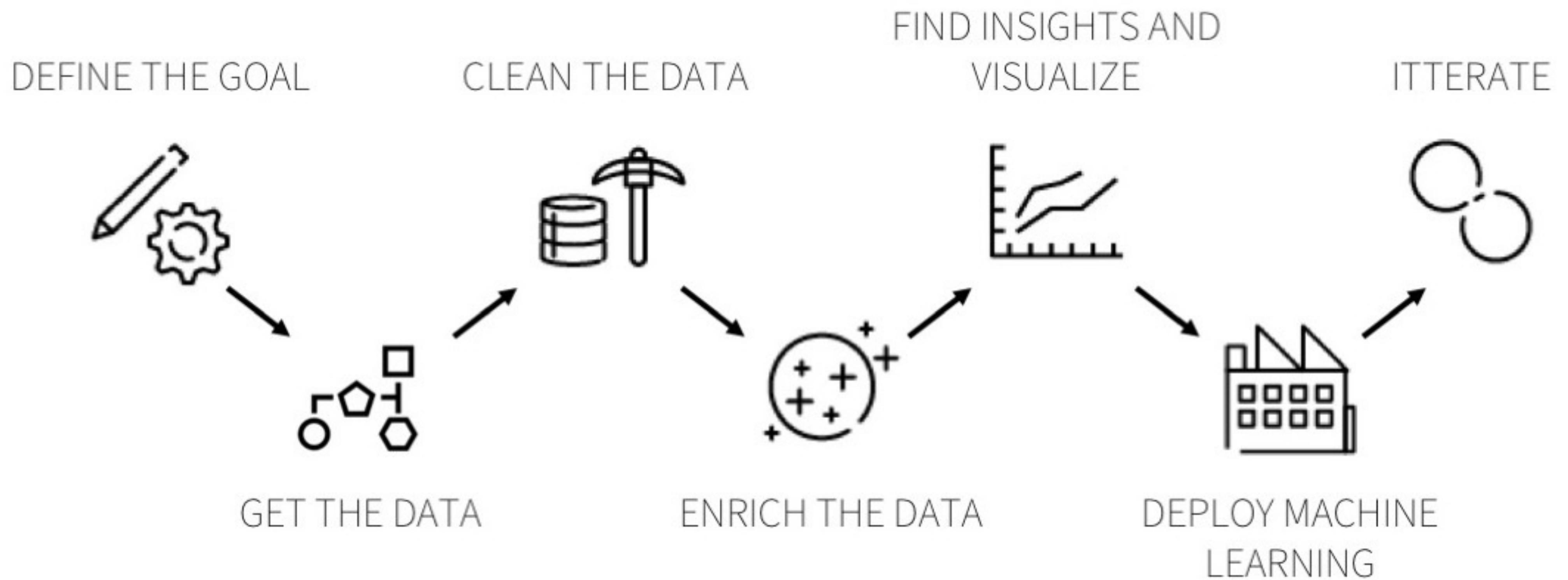
- Gebruik clustering algoritmes om groepen en trends te vinden die je met het blote oog niet kan zien
- Gebruik voorspellende algoritmes om toekomstige waarden te voorstellen
- Vergeet niet je data te splitsen zodat je de kwaliteit van je model kan testen
- Resultaat: getrainde modellen, performance rapport(en), voorspellingen (+ code)

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT

## STAP 7: HERHAAL

- De meeste data is niet statisch, en je code en modellen zullen gereeld bijgewerkt moeten worden
  - De meeste voorspelmodellen moeten regelmatig opnieuw getraind worden
  - Als je je model meer dan eens wilt gaan inzetten moet het geoperationaliseerd worden: het moet ergens gaan draaien en gebruikt kunnen worden
  - Voorbeeld: een Shiny app waarmee mensen de resultaten van je analyse interactief kunnen bekijken
  - Resultaat: een bruikbaar product

# DE ZEVEN STAPPEN VAN EEN DATA ANALYTICS PROJECT



# PROGRAMMING A DATA ANALYTICS PROJECT RSTUDIO PROJECT



# RSTUDIO PROJECT

## WAAROM

- Een RStudio Project is een praktische manier om al je bestanden geordend bij elkaar te bewaren
  - Eigen environment voor variabelen
  - Eigen folder op je harde schijf
    - Bewaar scripts, data bestanden, documentatie en output
  - Mogelijkheid tot versiebeheer met Git

# RSTUDIO PROJECT

## HOE

- Kies menu “File” -> “New Project”
- Klik “New Directory”
- Klik “New Project”
- Geef je project folder een naam, bijv. “text\_mining\_assignment”  
(gebruik liever geen spaties, dit kan soms problemen geven)
- Vink “Create a git repository” aan om versiebeheer te gebruiken voor dit project
- Klik “Create Project” knop om het project aan te maken

# RSTUDIO PROJECT ORGANISATIE

- Bewaar je ruwe data in een aparte folder (`raw_data`) en maak deze “read-only”
  - Voorkom aanpassingen aan de originele data
- Bewaar je opgeschoonde data in een aparte folder (`clean_data`)
  - Zo kan je snel je schone data inladen en verder gaan met je analyse
- Bewaar je scripts in een aparte folder (`src`)
- Bewaar je output in een aparte folder (`output` of `results`)
  - Alles in deze folder moet opnieuw te genereren zijn met je scripts

# RSTUDIO PROJECT

## SCRIPTS

- Doorgaans begin je met één lang script met een hoop code erin
- Probeer logische blokken in eigen functies onder te brengen voor meer structuur
- Kopieer nooit code: dit is een teken dat je een functie moet maken
- Het 'hoofd'-script roept de functies aan die je hebt gemaakt
- Als je meer generieke functies maakt, sla die dan op in een apart bestand
  - Dit kan nuttig zijn bij een volgend project

# RSTUDIO PROJECT

## DEPENDENCIES

- Zet de optie aan dat RStudio de Global Environment altijd leeg maakt wanneer je afsluit
  - Dit voorkomt onbedoelde afhankelijkheden met variabelen in je Global Environment
- Zorg dat je 'hoofd'-script alle packages en scripts inlaadt zodat je 'schoon' kan beginnen
  - Gooi evt. in je script de Global Environment leeg
- Gebruik altijd relatieve paden i.p.v. absolute paden
  - Een RStudio project opent altijd met de eigen folder als working directory

# RSTUDIO PROJECT FUNCTIONS

- Een functie heeft één functie
- Een functie is een input-output mapping (`output <- f(input)`)
  - Probeer neveneffecten te voorkomen (bijv. aanpassingen in `.GlobalEnv`)
  - Voorkom afhankelijkheden, anders dan op de input variabelen
- Geef een functie altijd gestructureerde documentatie (bijv. met 'Roxygen skeleton')
  - Documenteer doel van functie, input variabelen en output variabelen
  - Documenteer bijzondere omstandigheden, uitzonderingen en mogelijke errors
- Dit maakt een functie voorspelbaar, testbaar, makkelijk leesbaar

# RSTUDIO PROJECT



Do something today  
that your future self  
will thank you for.

SEAN PATRICK FLANERY

# PROGRAMMING A DATA ANALYTICS PROJECT TIPS & TRICKS



# TIPS & TRICKS

## LIBRARY(...) CONFLICTEN

- Een `library()` call voegt alle geëxporteerde functies van een package toe aan de lijst met bekende functies in je environment
- Probleem: deze lijst kan elke naam maar één keer bevatten: voeg je een package toe met functienamen die al eerder geladen waren uit andere packages dan worden deze overschreven
- Voorbeeld: `library(dplyr)` laadt enkele functies met namen die ook in de stats en base package zitten
- Tip: gebruik alleen `library()` als je vaak een functie nodig hebt uit deze package
- In alle overige gevallen kan je de functie aanroepen met de volledige naam:  
`package_name::function_name` (bijv. `glmnet::glmnet(...)`)

# TIPS & TRICKS

## DEBUGGEN VAN CODE

- Zolang je in een simpel script werkt kan je alles regel voor regel uitvoeren
- Met functies blijft een deel onzichtbaar en buiten de Global Environment
- Met een browser() call in een functie stopt R met uitvoeren op die plek en kan je alsnog regel voor regel door je code heen stappen
- Je krijgt ook zicht op wat er op dat moment in de functie environment zit
- Met debug() kan je, vanuit de browser() omgeving een andere functie ook stap voor stap doorlopen (alsof je aan het begin van de functie een browser() zou zetten)
- undebug() heft dit weer op zodat je doorkan zonder opnieuw elke keer te stoppen

# TIPS & TRICKS

## SNELHEID VAN CODE METEN

- Het kan zinvol zijn om te meten hoe lang een stuk code erover doet
- Start de teller met `tictoc::tic()` en stop de teller met `tictoc::toc()` en je krijgt de verstreken tijd in de console te zien.
- Voor meer precieze meting van kleine stukjes code, bijvoorbeeld om te bepalen welke functie sneller is, kun je beter `microbenchmark` gebruiken:
- `microbenchmark::microbenchmark({function1()}, {function2()}, ...)`
- Tip: zet op logische plekken in je code een `print(...)` neer zodat je een beetje gevoel krijgt voor hoe lang bepaalde dingen duren en waar je code op dit moment mee bezig is

# TIPS & TRICKS

## VERTRAGING

- Bepaalde code kan verrassend traag zijn
- Tip: sommige loops zijn te voorkomen door een vectorized functie te gebruiken bijv. `%in%` is vectorized:  
`c("a", "b") %in% c("a", "b", "c", "d")` geeft TRUE TRUE
- Loops zijn in zichzelf niet verkeerd en soms noodzakelijk
- Het is vaak dat wat in de loop zit wat traag is en dat wordt verergerd door het vele herhalen

# TIPS & TRICKS

## VERTRAGING

- Het kopiëren van data in R is langzaam, maar R kopieert vrijwel alles zodra je ergens iets verandert
- ```
a <- list()
for (i in 1:10000) {
  a <- c(a, i)
}
```
- Dit is traag omdat in elke iteratie er een nieuwe kopie van a wordt gemaakt
- Je kan dit checken door `print(lobstr::obj_addr(a))` in de loop te zetten  
Dit print het geheugenadres van het a object, wat elke iteratie verandert.
- Tip: normaal is dit geen probleem, maar let vooral op bij loops en grote objecten

# TIPS & TRICKS

## VERTRAGING

- Vergelijk de code in w1\_speed\_comparison.R
- Voor meer info: <https://adv-r.hadley.nz/names-values.html>

# PROGRAMMING A DATA ANALYTICS PROJECT VERSIEBEHEER MET GIT(HUB)



# VERSIEBEHEER MET GIT(HUB)

## GIT AANWEZIG?

- Kies in menu “File” voor “New Project”
- Kies, indien mogelijk, voor de optie “Version Control”
- Kies, indien mogelijk, voor de optie “Git”
- Hier kan je de URL van een Git repository (bijv. op GitHub) invullen en dan download RStudio automatisch dat project naar je computer en opent dit
- Als je ergens een optie mist of je krijgt de melding dat RStudio Git niet kan vinden dan heb je waarschijnlijk Git niet geïnstalleerd
- Installeer Git van <https://git-scm.com/> indien mogelijk

# VERSIEBEHEER MET GIT(HUB)

## WAAROM?

- Met versiebeheer maak je regelmatig 'snapshots' van je code die je voorziet van een korte omschrijving (niet als 'Dropbox', je moet zelf deze 'snapshots' maken)
- Hierdoor hou je overzicht op wat je al gedaan hebt en op wat je nog moet doen
- Bij ernstige fouten kan je ook terugvallen op een vorige versie van je code
- Voor experimentele aanpassingen kan je ook een tijdelijke aftakking van je code maken. Als je experiment lukt voeg je de code eenvoudig samen, als het niet lukt kan je de vertakking weggooien zonder dat het effecten heeft op de (werkende) hoofdversie van de code.
- Git is de meestgebruikte software voor versiebeheer en wordt vanuit RStudio ondersteund

# VERSIEBEHEER MET GIT(HUB)

## ONLINE OF OFFLINE?

- Git kan je lokaal gebruiken (zonder Internet, zonder GitHub)
- Om je code met anderen te delen en om een backup te hebben is het wel raadzaam een online repository te hebben, zoals een account op GitHub (of BitBucket, of GitLab, etc.)
- In dat geval moet je behalve het maken van de 'snapshots' ook synchroniseren met de server

# VERSIEBEHEER MET GIT(HUB)

## ONLINE OF OFFLINE?

- Een Git repository start in een door jou gekozen folder (bijv. de RStudio project folder)
- Je geeft zelf aan welke bestanden er in je repository bijgehouden moeten worden
- In principe hou je alleen de wijzigingen bij van bestanden met code/instellingen etc.
- Data bestanden, gegenereerde output bestanden, bestanden met wachtwoorden, etc. stop je niet in je Git repository
- Om bepaalde bestanden automatisch te negeren kun je het `.gitignore` bestand gebruiken

# VERSIEBEHEER MET GIT(HUB)

## SNAPSHOTS MAKEN (COMMITS)

- Zodra je een bestand aanvinkt in RStudio zit het in de *staging* fase en staat het klaar om meegenomen te worden in de volgende snapshot
- Zodra je op *commit* klikt, worden alle aangevinkte bestanden meegenomen in de *snapshot* ofwel *commit*.
- **Let op:** je kan pas een *commit* uitvoeren als je een username en email adres hebt gegeven aan Git
  - elke *commit* wordt voorzien van deze naam en email, dit moet ook als je Git alleen offline gebruikt
  - dit is niet dezelfde username of email als je GitHub account en hoeft geen bestaand mail adres te zijn (wel handig)
- `git config --global user.name "Mona Lisa"`
- `git config --global user.email "mona.lisa@louvre.fr"`

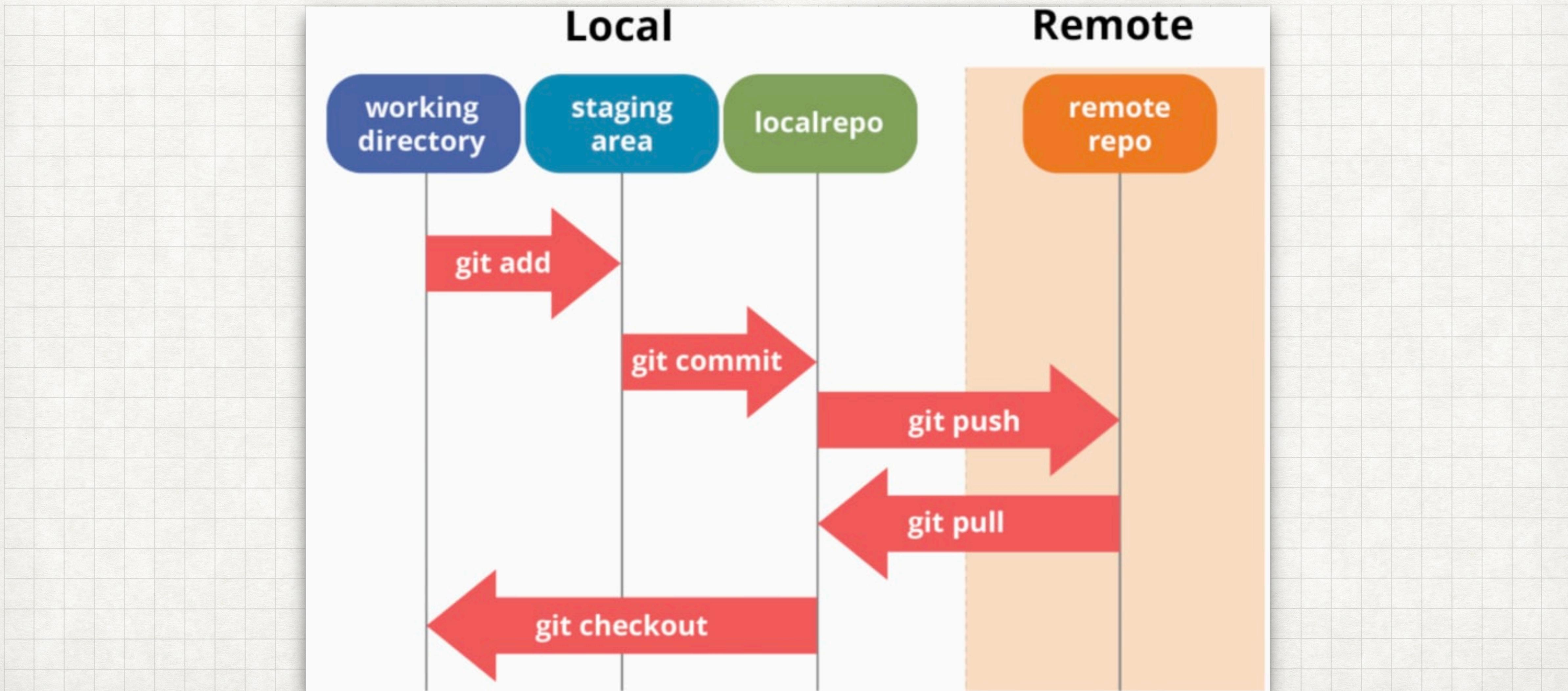
# VERSIEBEHEER MET GIT(HUB)

## GITHUB ACCOUNT

- Maak een account
- Maak een test repository (zonder README of LICENSE)
- Vind de twee commando's onder  
**"...or push an existing repository from the command line":**
- Voer deze uit in de Terminal of Git Bash, hiermee koppel je jouw lokale Git repository aan de *remote repository* op GitHub
- Dit geeft je de mogelijkheid om jouw wijzigingen naar de server te sturen met *push* en om wijzigingen (van bijv. iemand anders of vanaf een andere computer) op te halen met *pull*.

# VERSIEBEHEER MET GIT(HUB)

## WORKFLOW



# VERSIEBEHEER MET GIT(HUB)

## GITHUB PAGES

- Bij je repository instellingen kun je GitHub Pages aanzetten
- Hiermee kun je bijvoorbeeld je output rapport automatisch publiceren als website
- Wijzigingen maak je door je code te veranderen, het rapport opnieuw te draaien en alles te committen en te pushen naar GitHub
- Dit werkt alleen voor eenvoudige websites: de html output van een RMarkDown rapport werkt wel, maar een Shiny app dan weer niet
- Let op: de website moet zich bevinden in je hoofd folder of in de /docs folder

# VERSIEBEHEER MET GIT(HUB)

## MEER INFO

- Git is een zeer uitgebreid systeem en niet altijd even intuïtief
- Daarom veel hulp te vinden online!
- Het Git Book (beetje theoretisch, eerste 3 hoofdstukken zijn wel nuttig): <https://git-scm.com/book>
- Praktischer: <https://swcarpentry.github.io/git-novice/> (zie ook puntje 14 voor info over Git in RStudio)

# PROGRAMMING A DATA ANALYTICS PROJECT

