

# Een uitnodiging tot data science met R

Frank van der Meulen (Technische Universiteit Delft en ProjectsOne)

Veel data analyse vindt nog steeds plaats in spreadsheetprogramma's als Excel. Maar is dit wel zo handig? Daar is geen eenduidig antwoord op. Sommigen onder ons zijn ware Excel experts en kunnen verrassend veel bewerkingen gedaan krijgen binnen dit pakket. Het gaat dan om "data-handling", het bewerken van de data naar een wat prettiger vorm. Draaitabellen zijn hier een onderdeel van. Er kleven echter ook wat nadelen aan zo'n aanpak:

- Excel is traag met grote databestanden;
- Excel is zeer beperkt in statistische methoden;
- de workflow is niet eenvoudig reproduceerbaar.

Om aan het tweede bezwaar tegemoet te komen is binnen de lean-six-sigma gemeenschap het pakket Minitab een welbekende aanvulling. Copy-paste de data van Excel naar Minitab (of lees direct in in Minitab), en voer aanvullende statistische analyses uit in Minitab. De gebruikersvriendelijkheid van Minitab is voor veel mensen een groot voordeel. Echter, wat minder bekend lijkt te zijn, is dat het pakket zich beperkt tot eenvoudige statistische analyses. Afhankelijk van de gegeven data kan dit voldoende zijn, maar aangezien we steeds meer data verzamelen, met toenemende complexiteit van databestanden, is het steeds vaker het geval dat dit onvoldoende is.

En zodra over grote databestanden gesproken wordt, valt de term "machine learning" al snel, als ware het een magisch middel dat de informatie uit de data tevoorschijn tovert. Talloze tutorials zijn er op te vinden op internet waar methoden met prachtige namen als "random forest", "gradient boosting machine", "support vector machines" toegepast worden op een keurige dataset. Dat is interessant, maar veelal zit in het voorbereiden van de data veruit de grootste inspanning, alvorens dit soort methoden toegepast kan worden.

## Tidy data

Vrijwel alle statistisch pakketten gaan uit van data in "tidy"-format. Dit betekent:

- iedere rij in de dataset correspondeert met 1 experimentele eenheid (met andere woorden: per "wat" je meet)
- iedere kolom correspondeert met een variabele.

Dus als concreet voorbeeld: stel dat voor iedere week van het jaar de omzet van verschillende vestigingen van een bedrijf gegeven zijn, als ook de grootte van de vestiging. Dan veronderstellen we bij "tidy"-data 4 kolommen: "omzet", "weeknr", "grootte vestiging", "id vestiging". Iedere rij in de data correspondeert dan met een week.

Helaas zijn data vaak niet in dit gewenste formaat. Nu kun je eenmalig met veel moeite je data in Excel omzetten (als dit al lukt). Maar stel nu dat je iedere week of maand diezelfde rapportage uit moet voeren? Dan is reproduceerbaarheid cruciaal. Hoe fijn zou het zijn als je met een druk op de knop alle bewerkingstappen op nieuwe data kunt toepassen? Een bijkomend niet te onderschatten voordeel: slechts eenmaal moeten alle bewerkingstappen heel precies gecheckt worden, daarna niet meer. Met andere woorden: een reproduceerbare werkmethode vermindert fouten. En ja, ik geloof dat heel veel analyses (ook in de wetenschap) fouten bevatten door een onoverzichtelijke, niet reproduceerbare workflow in Excel.

## Case study

Bij een middelgrote organisatie wordt een workflow systeem gebruikt voor het goedkeuren van facturen. Zo zijn er verschillende activiteiten die bij dit proces horen. Denk bijvoorbeeld aan: het ontvangen van de factuur, doorsturen naar de juiste persoon ter goedkeuring en betaalbaar stellen. Het bijbehorende bestand bestaat uit een regel per activiteit. Per regel is aangegeven:

- het nummer van de factuur
- naam van de activiteit
- ActivityInboxDate: datum en tijd waarop de factuur naar de betrokken activiteit is gestuurd
- ActivityClaimDate: datum en tijd waarop met de activiteit gestart is
- ActivityCompletionDate: datum en tijd waarop de activiteit is afgerond.

In onderstaande screenshot is weergegeven welke velden (geel) te gebruiken zijn om de doorlooptijd van een factuur te bepalen. Let op: bepalend zijn steeds de eerste en laatste regel per factuur. De activiteitscode hoeft niet steeds hetzelfde te zijn!

PRNumbr	ActivityCode	ActivityInboxDate	ActivityClaimDate	ActivityCompletionDate
010875	LOG Check by AP	5-01-18 13:00	11-01-18 8:52	18-01-18 9:56
010875	LOG Check by AP	5-01-18 13:00	11-01-18 8:52	18-01-18 9:56
010875	LOG Check by AP	5-01-18 13:00	11-01-18 8:52	18-01-18 9:56
010875	LOG Check by AP	5-01-18 13:00	11-01-18 8:52	18-01-18 9:56
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Approve 2	3-01-18 11:43	9-01-18 15:32	9-01-18 15:32
010894	LOG Place order	9-01-18 15:41	10-01-18 14:12	10-01-18 14:12
010894	LOG Place order	9-01-18 15:41	10-01-18 14:12	10-01-18 14:12
010894	LOG Place order	9-01-18 15:41	10-01-18 14:12	10-01-18 14:12
010894	LOG Place order	9-01-18 15:41	10-01-18 14:12	10-01-18 14:12
010894	LOG Check by AP	10-01-18 14:12	10-01-18 14:20	10-01-18 14:21
010894	LOG Check by AP	10-01-18 14:12	10-01-18 14:20	10-01-18 14:21
010907	LOG Compose Purchase	3-01-18 9:58	3-01-18 9:58	3-01-18 9:58
010907	LOG Approve 1	3-01-18 9:58	8-01-18 13:03	8-01-18 13:04
010907	LOG Approve 1	3-01-18 9:58	8-01-18 13:03	8-01-18 13:04

Figure 1: Een stukje van de data

## WAT IS DE DOORLOOPTIJD PER FACTUUR?

Dat gaan we stapsgewijs onderzoeken, gevolgd door een visualisatie van de doorlooptijd. Bedenk vooraf: “Hoe zou ik dit zelf aanpakken?”. En mocht je bekend zijn met Excel en Minitab, hoe zou je met behulp van deze pakketten dit kunnen oplossen?

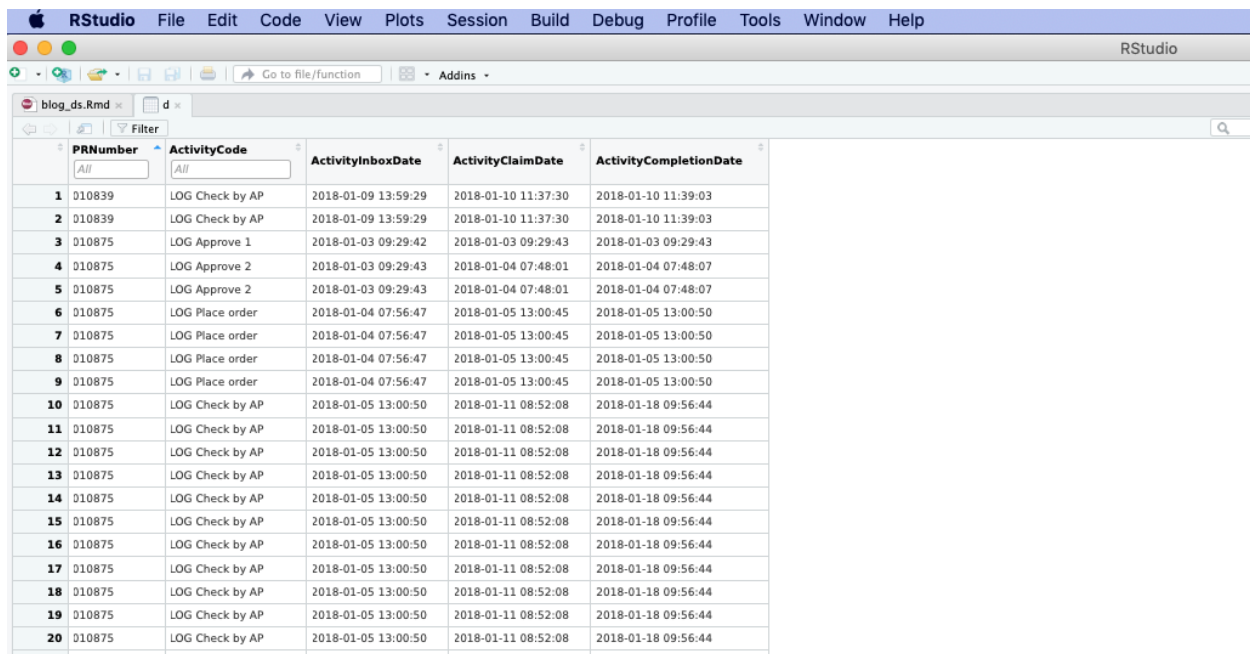
### Opschonen van de data

Opschonen van de data is vereist omdat de data niet in *tidy*-format zijn aangeleverd. Dit is eenvoudig in te zien: de experimentele eenheid is `PRNumber`, maar de gegevens voor ieder `PRNumber` staan over meerdere regels verdeeld. Vervelend is daarbij ook dat het aantal regels waarover de gegevens staan niet altijd gelijk is voor ieder `PRNumber`. Om dit met de hand aan te passen is zeer arbeidsintensief en foutgevoelig. Bovendien resulteert het in een niet-reproduceerbare workflow. We zullen later zien dat als we de data eenmaal opgeschoond hebben, het berekenen van de gevraagde doorlooptijd vrij eenvoudig is.

R is geen pakket waarin je via vensters de gewenste bewerking aanklinkt. In plaats daarvan schrijf je een script, dat wil zeggen een tekst bestand waarin je de computer vertelt wat de gewenste bewerkingen zijn. Het is direct duidelijk dat dit initieel iets lastiger is, maar automatisch een reproduceerbare workflow geeft. Inlezen van de data is in dit geval niet lastig.

```
d <- read_excel("Data.xlsx")
```

Met `View` is direct een overzicht zoals in Excel te verkrijgen (voor grote datasets is het bovendien veel sneller dan Excel).



	PRNumber	ActivityCode	ActivityInboxDate	ActivityClaimDate	ActivityCompletionDate
1	010839	LOG Check by AP	2018-01-09 13:59:29	2018-01-10 11:37:30	2018-01-10 11:39:03
2	010839	LOG Check by AP	2018-01-09 13:59:29	2018-01-10 11:37:30	2018-01-10 11:39:03
3	010875	LOG Approve 1	2018-01-03 09:29:42	2018-01-03 09:29:43	2018-01-03 09:29:43
4	010875	LOG Approve 2	2018-01-03 09:29:43	2018-01-04 07:48:01	2018-01-04 07:48:07
5	010875	LOG Approve 2	2018-01-03 09:29:43	2018-01-04 07:48:01	2018-01-04 07:48:07
6	010875	LOG Place order	2018-01-04 07:56:47	2018-01-05 13:00:45	2018-01-05 13:00:50
7	010875	LOG Place order	2018-01-04 07:56:47	2018-01-05 13:00:45	2018-01-05 13:00:50
8	010875	LOG Place order	2018-01-04 07:56:47	2018-01-05 13:00:45	2018-01-05 13:00:50
9	010875	LOG Place order	2018-01-04 07:56:47	2018-01-05 13:00:45	2018-01-05 13:00:50
10	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
11	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
12	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
13	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
14	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
15	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
16	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
17	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
18	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
19	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44
20	010875	LOG Check by AP	2018-01-05 13:00:50	2018-01-11 08:52:08	2018-01-18 09:56:44

Mijn bewering is dat de belangrijkste stap in het opschonen van de data gedaan kan worden met de volgende regels:

```
dclean <- d %>%
  distinct() %>%
  group_by(PRNumber) %>%
  slice(c(1,n())) %>%
  mutate(ordening = c("S", "E")) %>%
```

```
pivot_wider(names_from=ordering, values_from=c("ActivityCode", "ActivityInboxDate",
                                               "ActivityClaimDate", "ActivityCompletionDate"))
```

Ok, fijn dat dit blijkbaar zo kan, maar is hier iets van te snappen? Is dit iets dat een ervaren statisticus met bekendheid in R zo even snel opschrijft? Nee, dat is niet het geval. Het is belangrijk te begrijpen dat bovenstaand stukje code is opgebouwd uit losse eenvoudige bewerkingen, verbonden door **en dan** statements, die in R worden weergegeven met `%>%`. De kracht schuilt in de beschikbaarheid van een ruime collectie van zulke eenvoudige bewerkingen.

Merk eerst op dat doordat we deze dataset verkregen hebben uit een grotere dataset en er uit de oorspronkelijk dataset kolommen weggelaten zijn, er rijen zijn die exact hetzelfde zijn. Deze zijn nu dus redundant. Alle dubbelingen kunnen verwijderd worden met `distinct`.

```
d
```

```
## # A tibble: 1,012 x 5
##   PRNumber ActivityCode ActivityInboxDate ActivityClaimDate
##   <chr>      <chr>          <dtm>              <dtm>
## 1 010839    LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 2 010839    LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 3 010875    LOG Approve~ 2018-01-03 09:29:42 2018-01-03 09:29:43
## 4 010875    LOG Approve~ 2018-01-03 09:29:43 2018-01-04 07:48:01
## 5 010875    LOG Approve~ 2018-01-03 09:29:43 2018-01-04 07:48:01
## 6 010875    LOG Place o~ 2018-01-04 07:56:47 2018-01-05 13:00:45
## 7 010875    LOG Place o~ 2018-01-04 07:56:47 2018-01-05 13:00:45
## 8 010875    LOG Place o~ 2018-01-04 07:56:47 2018-01-05 13:00:45
## 9 010875    LOG Place o~ 2018-01-04 07:56:47 2018-01-05 13:00:45
## 10 010875   LOG Check b~ 2018-01-05 13:00:50 2018-01-11 08:52:08
## # ... with 1,002 more rows, and 1 more variable: ActivityCompletionDate <dtm>
```

```
distinct(d)
```

```
## # A tibble: 227 x 5
##   PRNumber ActivityCode ActivityInboxDate ActivityClaimDate
##   <chr>      <chr>          <dtm>              <dtm>
## 1 010839    LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 2 010875    LOG Approve~ 2018-01-03 09:29:42 2018-01-03 09:29:43
## 3 010875    LOG Approve~ 2018-01-03 09:29:43 2018-01-04 07:48:01
## 4 010875    LOG Place o~ 2018-01-04 07:56:47 2018-01-05 13:00:45
## 5 010875    LOG Check b~ 2018-01-05 13:00:50 2018-01-11 08:52:08
## 6 010894    LOG Approve~ 2018-01-03 11:43:02 2018-01-09 15:32:42
## 7 010894    LOG Place o~ 2018-01-09 15:41:26 2018-01-10 14:12:34
## 8 010894    LOG Check b~ 2018-01-10 14:12:45 2018-01-10 14:20:44
## 9 010907    LOG Compose~ 2018-01-03 09:58:50 2018-01-03 09:58:50
## 10 010907   LOG Approve~ 2018-01-03 09:58:52 2018-01-08 13:03:35
## # ... with 217 more rows, and 1 more variable: ActivityCompletionDate <dtm>
```

Merk op dat het aantal rijen van 1012 naar 227 is gegaan. Een andere manier om dit te doen is

```
d %>% distinct()
```

en dat is precies het eerste stukje van de code! Vervolgens willen we per `PRNumber` de eerste en laatste regel. M.a.w. we willen eerst groeperen naar `PRNumber`, en dan een deel van de rijen voor ieder geconstrueerd groepje eruit halen (in dit geval de eerste en laatste regel). Het selecteren van bepaalde rijen kan met ofwel `filter`, ofwel `slice`. Hier gebruiken we het laatste commando:

```
d %>%
  distinct() %>%
```

```
group_by(PRNumber) %>%
slice(c(1,n()))
```

```
## # A tibble: 98 x 5
## # Groups:   PRNumber [49]
##   PRNumber ActivityCode ActivityInboxDate ActivityClaimDate
##   <chr>      <chr>          <dtm>              <dtm>
## 1 010839 LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 2 010839 LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 3 010875 LOG Approve~ 2018-01-03 09:29:42 2018-01-03 09:29:43
## 4 010875 LOG Check b~ 2018-01-05 13:00:50 2018-01-11 08:52:08
## 5 010894 LOG Approve~ 2018-01-03 11:43:02 2018-01-09 15:32:42
## 6 010894 LOG Check b~ 2018-01-10 14:12:45 2018-01-10 14:20:44
## 7 010907 LOG Compose~ 2018-01-03 09:58:50 2018-01-03 09:58:50
## 8 010907 LOG Check b~ 2018-01-10 10:20:52 2018-01-10 13:34:55
## 9 010908 LOG Compose~ 2018-01-03 10:07:59 2018-01-03 10:07:59
## 10 010908 LOG Check b~ 2018-01-10 10:18:04 2018-01-10 13:36:31
## # ... with 88 more rows, and 1 more variable: ActivityCompletionDate <dtm>
```

Dit leest dus als volgt:

- neem de data in d;
- verwijder dubbele rijen;
- groepeer ten opzichte van PNumber;
- pak de eerste (1) en laatste regel van ieder groepje (n()).

Nu krijgen we dus bij ieder PRNumber de eerste en laatste regel. Om dit duidelijk te krijgen, voegen we een kolom toe waarin voor iedere regel ofwel S (Start) ofwel E (Einde) komt te staan.

```
d %>%
distinct() %>%
group_by(PRNumber) %>%
slice(c(1,n())) %>%
mutate(ordening = c("S", "E"))
```

```
## # A tibble: 98 x 6
## # Groups:   PRNumber [49]
##   PRNumber ActivityCode ActivityInboxDate ActivityClaimDate
##   <chr>      <chr>          <dtm>              <dtm>
## 1 010839 LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 2 010839 LOG Check b~ 2018-01-09 13:59:29 2018-01-10 11:37:30
## 3 010875 LOG Approve~ 2018-01-03 09:29:42 2018-01-03 09:29:43
## 4 010875 LOG Check b~ 2018-01-05 13:00:50 2018-01-11 08:52:08
## 5 010894 LOG Approve~ 2018-01-03 11:43:02 2018-01-09 15:32:42
## 6 010894 LOG Check b~ 2018-01-10 14:12:45 2018-01-10 14:20:44
## 7 010907 LOG Compose~ 2018-01-03 09:58:50 2018-01-03 09:58:50
## 8 010907 LOG Check b~ 2018-01-10 10:20:52 2018-01-10 13:34:55
## 9 010908 LOG Compose~ 2018-01-03 10:07:59 2018-01-03 10:07:59
## 10 010908 LOG Check b~ 2018-01-10 10:18:04 2018-01-10 13:36:31
## # ... with 88 more rows, and 2 more variables: ActivityCompletionDate <dtm>,
## #   ordening <chr>
```

De laatste stap om de data netjes (*tidy*) te maken, is er voor te zorgen dat iedere regel alle informatie voor een specifiek PRNumber bevat. Dit betekent dat het aantal rijen halveert, en er extra kolommen toegevoegd moeten

worden (voor ActivityCode, ActivityInboxDate, ActivityClaimDate en ActivityCompletionDate). Het data bestand wordt dus breder (wider).

```
d1 <- d %>%
  distinct() %>%
  group_by(PRNumber) %>%
  slice(c(1,n())) %>%
  mutate(ordening = c("S", "E")) %>%
  pivot_wider(names_from=ordening, values_from=c("ActivityCode", "ActivityInboxDate",
                                                "ActivityClaimDate", "ActivityCompletionDate"))
glimpse(d1)
```

```
## Observations: 49
## Variables: 9
## Groups: PRNumber [49]
## $ PRNumber          <chr> "010839", "010875", "010894", "010907", "0...
## $ ActivityCode_S    <chr> "LOG Check by AP", "LOG Approve 1", "LOG A...
## $ ActivityCode_E    <chr> "LOG Check by AP", "LOG Check by AP", "LOG...
## $ ActivityInboxDate_S <dtm> 2018-01-09 13:59:29, 2018-01-03 09:29:42,...
## $ ActivityInboxDate_E <dtm> 2018-01-09 13:59:29, 2018-01-05 13:00:50,...
## $ ActivityClaimDate_S <dtm> 2018-01-10 11:37:30, 2018-01-03 09:29:43,...
## $ ActivityClaimDate_E <dtm> 2018-01-10 11:37:30, 2018-01-11 08:52:08,...
## $ ActivityCompletionDate_S <dtm> 2018-01-10 11:39:03, 2018-01-03 09:29:43,...
## $ ActivityCompletionDate_E <dtm> 2018-01-10 11:39:03, 2018-01-18 09:56:44,...
```

Hier hebben we het opgeschoonde databestand opgeslagen in d1. Het laatste commando doet precies wat het zegt: we willen een glimpse van de data zien. We zien de kolommen in ons opgeschoonde databestand met daarbij enkele van de eerste elementen (van iedere kolom).

## Visualiseren van de data

Alhoewel de nadruk hier ligt op het illustreren van mogelijkheden om data op te schonen in R, doen we aansluitend ook nog een eenvoudige visualisatie. Eerst berekenen we de verstreken tijd, wat we `timespan` noemen:

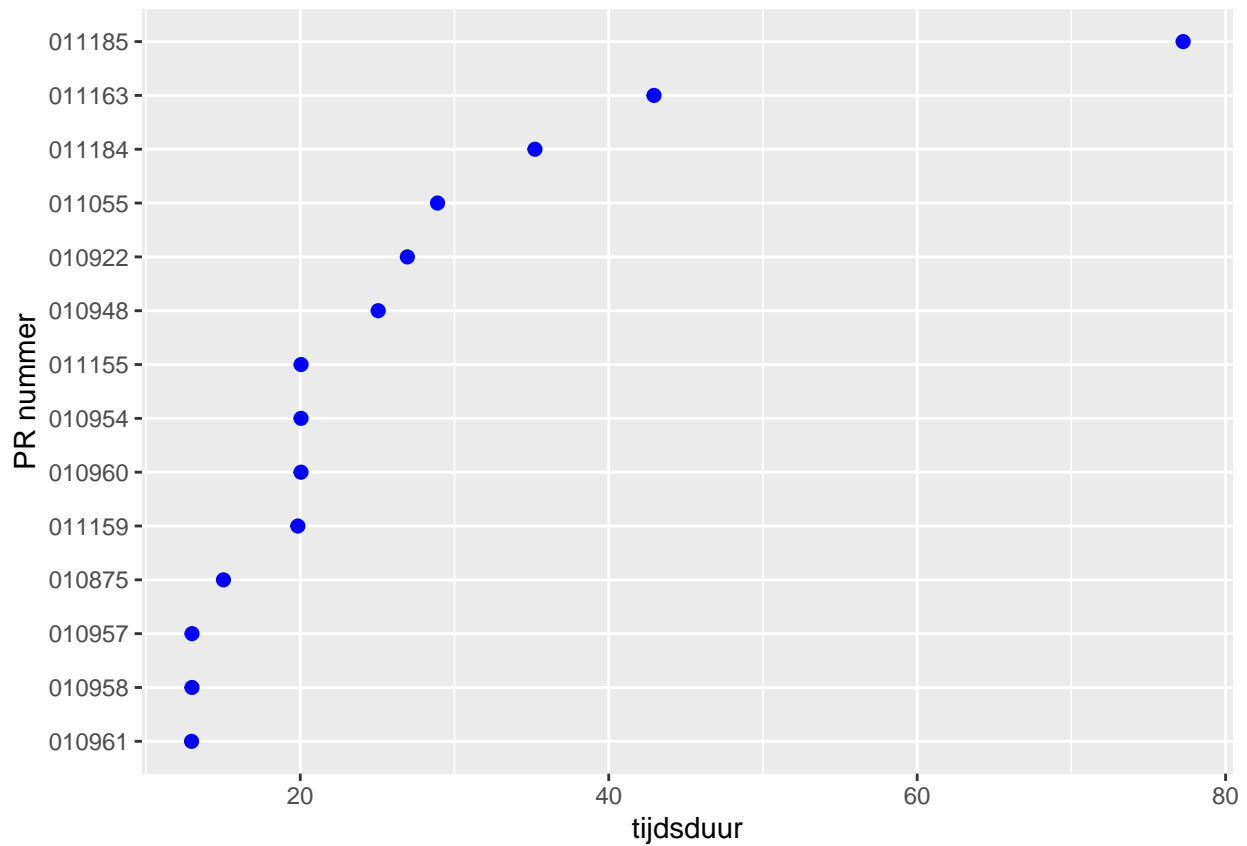
```
d2 <- d1 %>% dplyr::select(PRNumber, ActivityInboxDate_S, ActivityCompletionDate_E) %>%
  mutate(timespan=difftime(ActivityCompletionDate_E, ActivityInboxDate_S, units='days'))
head(d2)
```

```
## # A tibble: 6 x 4
## # Groups:   PRNumber [6]
##   PRNumber ActivityInboxDate_S ActivityCompletionDate_E timespan
##   <chr>      <dtm>                <dtm>                <drtn>
## 1 010839    2018-01-09 13:59:29 2018-01-10 11:39:03    0.9024768 days
## 2 010875    2018-01-03 09:29:42 2018-01-18 09:56:44   15.0187732 days
## 3 010894    2018-01-03 11:43:02 2018-01-10 14:21:14    7.1098611 days
## 4 010907    2018-01-03 09:58:50 2018-01-10 13:36:20    7.1510417 days
## 5 010908    2018-01-03 10:07:59 2018-01-10 13:36:47    7.1450000 days
## 6 010909    2018-01-03 10:10:34 2018-01-04 17:14:11    1.2941782 days
```

Hiermee krijgen we de verstreken tijd in dagen. Een nette visualisatie is weer een stapje verder, en bestaat ook weer uit wat eenvoudige stappen, die we aan “aan elkaar plakken”.

```
d2 %>%
  filter(timespan > 10) %>%
  ggplot(aes(timespan, fct_reorder(PRNumber, timespan))) +
```

```
geom_point(size=2, colour='blue') +
ylab("PR nummer") + xlab("tijdsduur")
```



### R, Python, Julia... Welke taal?

Hier is gekozen voor het statistisch pakket R. Waarom? Niet noodzakelijk omdat het de meest elegante programmeertaal is. Echter, er is een gigantische functionaliteit aanwezig, en met name voor het opschonen van data is het zgn. *tidyverse*-package, waar ik hier gebruik van heb gemaakt, zeer goed ontwikkeld, gedocumenteerd en geïntegreerd in R-studio. Op den duur zou deze plaats wel eens ingenomen kunnen worden door het wat recentere Julia, maar voor toepassers die nu vlot aan de slag willen en niet de beschikking hebben over een achtergrond in scientific computing, is R thans een uitstekende optie.

### Interesse?

Ieder jaar wordt er bij ProjectsOne een 5-daagse data-science cursus gegeven.

Verdere informatie is op <https://projectsone.nl/>.