R - Dataverwerking

J.J. van Nijnatten

Contents

contact: J.J.vanNijnatten@uva.nl

broncode: https://github.com/jonasvannijnatten/R_Data_wrangling

1 Datasets aanmaken

Hieronder staat de code om de datasets te genereren die in deze handleiding wordt gebruikt: Toon code om data te genereren

```
set.seed(06)   # set seed
nrofsubs  = 4  # set number of subjects
nrofconds = 3  # set number of conditions
subj = rep(1:nrofsubs,nrofconds)                  # create array with subject IDs
time = rep(LETTERS[1:nrofconds],each=nrofsubs)    # create array with condition values
score = as.vector( replicate(
          nrofconds , rnorm(n = nrofsubs, mean = sample(8,1)+10 , sd = sample(5,1) )
        ) )                                       # create array with measurement values
data.long = data.frame(subj, time, score);        # combine arrays into a data.frame
rm(list=setdiff(ls(), c("data.long", "nrofsubs","nrofconds"))) # delete arrays


# transform data from long to wide format ----
data.wide =
  reshape(data = data.long        # naam van de oude dataset
          ,direction = 'wide'     # richting van de data-transformatie
          ,v.names = 'score'      # kolomnaam met de meetwaarden
          ,timevar = 'time'       # kolomnaam met de conditienamen
          ,idvar = 'subj'         # kolomnaam met de ppn-nummers
  )
names(data.wide)[2:4] = c('A','B','C') # pas de kolomnamen met scores aan

# create data.bad ----
data.bad           = data.long     # kopieer data.long
data.bad[c(4,7),3] = NA            # vervang rij 4 & 7 van kolom 3 met NA
data.bad[4,2]      = NA            # vervang rij 4     van kolom 2 met NA
data.bad[2,3]      = NaN           # vervang rij 2     van kolom 3 met NaN
data.bad[9,3]      = 35.90         # vervang rij 9     van Kolom 3 met 35.90
data.bad[13,]      = data.bad[12,]  # dupliceer rij 12 in rij 13
rownames(data.bad) = NULL          # verwijder rownames
```

Toon dataset **data.long**

```
##      subj     time     score
##         1        A     11.85
##         2        A     19.34
##         3        A     23.64
##         4        A     15.12
##         1        B      9.45
##         2        B     19.69
##         3        B     16.22
##         4        B     10.76
##         1        C     12.11
##         2        C     21.27
##         3        C     16.16
##         4        C     15.00
```

Toon dataset **data.wide**

```
##      subj        A         B        C
##         1     11.9      9.45     12.1
##         2     19.3     19.69     21.3
##         3     23.6     16.22     16.2
##         4     15.1     10.76     15.0
```

Toon dataset **data.bad**

```
##        subj     time     score
## 1         1        A     11.85
## 2         2        A       NaN
## 3         3        A     23.64
## 4         4     <NA>        NA
## 5         1        B      9.45
## 6         2        B     19.69
## 7         3        B        NA
## 8         4        B     10.76
## 9         1        C     35.90
## 10        2        C     21.27
## 11        3        C     16.16
## 12        4        C     15.00
## 13        4        C     15.00
```

inhoud

---

2 Herstructureren van data

2.1 Datastructuren: wide vs. long format

Wanneer je alle data hebt opgeschoond en gecontroleerd kan het nog zijn dat de dataset niet de juiste *structuur* heeft. Met structuur bedoelen we hier hoe de onafhankelijk en afhankelijke variabelen en de subjectnummers etc. zijn ingedeeld. Er wordt een onderscheid gemaakt tussen het **LONG** en **WIDE** format.

WIDE FORMAT WIDE FORMAT
Wanneer een proefobject meermaals is gemeten en alle meetwaarden (afhankelijke variabelen) van de verschillende condities naast elkaar in aparte kolommen staan (dus 1 rij per proefobject) spreken we van een **WIDE** format. show example

```
##      subj        A         B        C
##         1     11.9      9.45     12.1
##         2     19.3     19.69     21.3
##         3     23.6     16.22     16.2
##         4     15.1     10.76     15.0
```

LONG FORMAT LONG FORMAT
Bij dit format staan alle meetwaarden van de verschillende condities in dezelfde kolom, met daarnaast een kolom die aanduid bij welke conditie iedere meetwaarde hoort, en een kolom die aanduid van welk proefobject de meetwaarde afkomstig is zoals geillustreerd in Fig X. show example

```
##      subj     time     score
##         1        A     11.85
##         2        A     19.34
##         3        A     23.64
##         4        A     15.12
##         1        B      9.45
##         2        B     19.69
##         3        B     16.22
##         4        B     10.76
##         1        C     12.11
##         2        C     21.27
##         3        C     16.16
##         4        C     15.00
```

inhoud

2.2 Van wide naar long

Stel je hebt 4 proefobjecten gemeten in 3 verschillende condities je data staan in het WIDE format:

```
##      subj        A         B        C
##         1     11.9      9.45     12.1
##         2     19.3     19.69     21.3
##         3     23.6     16.22     16.2
##         4     15.1     10.76     15.0
```

Code om data van WIDE format om te zetten naar LONG format:

```
# transform data from wide to long format ----
data.long =
    reshape(data        = data.wide       # naam van de oude dataset in het WIDE format
            ,direction  = 'long'          # richting van de data-transformatie
            ,varying    = c('A','B','C')  # kolomnamen die worden samengevoegd
            ,idvar      = 'subj'          # kolomnaam met de ppn-nummers
            ,v.names    = 'score'         # naam van nieuwe kolom met meetwaarden
            ,timevar    = 'time'          # naam van nieuwe kolom met condities
            ,times      = c('A','B','C')  # waardes
    )
```

Resultaat:

```
##      subj     time     score
##         1        A     11.85
##         2        A     19.34
##         3        A     23.64
##         4        A     15.12
##         1        B      9.45
##         2        B     19.69
##         3        B     16.22
##         4        B     10.76
##         1        C     12.11
##         2        C     21.27
##         3        C     16.16
##         4        C     15.00
```

inhoud

2.3 Van long naar wide

Stel je hebt 4 proefobjecten gemeten in 3 verschillende condities je data staan in het LONG format:

```
##      subj     time     score
##         1        A     11.85
##         2        A     19.34
##         3        A     23.64
##         4        A     15.12
##         1        B      9.45
##         2        B     19.69
##         3        B     16.22
##         4        B     10.76
##         1        C     12.11
##         2        C     21.27
##         3        C     16.16
##         4        C     15.00
```

Code om data van LONG format om te zetten naar WIDE format:

```
# transform data from long to wide format ----
data.wide =
reshape(data          = data.long      # naam van de oude dataset
          ,direction  = 'wide'         # richting van de data-transformatie
          ,v.names    = 'score'        # kolomnaam met de meetwaarden
          ,timevar    = 'time'         # kolomnaam met de conditienamen
          ,idvar      = 'subj'         # kolomnaam met de ppn-nummers
  )
names(data.wide)[2:4] = c('A','B','C') # pas de kolomnamen met scores aan
```

Resultaat:

```
##      subj        A         B        C
##         1     11.9      9.45     12.1
##         2     19.3     19.69     21.3
##         3     23.6     16.22     16.2
##         4     15.1     10.76     15.0
```

inhoud

---

3 Data inspecteren

In de praktijk van het wetenschappelijk onderzoek zijn de data zoals die uit een experiment komen rollen meestal niet geschikt om direct een statistische toets op uit te voeren. Vaak moeten ze eerst worden ingelezen in R vanuit een ander format (matlab, excel, .txt etc.). Controleer altijd eerst of de waardes goed zijn ingelezen. Het kan zijn dat de data eerst moet worden opgeschoond. Dat wil zeggen, je moet controleren of de gemeten waardes wel binnen een plausibel bereik vallen, dus geen onmogelijke of onwaarschijnlijke waardes bevatten. Het kan ook gebeuren dat een proefdier of proefpersoon niet altijd de taak uitvoert zoals bedoeld en er een deel van de data ontbreekt. Dan is het belangrijk in beeld te brengen hoe vaak en wanneer dat gebeurt, en deze metingen weg te laten uit de analyse, of eventueel zelf alle metingen van betreffende proefdier / -persoon weg te laten.

Daarnaast werken sommige functies alleen correct wanneer de data van het juiste type zijn (bv. *numeric, character, factor*) en dat kun je niet altijd duidelijk zien op het eerste oog (b.v.: een "5" kan door R als *character*worden gezien, en niet als een nummer waar je mee kunt rekenen). Wat ook mis kan gaan is wanneer je condities of proefpersonen aanduidt met nummers en R die getallen gebruikt om mee te rekenen i.p.v. deze te gebruiken om te weten bij welk proefobject / welke conditie een meetwaarde hoort. Het beste is om categorische data aan te duiden met letters. Gebruik bijvoorbeeld "S1","S2", .. i.p.v. "1","2",.. om proefobjecten aan te duiden, of "Conditie 1", "Conditie 2" i.p.v. "1","2" (nog beter is om informatieve namen te gebruiken zoals: "Control", "LowDose","HighDose").

Ten slotte moeten de data ook op de juiste manier gestructureerd zijn, wat kan verschillen per R functie. met de *reshape()* functie kan je de data binnen R herstructuren.

inhoud

3.1 Foutieve data opsporen

Er bestaan vele manieren om ontbrekende en foutieve data op te sporen in je dataset. In deze handleiding bespreken we het gebruik van de volgende functies:

```
Which()
is.na()
duplicated()
complete.cases()
```

De functie *which()* wordt gebruikt in combinatie met de andere functies om de positie op te vragen waar bepaalde waardes zich bevinden i.p.v. de waardes zelf.

inhoud

3.1.1 Ontbrekende data

is.nan()
In R worden ontbrekende waardes weergegeven als *NaN* (Not A Number) of *NA* (Not Available). Om deze op te sporen kun je de functies *is.na()* en *is.nan()* gebruiken.
Om de ontbrekende waardes in de dataset te vinden gebruiken we ..

```
is.na(data.bad)
```

show output

```
##        subj  time score
##  [1,] FALSE FALSE FALSE
##  [2,] FALSE FALSE  TRUE
```

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

.. en dit geeft voor ieder element uit de dataset aan of het een ontbrekende waarde is of niet met een waarde van TRUE / FALSE.

In combinatie met de functie *which()* krijgen we het volgende resultaat:

```
(naRows = which(is.na(data.bad)))
```

```
## [1] 17 28 30 33
```

Dit zijn dus de positienummers van de elementen in de dataset die geen waarde bevatten.

Meestal is het overzichtelijker om per kolom te bekijken welke rijnummers data missen. Door met indexeren alleen binnen een kolom van de dataset te kijken krijg je de rijnummers waarin data ontbreken:

```
(naRows = which(is.na(data.bad$score)))
```

```
## [1] 2 4 7
```

In rij 2, 4, 7 ontbreken dus meetwaardes in de kolom *score*.

complete.cases()
De functie *complete.cases()* geeft voor iedere rij uit de dataset aan of deze compleet is of niet (TRUE/FALSE). Een 'case' wordt alleen als compleet beschouwd wanneer er in alle kolommen waardes staan. In combinatie met de functie *which()* weet je welke rijnummers compleet zijn:

```
which(complete.cases(data.bad))
```

```
##  [1]  1  3  5  6  8  9 10 11 12 13
```

Andersom kun je door het symbool **!** voor de functie *complete.cases()* te zetten de selectie omdraaien en krijg je terug welke rijen incompleet zijn (m.a.w. welke rijen zijn NIET compleet):

```
(incomRows = which(!complete.cases(data.bad)))
```

```
## [1] 2 4 7
```

inhoud

3.1.2 Dubbel ingevoerde data

duplicated()
Deze functie controleert of er data dubbel voorkomt in het dataframe. Deze functie vergelijkt hele rijen met elkaar, dus niet of een waarde meerdere keren voorkomt binnen een kolom.

```
(dupRows = which(duplicated(data.bad)))
```

```
## [1] 13
```

inhoud

3.1.3 Extreme of foutieve waardes

Soms komt het voor dat er foutieve waardes in je dataset terecht komen, of je wilt een subset van waardes selecteren om mee verder te werken. Stel dat in deze dataset alleen scores van 1 t/m 20 mogelijk zijn dan kan je alsvolgt controleren of er waardes zijn die daar buiten vallen en welke dat zijn:

```
minScore = 1
maxScore = 20
(data.bad$score < minScore | data.bad$score > maxScore)
```

```
##  [1] FALSE    NA  TRUE    NA FALSE FALSE    NA FALSE  TRUE  TRUE FALSE
## [12] FALSE FALSE
```

Wederom kan je de functie *which()* gebruiken om op te vragen welke waardes dat precies zijn. Merk op dat de NA waardes niet worden genegeerd.

```
(extremeRows = which((data.bad$score < minScore | data.bad$score > maxScore)))
```

```
## [1]  3  9 10
```

inhoud

3.2 Data types controleren

R ken verschillende soorten data types zoals o.a. *numeric* (numerieke waardes) *character* (tekst), *logical* (TRUE / FALSE) en *factor* (categorische waardes). Als de data niet als het juiste type is opgeslagen in R zullen de functies een error geven, of (nog gevaarlijker) een verkeerde output geven. Raadpleeg altijd de help-files van de functies voor welk data type ze als input verwachten.

Functies om het datateyp te **controleren**:

```
is.numeric()
is.character()
is.logical()
is.factor()
```

Functies om het data type te **veranderen**:

```
as.numeric()
as.character()
as.logical()
as.factor()
```

Voorbeeld tekst naar nummerieke waardes:

```
numbers = c("1","2","3","4")     # variabele met getallen 1 t/m 4
is.numeric(numbers)              # FALSE: variabele is niet numeriek
is.character(numbers)            # TRUE:  variabele is tekst
numbers = as.numeric(numbers)    # verandert variabele van tekst naar numeriek
is.numeric(numbers)              # TRUE: variabele is numeriek
```

In bovenstaand voorbeeld wordt een variabele aangemaakt met getallen, maar omdat ze tussen aanhalingstekens staan wordt het door R gezien als tekst i.p.v. numerieke waardes. Met de functie *as.numeric()* wordt de tekst omgeschreven naar numerieke waardes.

Voorbeeld numerieke waardes naar categorische variabelen:

```
conditions = rep(x=c(1,2,3),times=5)      # variabele die conditie (1 t/m 3) codeert
is.numeric(conditions)                    # TRUE: variabele is numeriek
mean(conditions)                          # R kan hier mee rekenen, maar betkent niets
conditions = as.factor(conditions)        # verandert variabele naar categorisch (factor)
is.numeric(conditions)                    # FALSE: variabele is een factor
mean(conditions)                          # NA: met deze variabele kan R niet rekenen
```

In bovenstaand voorbeeld zijn de condities gecodeerd met 1 t/m 3. De cijfers 1 t/m 3 geven hier alleen aan tot welke conditie iets hoort, maar het getal geeft geen quantitatieve informatie. wanneer je per ongeluk de data verkeerd invoert in een functie kan R hier toch mee gaan rekenen en krijg je een onzinnige output zonder dat R een waarschuwing geeft. Door er een *factor* van te maken geef je expliciet aan dat het niet gaat om quantitatieve waardes maar om benaming.

inhoud

---

4 Data opschonen

Hiervoor hebben we besproken hoe je foutieve en ontbrekende data kunt opsporen. De volgende stap is om je dataset op te schonen. Je kunt dit op 2 manieren doen: 1) een-voor-een verschillende soorten foutieve data opsporen en verwijderen, of 2) alle soorten foutieve data opsporen en die in een keer verwijderen. Hieronder twee goede voorbeelden van code om data op te schonen, en een veelvoorkomende foute manier.

Methode 1:

```
(naRows = which(is.na(data.bad$score)))  # identificeer ontbrekende waardes
## [1] 2 4 7
data.bad = data.bad[-naRows,]            # verwijder rijen

(dupRows = which(duplicated(data.bad)))  # identificeer gedupliceerde rijen
## [1] 10
data.bad = data.bad[-dupRows,]           # verwijder rijen

(incomRows = which(!complete.cases(data.bad))) # identificeer incomplete rijen
## integer(0)
# niets gevonden..

# identificeer scores buiten het bereik 1 t/m 20
minScore = 1
maxScore = 20
(extremeRows = which((data.bad$score < minScore | data.bad$score > maxScore)))
## [1] 2 6 7
data.bad = data.bad[-extremeRows,]      # verwijder rijen
```

Resultaat:

```
##      subj     time     score
##         1        A     11.85
##         1        B      9.45
##         2        B     19.69
##         4        B     10.76
##         3        C     16.16
##         4        C     15.00
```

Methode 2:

```
(naRows = which(is.na(data.bad$score)))        # identificeer ontbrekende waardes
## [1] 2 4 7
(dupRows = which(duplicated(data.bad)))        # identificeer gedupliceerde rijen
## [1] 13
(incomRows = which(!complete.cases(data.bad))) # identificeer incomplete rijen
## [1] 2 4 7
# identificeer scores buiten het bereik 1 t/m 20
minScore = 1
maxScore = 20
(extremeRows = which((data.bad$score < minScore | data.bad$score > maxScore)))
## [1]  3  9 10
# creeer een variabele met alle unieke rijnummers die verwijderd moeten worden
(badRows = unique(c(naRows, extremeRows, dupRows)))
## [1]  2  4  7  3  9 10 13
data.clean = data.bad[-badRows,] # verwijder alle rijen in een keer
```

Resultaat:

```
##      subj     time     score
##         1        A     11.85
##         1        B      9.45
##         2        B     19.69
##         4        B     10.76
##         3        C     16.16
##         4        C     15.00
```

Hoe niet:

```
# BAD CODE !!
(naRows = which(is.na(data.bad$score)))        # identificeer ontbrekende waardes
## [1] 2 4 7
(dupRows = which(duplicated(data.bad)))        # identificeer gedupliceerde rijen
## [1] 13
(incomRows = which(!complete.cases(data.bad))) # identificeer incomplete rijen
## [1] 2 4 7
# identificeer scores buiten het bereik 1 t/m 20
minScore = 1
maxScore = 20
(extremeRows = which((data.bad$score < minScore | data.bad$score > maxScore)))
## [1]  3  9 10

# verwijder om de beurt de verschillende foutieve rijen
data.bad = data.bad[-naRows,]
data.bad = data.bad[-extremeRows,]
data.bad = data.bad[-dupRows]
data.bad = data.bad[-incomRows,]
# BAD CODE !!
```

Resultaat:

```
##        subj     time     score
## 1         1        A      11.9
## 6         2        B      19.7
## 9         1        C      35.9
## 10        2        C      21.3
```

Hier zie je dat de NAs wel uit de dataset zijn verwijderd, maar er zitten nog wel extreme waardes tussen, en andere, valide waardes zijn wel verwijderd. Dit komt doordat bij het een-voor-een verwijderen van de foutieve data de rijnummers in de dataset telkens opschuiven maar de identificatie van die rijen, die daarvoor was gedaan, niet tussendoor aangepast.

inhoud
