R - data verwerking

J.J. van Nijnatten

Contents

Generate data

Data inspecteren

Data inspecteren

In dit deel wordt onderstaande dataset, genaamd *badData* gebruikt om voorbeelden te geven.
Code om *badData* te genereren

```
badData = data.long
badData[c(4,7),3] = NA
badData[4,2] = NA
badData[2,3] = NaN
badData[9,3] = 35.90
badData[13,] = badData[12,]
rownames(badData) = NULL
```

```
##        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
```

Foutieve data opsporen

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()
is.nan()
is.null()
duplicated()
complete.cases()
```

Ontbrekende data

Ontbrekende data

In R worden ontbrekende waardes weergegeven als *NaN* (Not A Number), *NA* (Not Available) of *NULL*. Om deze op te sporen kun je de functies *is.na(), is.nan(), is.null()* gebruiken. In dit voorbeeld staan er missende waardes als NA in de dataset:

```
(naRows = which(is.na  (badData$score)))
```

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

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

Een andere optie is om specifiek te zoeken naar proefobjecten waarvan de gegevens (in)compleet zijn. 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.

```
(complRows = complete.cases(badData))
```

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

In combinatie met de functie *which()* weet je welke rijnummers compleet zijn:

```
which(complete.cases(badData))
```

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

Andersom kun je met het symbool *!* de selectie omdraaien en krijg je terug welke rijen incompleet zijn:

```
(incomRows = which(!complete.cases(badData)))
```

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

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(badData)))
```

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

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
(badData$score < minScore | badData$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((badData$score < minScore | badData$score > maxScore)))
```

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

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:

```
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.

Data opschonen

Wanneer je foutieve waardes in je dataset hebt opgespoord kun je twee dingen doen: 1) De proefobjecten met die foutieve waardes verwijderen uit je dataset, of 2) alleen de correcte data opslaan in een nieuwe dataset.

Verwijderen van foutieve datapunten

Als je weet welke rijen in je dataset waardes bevatten die je wilt verwijderen kan dat alsvolgt:

```
naRows = which(is.na(badData$score)) # rij-nummers met incomplete data
badData = badData[-naRows,] # selecteer alles behalve incomplete rijen
```

Let op! wanneer je de rijnummers bewaart met slechte data, zorg dan dat je alles in een keer verwijdert. Stel dat je eerst de incomplete rijen verwijdert, en daarna de extreme waardes, dan zullen na de eerste keer verwijderen de rijnummers opschuiven en gooi je bij de volgende stap mogelijk de vereerde data weg.

Dus niet zo

```
# BAD CODE !!
badData = badData[-dupRows]
badData = badData[-extremeRows,]
badData = badData[-incomRows,]
badData = badData[-naRows,]
# BAD CODE !!
```

```
##      subj     time     score
##         1        A      11.9
##         2        B      19.7
##         3        C      16.2
##         4        C      15.0
```

maar zo

```
# maak verzameling met unieke rij-nummers met foute data.
badRows = unique(c(incomRows, naRows, extremeRows, dupRows))
# verwijder alle rijen in een keer om te voorkomen dat rijen opschuiven
badData = badData[-badRows,]
```

```
##      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
```

Let hierbij op het min-teken (*-}) wat aangeeft dat je alles behalve die data selecteert.

Selecteren van correcte data (indexing)

Correcte data opslaan in nieuwe dataset

Herstructureren van data

In de praktijk van het wetenschappelijk onderzoek worden experimenten vaak uitgevoerd met andere apparatuur en computers dan waar je de uiteindelijke statistische analyse gaat uitvoeren. De data zoals die uit het experiment komen rollen zijn meestal niet geschikt om direct een statistische toets op uit te voeren. 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").

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 Wanneer een proefobject meermaals hebt 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 zoals geillustreerd in Fig X.

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.

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
```

Om de data te herstructureren naar het LONG format kun je de *reshape()* functie alsvolgt gebruiken:

```
# 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
    )
```

```
##      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
```

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
```

Om de data te herstructureren naar het WIDE format kun je de *reshape()* functie alsvolgt gebruiken:

```
# transform data from long to wide format ----
data.wide =
reshape(data          = data.long      # naam van de 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
```

```
##      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
```