

R - data wrangling

J.J. van Nijnatten

Contents

Generate data	3
Data cleaning	4
foutieve datapunten identificeren (missing data, extreme values, wrong data types)	4
Verwijderen van foutieve datapunten	5
Selecteren van correcte data (indexing)	5
dubbele datainvoer	5
Correcte data opslaan in nieuwe dataset	5
Herstructuren van data	6
datastructuren wide vs. long format	6
van wide naar long	7
van long naar wide	8

Generate data

```
nrofsubs = 4
nrofconds = 3
subj = rep(1:nrofsubs,nrofconds)
time = rep(LETTERS[1:nrofconds],each=nrofsubs)
score = as.vector( replicate(
  nrofconds , rnorm(n = nrofsubs, mean = sample(8,1)+10 , sd = sample(5,1) )
) )
data.long = data.frame(subj, time, score);
rm(list=setdiff(ls(), c("data.long", "nrofsubs","nrofconds")))
```

##	subj	time	score
##	1	A	12.29
##	2	A	15.36
##	3	A	9.57
##	4	A	1.02
##	1	B	14.47
##	2	B	16.51
##	3	B	13.41
##	4	B	16.43
##	1	C	9.54
##	2	C	13.58
##	3	C	4.44
##	4	C	12.38

##	subj	A	B	C
##	1	12.29	14.5	9.54
##	2	15.36	16.5	13.58
##	3	9.57	13.4	4.44
##	4	1.02	16.4	12.38

Data cleaning

```
##      subj      time      score
##      1         A      12.29
##      2         A      15.36
##      3         A         NA
##      4         A       1.02
##      1         B         NA
##      2         B      16.51
##      3         B      13.41
##      4         B      16.43
##      1         C      90.35
##      2         C      13.58
##      3         C       4.44
##      4         C      12.38
##      4         C      12.38
```

foutieve datapunten identificeren (missing data, extreme values, wrong data types)

Missing Data

De functie `complete.cases()` geeft voor iedere rij uit de dataset aan of deze compleet is of niet (TRUE/FALSE). In combinatie met de functie `which()` weet je welke rijnummers compleet zijn:

```
complete.cases(messyData)
```

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

```
which(complete.cases(messyData))
```

```
## [1] 1 2 4 6 7 8 9 10 11 12 13
```

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

```
which(!complete.cases(messyData))
```

```
## [1] 3 5
```

Dubbele data

De functie `duplicated()` 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.)

```
which(duplicated(messyData))
```

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

Extreme of foutieve waardes

Verwijderen van foutieve datapunten

Selecteren van correcte data (indexing)

dubbele datainvoer

Correcte data opslaan in nieuwe dataset

Herstructuren 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 waarden wel binnen een plausibel bereik vallen, dus geen onmogelijke of onwaarschijnlijke waarden bevatten. Het kan ook gebeuren dat een proefdier of proefpersoon niet altijd de taak uitvoert zoals bedoeld en er een deel van de data mist. 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, dus “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 onafhankelijke 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 geïllustreerd in Fig X.

LONG FORMAT Bij dit format staan alle meetwaarden van de verschillende condities in dezelfde kolom, met daarnaast een kolom die aanduidt bij welke conditie iedere meetwaarde hoort, en een kolom die aanduidt van welk proefobject de meetwaarde afkomstig is zoals geïllustreerd 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     12.29    14.5    9.54
##      2     15.36    16.5   13.58
##      3      9.57    13.4    4.44
##      4      1.02    16.4   12.38
```

Om de data te herstructureren naar het LONG format kun je de `reshape()` functie als volgt 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') # waarden
  )
```

```
##      subj      time      score
##      1         A     12.29
##      2         A     15.36
##      3         A      9.57
##      4         A      1.02
##      1         B     14.47
##      2         B     16.51
##      3         B     13.41
##      4         B     16.43
##      1         C      9.54
##      2         C     13.58
##      3         C      4.44
##      4         C     12.38
```

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      12.29
##      2         A      15.36
##      3         A       9.57
##      4         A       1.02
##      1         B      14.47
##      2         B      16.51
##      3         B      13.41
##      4         B      16.43
##      1         C       9.54
##      2         C      13.58
##      3         C       4.44
##      4         C      12.38
```

Om de data te herstructureren naar het WIDE format kun je de `reshape()` functie als volgt 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     12.29    14.5    9.54
##      2     15.36    16.5   13.58
##      3      9.57    13.4    4.44
##      4      1.02    16.4   12.38
```