Statistics: Analyzing in 

Bernd Figner
b.figner@psych.ru.nl

Week 3: Nov 23, 2015

1

This week

- Some recap and last week's leftovers
- Data visualization with `qplot()` and `ggplot()` from package `ggplot2`

2

Recap and leftovers

`which()` → **Can be used to select specific entries, e.g.:**

What does the following command do?

```
which(r_1$rating_1 == 0)
```

→ returns the **index numbers** of entries that are 0

```
which(r_1$rating_1 != 0)
```

→ returns the index numbers of entries that are NOT 0

```
which(r_1$rating_1 > 5)
```

→ returns the index numbers of entries whose value is larger than 5

```
which(r_1$pp_code == "pp_3")
```

→ returns the index numbers of entries with pp_code pp_3

4

Thus...

What is the following doing?

```
r_1[which(r_1$rating_1 == 0 & r_1$pp_code ==
“pp_3”),]
```

Returns all columns of the data frame r_1,
but only those rows that:

- have a value of 0 in the variable rating_1 AND
- belong to participant pp_3

5

subset()

Can select specific rows and/or specific columns

```
?subset()
```

```
r_1_f <- subset(r_1, subset = f_gender == 'female')
```

```
r_1_f <- subset(r_1, f_gender == 'female')
```

→ r_1_f only female participants

BUT

```
r_1_f2$f_gender
[1] female female female female female female
Levels: female male
```

6

To remove unused factor levels, use `droplevels()`

```
r_1_f <- droplevels(subset(r_1, subset = f_gender == 'female'))
```

```
r_1_f <- droplevels(subset(r_1, f_gender == 'female'))
```

NOW

```
r_1_f2$f_gender
[1] female female female female female female
Levels: female
```

7

How to select **columns**

```
r_1_r1_2 <- subset(r_1, select = c(pp_code, rating_1, rating_2))
```

→ new data frame with only columns pp_code, rating_1, rating_2

To remove data (rows and/or columns)

```
r_1_r1_2 <- subset(r_1, gender != "male", select = -c(2,4))
```

Selection of rows and columns can be done at the same time and can be combined with logical and & and logical or |

8

Data frame diamonds from package ggplot2

```
head(diamonds)
```

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

9

Select rows with Ideal, Very Good, and Good cuts

```
diamonds_2 <- droplevels(subset(diamonds,
cut == "Ideal" | cut == "Very Good" | cut
== "Good"))
```

Another option

```
diamonds_2 <-
droplevels(diamonds[diamonds$cut %in%
c("Ideal", "Very Good", "Good"),])
```

10

t tests

Different ways to do t tests, e.g.

`t.test()` → from the base package

Wide Format

```
t.test(df$Var1, df$Var2, paired = TRUE)
```

Long Format

```
t.test(df$DV ~ df$IV, paired = TRUE)
```

~ "as a function of"

Analyze the dependent variable as a function of the independent variable (e.g., rating 1 vs 2; gender; ...)

More info: FMF or http://www.statmethods.net/stats/ttest.html

11

`round()` → **To round numbers; for example**

```
round(MyVariable, digits = 0)
```

→ rounds the values in the variable "MyVariable" to integers (i.e., no decimals)

→`digits = 2` means two decimals etc

12

Reshaping Data

- `stack()`, `unstack()`
- `melt()`, `cast()` from library `reshape`
- `reshape()`

`stack()`, `unstack()` → FMF page 110/111

Only for simple data; not recommended, really

13

`melt()`, `cast()`

```
data2 <- melt(data1, id = c(constant variables), measured
= c(variables changing among columns))
```

for example with repeated measures

```
data2 <- melt(data1, id = c("pp_code", "gender", "age"),
measured = c("trial1", "trial2", "trial3", "trial4"))
```

and from long to wide format

```
data3 <- cast(data2, pp_code + gender + age ~ variable)
```

in simple cases, this also works:

```
data3 <- cast(data2)
```

14

reshape()

```
data2 <- reshape(data1, idvar = 'pp_code', varying =
c('rating_1', 'rating_2', 'rating_3'), timevar =
'rating_1or2or3', v.names = 'rating', direction = 'long')
```

- `idvar` variable indicating "who is who" (typically the participant code)
- `varying` the columns we want to stack
- `timevar` the variable that will indicate—in the new stacked data frame—from which original column the data point came
- `v.names` the name of the new stacked variable (in our example the ratings 1, 2, 3)
- `direction` whether the reshape is from wide to **long** or from long to **wide** (you specify how it should look *after* it has been reshaped, i.e., what you want, not what you have)

15

Reshaping is extremely important!

- Most analyses in R need a data frame in long format

→ Invest enough time to thoroughly learn how to change from wide to long format (and back)

16

Ordering entries

After reshaping from wide to long and back, rows in data frame might be ordered differently:

In original data frame: `pp_1, pp_2, pp_3, etc`

After reshaping: `pp_1, pp_10, pp_2, pp_3, etc`

Order of rows typically doesn't matter; so no reason to change it.

If you want to order entries: `order()` (do NOT use `sort()`)

```
r_1b <- r_1[order(r_1[,3]),]
```

17

Ordering entries

After reshaping from wide to long and back, rows in data frame might be ordered differently:

In original data frame: `pp_1, pp_2, pp_3, etc`

After reshaping: `pp_1, pp_10, pp_2, pp_3, etc`

Order of rows typically doesn't matter; so no reason to change it.

If you want to order entries: `order()` (do NOT use `sort()`)

```
r_1b <- r_1[order(r_1[,3]),]
```

18

Ordering entries

After reshaping from wide to long and back, rows in data frame might be ordered differently:

In original data frame: `pp_1, pp_2, pp_3, etc`

After reshaping: `pp_1, pp_10, pp_2, pp_3, etc`

Order of rows typically doesn't matter; so no reason to change it.

If you want to order entries: `order()` (do NOT use `sort()`)

```
r_1b <- r_1[order(r_1[,3]),]
```

19

Ordering entries

After reshaping from wide to long and back, rows in data frame might be ordered differently:

In original data frame: `pp_1, pp_2, pp_3, etc`

After reshaping: `pp_1, pp_10, pp_2, pp_3, etc`

Order of rows typically doesn't matter; so no reason to change it.

If you want to order entries: `order()` (do NOT use `sort()`)

```
r_1b <- r_1[order(r_1[,3]),]
```

20

Ordering entries

After reshaping from wide to long and back, rows in data frame might be ordered differently:

In original data frame: `pp_1, pp_2, pp_3, etc`

After reshaping: `pp_1, pp_10, pp_2, pp_3, etc`

Order of rows typically doesn't matter; so no reason to change it.

If you want to order entries: `order()` (do NOT use `sort()`)

```
r_1b <- r_1[order(r_1[,3]),]
```

Actually, doesn't help in the case of `pp_1` etc...

You could use `pp_01`, `pp_02`, etc to solve that specific reordering problem.

21

Questions?

22