psych: A general purpose toolkit for personality and psychological Research

October 28, 2016

Format of the current workshop

I will present several key functions. For each function, I will present:

- The reason or motivation for using the function
- How to use the function (key arguments, etc)
- An example of the output
- Tips
- Other relevant functions

Functions

- describe
- scrub
- pairs.panels
- corr.test
- scoreItems
- alpha

Preliminary steps - install and load

To install psych:

```
install.packages("psych")
```

To load psych:

```
library(psych)
```

Preliminary steps - Load two fake data sets

```
data(bfi)
data(iris)
```

If you type 'bfi' into your console, you can now see the data.

```
bfi
```

describe() - Motivation

Motivation: calculate descriptive statistics for all or part of your data set.

Descriptive statistics include:

- n - number of participants who have non-missing data
- mean
- standard deviation (unbiased)
- median
- mean absolute deviation
- minimum value
- maximum value
- range
- skew
- kurtosis
- standard error of the mean

describe() - Use

```
describe(x = bfi) # the data set
```

You can also put part of the data set in here, like only some of the variables or some of the rows.

```
describe(x = bfi[,c("gender","education","age")])
```

```
describe(x = bfi[1:50, ])
```

Finally, you can speed up the function by showing only some of the variables.

```
describe(x = bfi,
         fast=TRUE) # argument to calculate limited number
```

describe() - example

```
##    vars    n mean   sd min max range   se
## A1    1 2784 2.41 1.41   1   6     5 0.03
## A2    2 2773 4.80 1.17   1   6     5 0.02
## A3    3 2774 4.60 1.30   1   6     5 0.02
## A4    4 2781 4.70 1.48   1   6     5 0.03
## A5    5 2784 4.56 1.26   1   6     5 0.02
```

describe() - tips

describe will calculate statistics for factor-level variables if those levels are represented with numbers. It *will* warn you though.

```
describe(iris, fast=T)
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments
## Inf

## Warning in FUN(newX[, i], ...): no non-missing arguments
## -Inf

##                vars   n mean   sd min  max range   se
## Sepal.Length      1 150 5.84 0.83 4.3  7.9   3.6 0.07
## Sepal.Width       2 150 3.06 0.44 2.0  4.4   2.4 0.04
## Petal.Length      3 150 3.76 1.77 1.0  6.9   5.9 0.14
## Petal.Width       4 150 1.20 0.76 0.1  2.5   2.4 0.06
## Species*          5 150  NaN   NA Inf -Inf  -Inf   NA
```

describe() - related functions

describeBy() calculates all statistics separately for each group.

```
describeBy(x = iris, # full data set
           group=iris$Species) # vector that holds
                               # group assignment
```

scrub() - motivation

scrub() is designed to help you clean your data. You can use it to remove values from a variable (for example, if 9999 means missing, remove all instances of 9999) or change one value to another.

scrub - use

```
cleaned <- scrub(x = bfi,  #data set
                 where = "age", #variables to be changed
                 isvalue = 74, # what values to look for
                 newvalue = 64) # what value to change tha
```

scrub - use

If you leave "newvalue" blank, it will default to NA.

```
clean <- scrub(x = bfi,  #data set
               where = "age", #variables to be changed
               isvalue = 74)
```

Another option is to set the minimum and/or maximum possible values.

```
clean <- scrub(x = bfi,  #data set
               where = "age", #variables to be changed
               min = 16, max = 70) # possible range
```

scrub - example

```
bfi$age
```

```
## [1] 16 18 17 17 17 21 18 19
```

```
scrub(x = bfi, where = "age", isvalue = 17)
```

```
## [1] 16 18 NA NA NA 21 18 19
```

```
scrub(x = bfi, where = "age", isvalue = 17, newvalue=77)
```

```
## [1] 16 18 77 77 77 21 18 19
```

scrub- tips

- scrub only takes a data frame and outputs a data frame. Make sure you name your output. I often replace the original data with the cleaned data:

```
bfi <- scrub(bfi, where="age", isvalue=17)
```

- the argument "where" can take either names or locations of variables. If you use names, make sure you put quotes around them.
- scrub will not always work if you ask it to replace a value that does not exist.

scrub() - other functions

reverse.code() will reverse code items in a scale. Requires a key vector.

pairs.panels - motivation

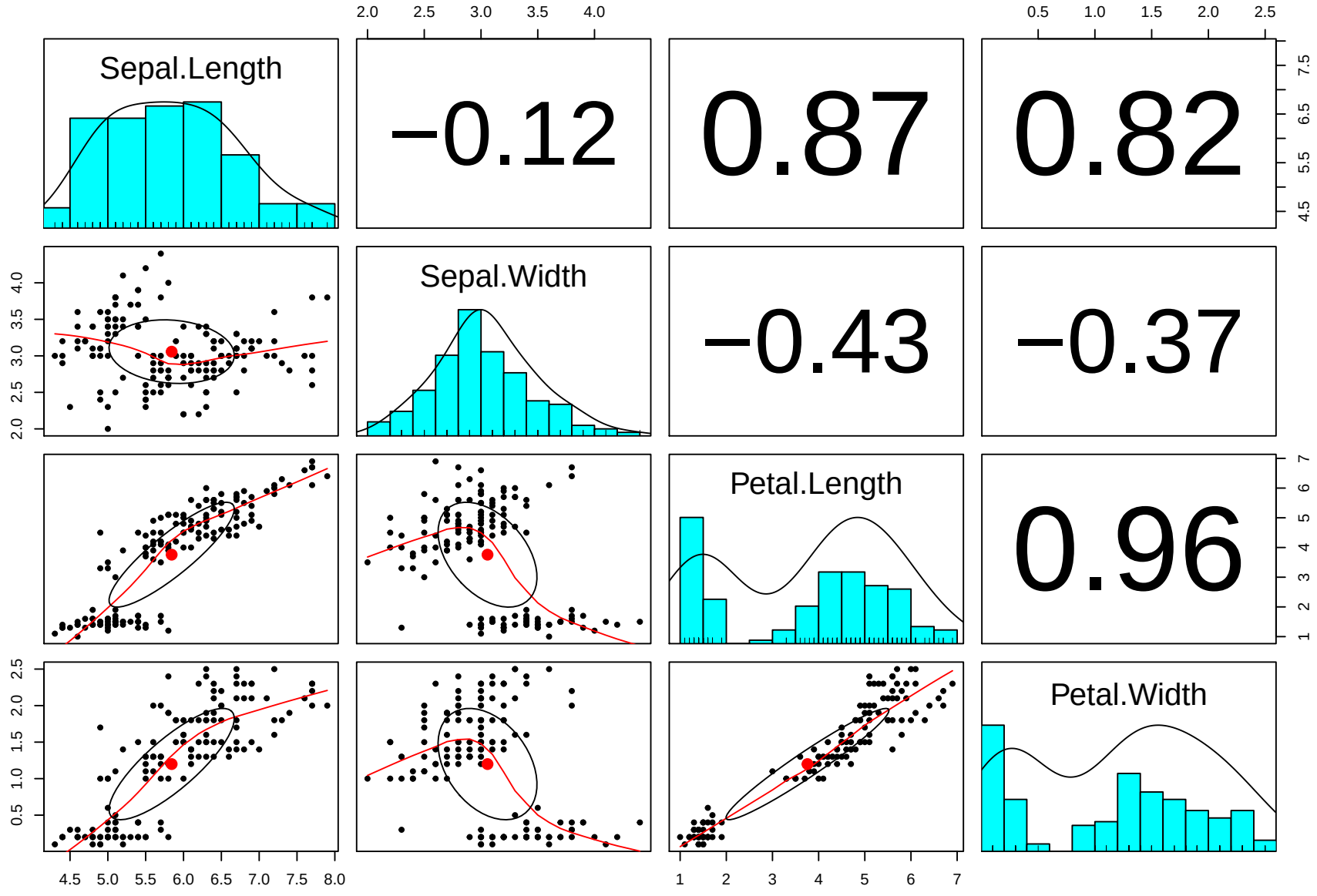
Scatter Plot of Matrices

Graphically display several variables and their relationships to one another. Great for posters!

pairs.panels - use

```
pairs.panels(x = iris[,1:4]) # data set
```

pairs.panels - example



pairs.panels - tips

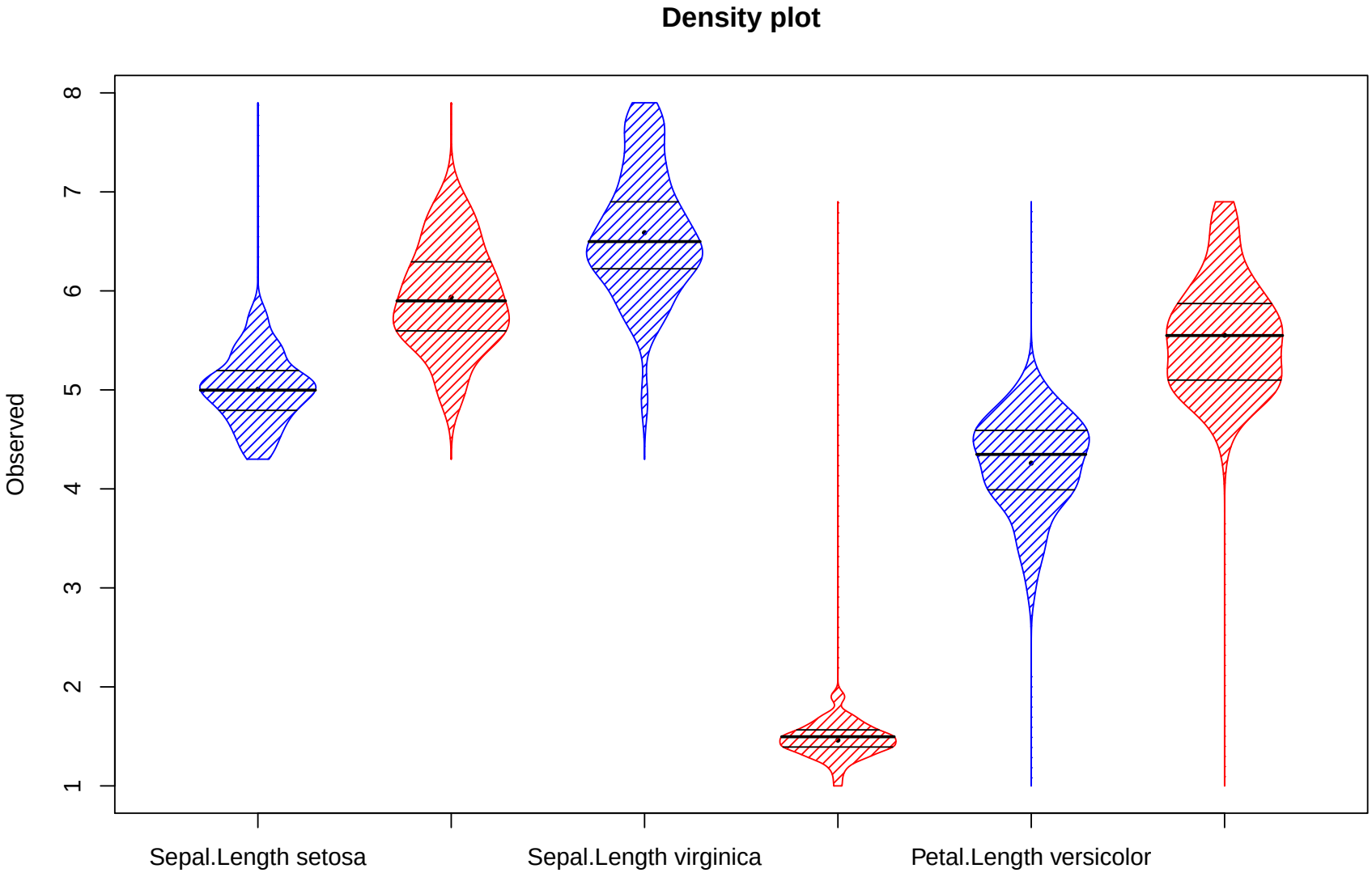
- This function is quickly overwhelmed. Limit each plot to only a few variables. I recommend no more than 8.
- Don't use binary or factor-level variables. The plots are difficult to interpret, and there are easier ways to display those data and their relationships.

pairs.panels - other functions

- violinBy is a great way to show the relationship between several continuous variable and a factor variables.

```
violinBy(x = iris[,c(1,3)], # continuous variables
         grp=iris$Species, # factor variable
         grp.name = unique(iris$Species)) # names of level
```

pairs.panels - other functions



corr.test - motivation

Calculate the correlations between a set of variables.

Correct for multiple comparisons.

corr.test - use

```
corr.test(x = bfi, # first set of variables
          y = NULL, #second set of variables
          use = "pairwise", # how to deal with missing
          method="pearson", # correlation
          adjust = "holm", # for multiple comparisons
          alpha = .05) # alpha level
```

Most of these are unncessary.

```
corr.test(x = bfi)
```

corr.test() - example

corr.test() has three main output objects:

- correlation
- sample size
- pvalue

We will look at them one at a time. Note: I show only the 5 rows and 5 columns for space.

corr.test() - example correlation

```
##       A1    C2    E3    N2    O1
## A1  1.00  0.02 -0.05  0.14  0.01
## C2  0.02  1.00  0.15 -0.01  0.16
## E3 -0.05  0.15  1.00 -0.07  0.33
## N2  0.14 -0.01 -0.07  1.00 -0.05
## O1  0.01  0.16  0.33 -0.05  1.00
```

corr.test() - example correlation sample size

```
##       A1   C2   E3   N2   O1
## A1 2784 2761 2760 2763 2762
## C2 2761 2776 2757 2756 2759
## E3 2760 2757 2775 2754 2757
## N2 2763 2756 2754 2779 2757
## O1 2762 2759 2757 2757 2778
```

corr.test() - example correlation significance

```
##       A1   C2 E3   N2   O1
## A1 0.00 1.00  1 0.00 1.00
## C2 0.39 0.00  0 1.00 0.00
## E3 0.01 0.00  0 0.05 0.00
## N2 0.00 0.64  0 0.00 0.95
## O1 0.51 0.00  0 0.01 0.00
```

corr.test() - tips

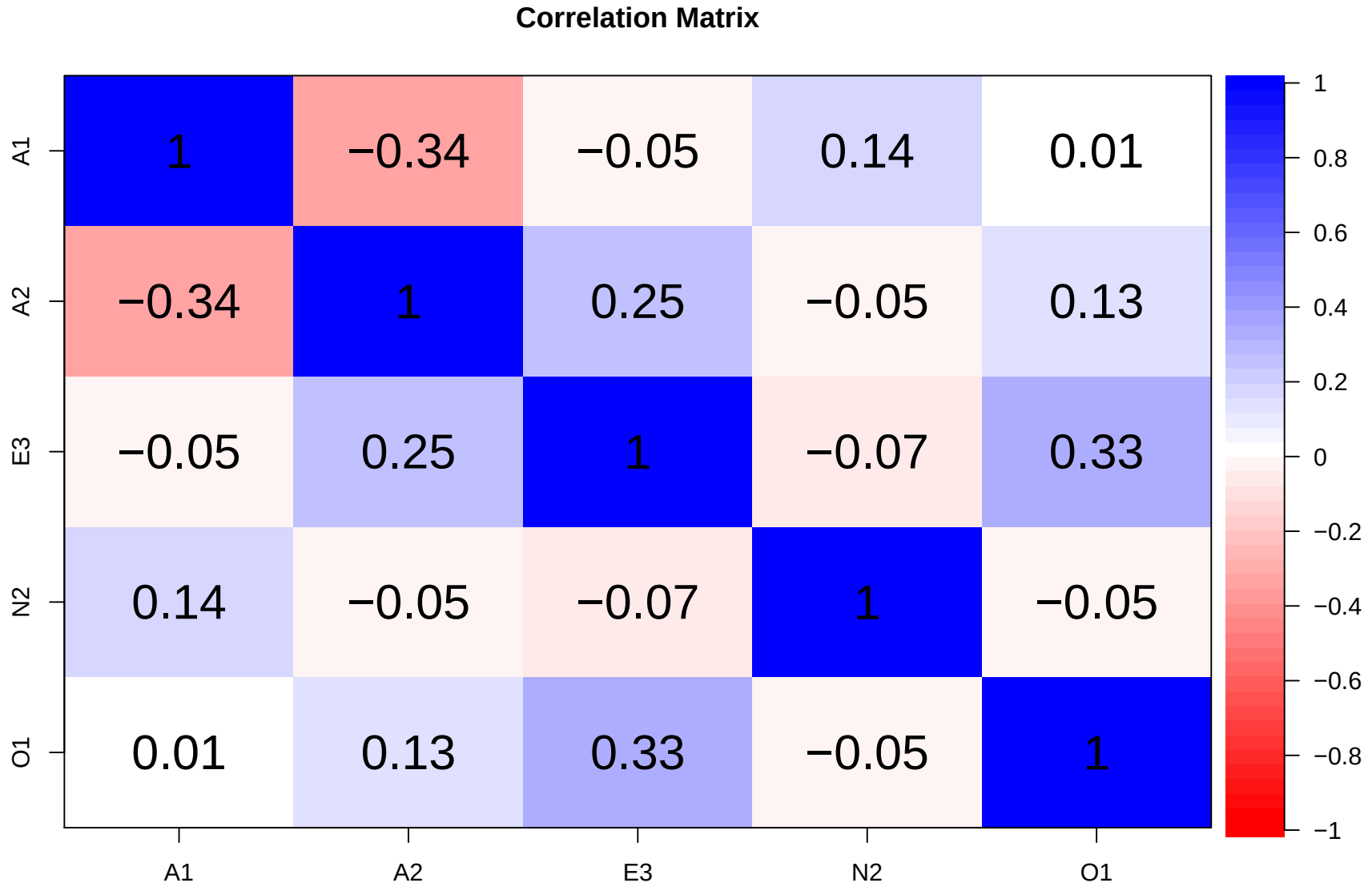
- Only use numeric variables.
- Print with short=F option to also see the bootstrapped confidence intervals!

```
R <- corr.test(bfi)
print(R, short=F)
```

corr.test() - other functions

- lowerCor displays the lower half of a correlation matrix.
- cor.plot creates a visual display of a correlation matrix.

corr.test() - other functions



scoreItems - motivation

Set of functions that score scales and provide statistics for those scales.

Steps:

- make scoring key
- score items
- view item and scale statistics

scoreItems - use

First, make a list. You can use the names of the variables or the locations. Put a - before the name of the variable to indicate that it is reverse scored.

```
keys.list <- list(agree=c("-A1","A2","A3","A4","A5"),
                  con=c("C1","C2","C3","-C4","-C5"),
                  extra=c("-E1","-E2","E3","E4","E5"),
                  neur=c("N1","N2","N3","N4","N5"),
                  open = c("O1","-O2","O3","O4","-O5"))
```

scoreItems - use

Then, use the make.keys() function to turn the list into keys.

```
keys <- make.keys(bfi, keys.list = keys.list)
```

scoreItems - use

Use the scoreItems function to do. . . you know what.

```
scores <- scoreItems(keys = keys, items=bfi)
```

scoreItems - example

```
## Call: scoreItems(keys = keys, items = bfi)
##
## (Unstandardized) Alpha:
##       agree  con extra neur open
## alpha   0.7 0.72  0.76 0.81  0.6
##
## Standard errors of unstandardized Alpha:
##       agree   con extra  neur  open
## ASE   0.014 0.014 0.013 0.011 0.017
##
## Average item correlation:
##           agree  con extra neur open
## average.r  0.32 0.34  0.39 0.46 0.23
##
##  Guttman 6* reliability:
##          agree  con extra neur open
## Lambda.6   0.7 0.72  0.76 0.81  0.6
##
```

scoreItems - example

To see even more, print with the short option turned off.

```
print(scores, short=F)
```

```
## Call: scoreItems(keys = keys, items = bfi)
##
## (Unstandardized) Alpha:
##       agree  con extra neur open
## alpha   0.7 0.72  0.76 0.81  0.6
##
## Standard errors of unstandardized Alpha:
##       agree   con extra  neur  open
## ASE   0.014 0.014 0.013 0.011 0.017
##
## Average item correlation:
##           agree  con extra neur open
## average.r  0.32 0.34  0.39 0.46 0.23
##
```

scoreItems - example

Scores are saved in the scores part of the object

```
scores$scores
```

To add them to your data set:

```
bfi <- cbind(bfi, scores$scores)
```

scoreItems - tips

- Double and triple check your keys list. They're easy to mess up.
- If your dataset has factor-level variables, scoreItems will not work at all. Even if those variables are not part of your scales. The easiest fix is to copy the scale items into a new data set, and perform the functions on that data set.

```
scaleItems <- bfi[,1:25]
```

scoreItems - other functions

- scoreOverlap is useful if you have scales which share items. This function can report the correlations between scales corrected for item overlap.

alpha - motivation

Calculate Cronbach's alpha (and other measures of reliability) for a set of items.

alpha - use

```
alpha(x = bfi[,1:5], # variables in scale
      check.keys = T) # automatically reverse score items
```

alpha - example

```
## Warning in alpha(x = bfi[, 1:5], check.keys = T): Some i
##  This is indicated by a negative sign for the variable n

##
## Reliability analysis
## Call: alpha(x = bfi[, 1:5], check.keys = T)
##
##   raw_alpha std.alpha G6(smc) average_r S/N   ase mean
##         0.7      0.71    0.68      0.33 2.5 0.009  4.7 0
##
##  lower alpha upper     95% confidence boundaries
## 0.69 0.7 0.72
##
##  Reliability if an item is dropped:
##      raw_alpha std.alpha G6(smc) average_r S/N alpha se
## A1-      0.72      0.73    0.67      0.40 2.6   0.0087
## A2       0.62      0.63    0.58      0.29 1.7   0.0119
## A3       0.60      0.61    0.56      0.28 1.6   0.0124
```

alpha - tips

- Only numeric variables.
- Only the items that are in scale itself.
- Double check all items to see if the things that should be reverse scored actually are.

alpha - other functions

Other measures of reliability. All of these have extensive documentation, including suggestions about which to use.

- omega (there's a lot of documentation about why this is better than alpha)
- guttman
- splitHalf

Other uses

The psych package has many many more tools available for both data analysis and study design.

- factor analysis and principal components analysis
- bootstrapped mediation analyses
- item response theory
- inter-rater reliability
- block random assignment generation
- practice data sets