Item Analysis

Rachael Smyth and Andrew Johnson

October 1, 2015

Load Libraries

We'll need the `psych` package for the analyses in this demonstration.

```
library(psych)
```

When you go to use this package in your publications, you should reference the package, as this gives your readers an idea as to the version of the package used, and also gives credit to the authors of the package.

You can get the citation for any package using the `citation` function. So... to get the citation information for the `psych` package, we would do the following:

```
citation("psych")
```

```
##
## To cite the psych package in publications use:
##
##   Revelle, W. (2017) psych: Procedures for Personality and Psychological
##   Research, Northwestern University, Evanston, Illinois, USA,
##   https://CRAN.R-project.org/package=psych Version = 1.7.5.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {psych: Procedures for Psychological, Psychometric, and Personality Research},
##     author = {William Revelle},
##     organization = { Northwestern University},
##     address = { Evanston, Illinois},
##     year = {2017},
##     note = {R package version 1.7.5},
##     url = {https://CRAN.R-project.org/package=psych},
##   }
```

The Data

The dataset that we'll use for this demonstration is called `bfi` and comes from the `psych` package. It is made up of 25 self-report personality items from the International Personality Item Pool, gender, education level and age for 2800 subjects and used in the Synthetic Aperture Personality Assessment.

The personality items are split into 5 categories: Agreeableness (A), Conscientiousness (C), Extraversion (E), Neuroticism (N), Openness (O). Each item was answered on a six point scale: 1 Very Inaccurate, 2 Moderately Inaccurate, 3 Slightly Inaccurate, 4 Slightly Accurate, 5 Moderately Accurate, 6 Very Accurate.

```
data("bfi")
```

Because the data file is embedded within R, you can query this information directly within the console, with `?bfi`. You can also look at the variable names and the first six lines of data, using `head(bfi)`.

```
head(bfi)
```

```
##        A1 A2 A3 A4 A5 C1 C2 C3 C4 C5 E1 E2 E3 E4 E5 N1 N2 N3 N4 N5 O1 O2 O3 O4 O5 gender
## 61617  2  4  3  4  4  2  3  3  4  4  3  3  3  4  4  3  4  2  2  3  3  6  3  4  3      1
## 61618  2  4  5  2  5  5  4  4  3  4  1  1  6  4  3  3  3  3  5  5  4  2  4  3  3      2
## 61620  5  4  5  4  4  4  5  4  2  5  2  4  4  4  5  4  5  4  2  3  4  2  5  5  2      2
## 61621  4  4  6  5  5  4  4  3  5  5  5  3  4  4  4  2  5  2  4  1  3  3  4  3  5      2
## 61622  2  3  3  4  5  4  4  5  3  2  2  2  5  4  5  2  3  4  4  3  3  3  4  3  3      1
## 61623  6  6  5  6  5  6  6  6  1  3  2  1  6  5  6  3  5  2  2  3  4  3  5  6  1      2
##        education age
## 61617         NA  16
## 61618         NA  18
## 61620         NA  17
## 61621         NA  17
## 61622         NA  17
## 61623          3  21
```

Finally, let's create a data object that just has the personality items in it, to facilitate some of our later analyses. We'll call it `bfi.items`.

```
bfi.items <- bfi[,1:25]
```

Item Difficulty

The simplest item analyses are often the best, and that's the case with item difficulty. It is simply the proportion of individuals that got the "correct" answer on a question, and so it is thought of as a method of determining how "easy" an item is (i.e., "is the question answered correctly by a high proportion of individuals?") or how "difficult" an item is (i.e., "is the question answered correctly by a low proportion of individuals?"). The ability of an item to discriminate among individuals within the sample is related to this, because items that are too difficult, or too easy, are not particularly good at distinguishing amongst individuals within your sample. Thus, item difficulty is an excellent property to compute for your set of items. And the good news is that it's quite easy to derive, in R. To start with, we will need to convert our continuous Likert style items into dichotomous items (i.e., "1" or "0"). In the BFI data, we would recode *very inaccurate*, *moderately inaccurate*, and *slightly inaccurate* to become **inaccurate**, while *very accurate*, *moderately accurate*, and *slightly inaccurate* become **accurate**).

Creating a Dichotomization Function

Dichotomizing variables may be an operation that you do fairly frequently, and so it may be worthwhile to create a function for future use. In R Studio, if you wanted to save this function you could simply create a *R Script* document from the drop-down file menu, under *New File*.

The function we're using requires that you input the data object name, and the number of response categories in each variable. For the BFI, there are 6 categories, and so we would use the function call `dich(bfi,6)`.

```
dich <- function(variable,categories) {
  midpoint <- categories / 2
  even <- as.logical(ifelse((midpoint-trunc(midpoint))==0,
                            "TRUE","FALSE"))
  midpoint <- ifelse(even==TRUE,midpoint,midpoint+.5)
  if(even==FALSE) variable[variable==midpoint]<- NA
  variable <- ifelse(variable<midpoint,0,1)}
```

If you feed this function a variable, it will dichotomize just that variable. If you feed it a data frame with several variables, it will dichotomize all of the variables within that data frame. We just need the 25 personality items, and so we will use the `bfi.items` object, in place of the full `bfi` object.

```
bfi.dich <- data.frame(dich(bfi.items,6))
```

Calculating and Using Item Difficulty Scores

Because each of our variables are (now) scored "0" or "1", an average of each variable will yield a proportion of the number of individuals that were scored "1" on each variable. This value is the item difficulty, and we want to calculate this value for each of the 25 items.

```
difficulty.bfi <- colMeans(bfi.dich, na.rm=TRUE)
```

We want items with a difficulty that falls between 0.2 and 0.8. The simplest way to do this is to use the `subset` function, in a way that will give us the difficulties that fall outside this range.

```
subset(difficulty.bfi, (difficulty.bfi < .2 | difficulty.bfi > .8))
```

```
##        A2        A3        A4        A5        C1        C2        C3        E3        E4
## 0.9376127 0.9055516 0.8763035 0.9119971 0.9157970 0.8829251 0.8805755 0.8407207 0.8559656
##        E5        O1        O3        O4
## 0.8862900 0.9550036 0.9202742 0.9353912
```

Scoring The Data

We'll set up the scoring key by scale. There are five scales in our data:

- *Agreeableness*
- *Conscientiousness*
- *Extraversion*
- *Neuroticism*
- *Openness*

The first step is to set up a list object that indicates the items that go with each of our scales. We can indicate negatively-keyed items with a negative sign.

```
bfi.keys.list <- list(agree=c(-1, 2, 3, 4, 5),
                      consc=c(6, 7, 8, -9, -10),
                      extra=c(-11, -12, 13, 14, 15),
                      neuro=c(16, 17, 18, 19, 20),
                      open=c(21, -22, 23, 24, -25))
```

Notice that we used the index within the `bfi` object, for each of the items. We could have used the variable names ("A1", "A2", etc.) within this list object, and it would have been perfectly acceptable to the key generating function - but it would have been problematic for our use of the alpha command later on.

We will now take this list object, and use the `make.keys` function to create the scoring key itself.

```
bfi.keys <- make.keys(bfi.items,bfi.keys.list,item.labels=colnames(bfi))
```

Finally, we can use this scoring key to calculate the scale scores for each of our 5 scales.

This is also where decisions about missing data are made - I have chosen to just average the non-missing values. If we had wanted to impute missing values with mean substitution (a problematic, albeit common choice), we would use `impute = "mean"` in place of `impute = "none"`. Finally, another common (and also problematic) practice is to use listwise deletion on the data, by only analyzing complete cases. To do this, you would remove `impute = "none"` and replace it with `missing = FALSE`.

Because we have both positively and negatively keyed items within our scales, we need to tell the function the minimum and maximum on the scale. This allows it to reverse key the negatively keyed items before creating the scale scores (an item can be reversed by subtracting the observed value from the maximum scale score - in this case, "6").

```
bfi.scored <-scoreItems(bfi.keys, bfi.items, impute = "none",
                                min=1, max=6, digits=3)
```

The actual scale scores for each of our five personality variables are now available within the `scores` value of the `bfi.scored` object.

```
head(bfi.scored$scores)
```

```
##      agree consc extra neuro open
## [1,]   4.0   2.8   3.8   2.8  3.0
## [2,]   4.2   4.0   5.0   3.8  4.0
## [3,]   3.8   4.0   4.2   3.6  4.8
## [4,]   4.6   3.0   3.6   2.8  3.2
## [5,]   4.0   4.4   4.8   3.2  3.6
## [6,]   4.6   5.6   5.6   3.0  5.0
```

Reliability Analyses

We can now use the `alpha` function to calculate the item statistics. Let's start with the **agreeableness** scale.

```
output.alpha.agree <- alpha(bfi.items[,abs(bfi.keys.list$agree)],
                            check.keys=TRUE)
output.alpha.agree
```

```
##
## Reliability analysis
## Call: alpha(x = bfi.items[, abs(bfi.keys.list$agree)], check.keys = TRUE)
##
##   raw_alpha std.alpha G6(smc) average_r S/N   ase mean  sd
##         0.7      0.71    0.68      0.33 2.5 0.009  4.7 0.9
##
##  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
## A4       0.69      0.69    0.65      0.36 2.3   0.0098
## A5       0.64      0.66    0.61      0.32 1.9   0.0111
##
##  Item statistics
##        n raw.r std.r r.cor r.drop mean  sd
## A1- 2784  0.58  0.57  0.38   0.31  4.6 1.4
## A2  2773  0.73  0.75  0.67   0.56  4.8 1.2
## A3  2774  0.76  0.77  0.71   0.59  4.6 1.3
## A4  2781  0.65  0.63  0.47   0.39  4.7 1.5
## A5  2784  0.69  0.70  0.60   0.49  4.6 1.3
##
## Non missing response frequency for each item
##       1    2    3    4    5    6 miss
## A1 0.33 0.29 0.14 0.12 0.08 0.03 0.01
## A2 0.02 0.05 0.05 0.20 0.37 0.31 0.01
## A3 0.03 0.06 0.07 0.20 0.36 0.27 0.01
## A4 0.05 0.08 0.07 0.16 0.24 0.41 0.01
## A5 0.02 0.07 0.09 0.22 0.35 0.25 0.01
```

We used the list object that we created to produce the scoring key, as this contained the indices associated with each of the items in the `bfi.items` object. You'll notice, however, that we applied an absolute value function to the object prior to use - this is because the alpha function can't cope with negative indices. Some of our items were, however, negatively keyed - and this needs to be taken into account within our calculations of alpha. Otherwise, these items will **negatively** contribute to overall alpha. Fortunately, the `alpha` function has the facility to automatically reverse key any items that have a negative item-total correlation. If you want to take advantage of this facility, you need to include the parameter `check.keys = TRUE`.

Let's go ahead and generate item analysis objects for the other four scales.

```
output.alpha.consc <- alpha(bfi[,abs(bfi.keys.list$consc)],
                            check.keys=TRUE)
output.alpha.extra <- alpha(bfi[,abs(bfi.keys.list$extra)],
                            check.keys=TRUE)
output.alpha.neuro <- alpha(bfi[,abs(bfi.keys.list$neuro)],
                            check.keys=TRUE)
output.alpha.open <- alpha(bfi[,abs(bfi.keys.list$open)],
                            check.keys=TRUE)
```

Manipulating the reliability objects

We saved our alpha calculations as objects, and so we can either have them generate a full set of reliability information (by typing in the name of the object, as we did with the agreeableness reliability calculations), or we can extract just the information that we want for a particular purpose.

Probably the most common bit of information that you're going to want to extract from your data is Cronbach's alpha. Let's just grab the alpha for each of our scales, and format it into a neat table that is labeled with the item names.

```
scale.names <- c("Agreeableness", "Conscientiousness", "Extraversion",
                 "Neuroticism", "Openness to Experience")
bfi.alphas <- as.numeric(c(output.alpha.agree$total[2],
                           output.alpha.consc$total[2],
                           output.alpha.extra$total[2],
                           output.alpha.neuro$total[2],
                           output.alpha.open$total[2]))
bfi.alpha.table <- data.frame(Scale = scale.names, Std.Alpha = bfi.alphas)
bfi.alpha.table
```

```
##                    Scale Std.Alpha
## 1          Agreeableness 0.7130286
## 2      Conscientiousness 0.7300726
## 3           Extraversion 0.7617951
## 4            Neuroticism 0.8146747
## 5 Openness to Experience 0.6072684
```

We can also generate a confidence interval for each of these alphas, using the `cronbach.alpha.CI` function in the `cocron` package.

Let's look at the confidence interval for the agreeableness scale. The alpha for this scale is 0.713, we have 2800 observations in total, and there are 5 items on the scale. We can calculate a 95% confidence interval for this scale, as follows:

```
library(cocron)
cronbach.alpha.CI(alpha = 0.713, n = 2800, items = 5, conf.level = .95)
```

```
## lower.bound upper.bound
##   0.6958426   0.7294721
```

This tells us that we can be 95% confident that the Cronbach's alpha for this scale ranges from 0.70 to 0.73. As you can imagine, this is useful information - particularly for those occasions when a reviewer has casted aspersions on your sample size!

Another specific piece of information that we might be interested in is the *alpha-if-deleted* for each of the items. This gives us information as to how the alpha will change (up or down) when a particular item is deleted. We can access this information from within the `alpha.drop` value in our reliability objects.

```
output.alpha.agree$alpha.drop
```

```
##     raw_alpha std.alpha   G6(smc) average_r      S/N    alpha se
## A1- 0.7185174 0.7255091 0.6730278 0.3978723 2.643109 0.008725479
## A2  0.6171800 0.6255799 0.5794588 0.2946317 1.670797 0.011903883
## A3  0.6002596 0.6129447 0.5578155 0.2836176 1.583610 0.012439996
## A4  0.6858057 0.6935413 0.6498474 0.3613369 2.263083 0.009825119
## A5  0.6429530 0.6555302 0.6050623 0.3223798 1.903012 0.011147900
```

```
output.alpha.consc$alpha.drop
```

```
##     raw_alpha std.alpha   G6(smc) average_r      S/N    alpha se
## C1  0.6940004 0.6964219 0.6400689 0.3644787 2.294045 0.009337288
## C2  0.6735715 0.6748686 0.6189270 0.3416374 2.075679 0.009890727
## C3  0.6887341 0.6939587 0.6443433 0.3617903 2.267533 0.009564203
## C4- 0.6538256 0.6629030 0.6028021 0.3295908 1.966505 0.010664033
## C5- 0.6897249 0.6902020 0.6283368 0.3577300 2.227910 0.009561821
```

```
output.alpha.extra$alpha.drop
```

```
##     raw_alpha std.alpha   G6(smc) average_r      S/N    alpha se
## E1- 0.7256547 0.7254473 0.6731108 0.3977979 2.642289 0.008369560
## E2- 0.6901804 0.6930860 0.6341860 0.3608429 2.258242 0.009508847
## E3  0.7279142 0.7262478 0.6737381 0.3987619 2.652939 0.008240909
## E4  0.7018885 0.7032346 0.6464289 0.3720235 2.369665 0.009073215
## E5  0.7436327 0.7442029 0.6913944 0.4210742 2.909348 0.007823617
```

```
output.alpha.neuro$alpha.drop
```

```
##    raw_alpha std.alpha   G6(smc) average_r      S/N    alpha se
## N1 0.7581379 0.7583430 0.7109569 0.4396265 3.138096 0.007473708
## N2 0.7632327 0.7633957 0.7158526 0.4464791 3.226466 0.007321954
## N3 0.7553428 0.7567103 0.7311738 0.4374379 3.110326 0.007662889
## N4 0.7953499 0.7968948 0.7688488 0.4951762 3.923557 0.006404928
## N5 0.8126022 0.8128355 0.7870014 0.5205500 4.342892 0.005853676
```

```
output.alpha.open$alpha.drop
```

```
##     raw_alpha std.alpha   G6(smc) average_r      S/N   alpha se
## O1  0.5315935 0.5340608 0.4761929 0.2227279 1.146203 0.01427836
## O2- 0.5672275 0.5701068 0.5103454 0.2489897 1.326159 0.01334500
## O3  0.4973614 0.5005554 0.4417967 0.2003557 1.002224 0.01526630
## O4  0.6114811 0.6208252 0.5602676 0.2904412 1.637306 0.01190493
## O5- 0.5116576 0.5279603 0.4738055 0.2185158 1.118466 0.01503747
```

Or we can ask for a reasonably comprehensive set of item statistics, by accessing the `item.stats` value in our reliability objects. This value will generate a data frame that includes the following:

value	description
n	number of complete cases for the item
raw.r	correlation of each item with the total score, not corrected for item overlap
std.r	correlation of each item with the total score, not corrected for item overlap, based on standardized items
r.cor	correlation of each item with the total score, corrected for item overlap and scale reliability
r.drop	correlation of each item with the total score, NOT including this item
mean	mean of the item
sd	standard deviation of the item

Recall when we discussed item discrimination, in the context of item difficulty earlier? Those item-total correlations (`raw.r`, `std.r`, `r.cor`, `r.drop`) are a direct estimate of item discrimination. Items that are particularly good at discriminating between individuals at the extreme ends of the scale will have strong positive correlations with the total score, and so this correlation is often cited as a measure of the "discriminatory power" of an item.

```
output.alpha.agree$item.stats
```

```
##        n     raw.r     std.r     r.cor    r.drop     mean       sd
## A1- 2784 0.5806903 0.5664248 0.3763568 0.3084177 4.586566 1.407737
## A2  2773 0.7279845 0.7479919 0.6665475 0.5636152 4.802380 1.172020
## A3  2774 0.7603190 0.7673623 0.7092193 0.5870046 4.603821 1.301834
## A4  2781 0.6541864 0.6306788 0.4712401 0.3944441 4.699748 1.479633
## A5  2784 0.6865886 0.6991918 0.5956637 0.4885651 4.560345 1.258512
```

```
output.alpha.consc$item.stats
```

```
##        n     raw.r     std.r     r.cor    r.drop     mean       sd
## C1  2779 0.6457479 0.6701768 0.5398979 0.4502417 4.502339 1.241347
## C2  2776 0.6964417 0.7097047 0.6027268 0.5045647 4.369957 1.318347
## C3  2780 0.6638749 0.6748292 0.5389117 0.4642048 4.303957 1.288552
## C4- 2774 0.7365348 0.7305518 0.6413275 0.5525467 4.446647 1.375118
## C5- 2784 0.7196827 0.6818557 0.5659503 0.4774692 3.703305 1.628542
```

```
output.alpha.extra$item.stats
```

```
##        n     raw.r     std.r     r.cor    r.drop     mean       sd
## E1- 2777 0.7238448 0.7026890 0.5882184 0.5162729 4.025567 1.631505
## E2- 2784 0.7797002 0.7646597 0.6935939 0.6053688 3.858118 1.605210
## E3  2775 0.6829944 0.7010725 0.5826706 0.5045725 4.000721 1.352719
## E4  2791 0.7466939 0.7459107 0.6625091 0.5779940 4.422429 1.457517
## E5  2779 0.6432360 0.6636566 0.5229059 0.4542446 4.416337 1.334768
```

```
output.alpha.neuro$item.stats
```

```
##       n     raw.r     std.r     r.cor    r.drop     mean       sd
## N1 2778 0.8000205 0.8025129 0.7647699 0.6672057 2.929086 1.570917
## N2 2779 0.7872842 0.7916618 0.7496171 0.6526162 3.507737 1.525944
## N3 2789 0.8080845 0.8059785 0.7425154 0.6748167 3.216565 1.602902
## N4 2764 0.7151742 0.7145498 0.5984984 0.5428001 3.185601 1.569685
## N5 2771 0.6806090 0.6743705 0.5317978 0.4864824 2.969686 1.618647
```

```
output.alpha.open$item.stats
```

```
##        n     raw.r     std.r     r.cor    r.drop     mean       sd
## O1  2778 0.6151215 0.6496038 0.5156383 0.3906794 4.816055 1.129530
## O2- 2800 0.6540084 0.5990735 0.4298223 0.3321195 4.286786 1.565152
## O3  2772 0.6746804 0.6926499 0.5910625 0.4505342 4.438312 1.220901
## O4  2786 0.4978647 0.5193170 0.2902618 0.2179427 4.892319 1.221250
## O5- 2780 0.6703869 0.6577082 0.5236747 0.4162105 4.510432 1.327959
```

Final Thoughts

As you can see, R gives you tremendous flexibility in the way that you present results and outputs. You can select only the output that you want, manipulate the way in which you present the findings, and even use your results in subsequent analyses.