Advanced Lab 5: Odds and ends

This section contains various topics that are of importance but do not necessarily fit the previous narrative.

1. The psych package contains useful functions especially for psychometric studies. But many of these functions have broader uses. We will provide a brief overview of some functions here, and provide links to the main documentations for the package.
2. ANOVA output can contain different types of results, typically called "type I, II, III" sums of squares. We discuss the differences here.

The psych package

First off, you can access important and helpful documentation about the package in the form of *vignettes*. You can see all available vignettes, or the vignettes specific to a package, as well as open up a specific vignette, using the vignette function provided by R:

```
vignette()
vignette(package="psych")              # Three available vignettes
vignette("intro", package="psych")     # Open the "intro" vignette
```

There is also a lot of information in the accompanying website:

- http://personality-project.org/r/book/
- http://personality-project.org/r/psych/
- http://personality-project.org/r/r.guide.html

Some of the code uses base R and is different in style than the approach we have taken thus far, but there is plenty of useful information there. Here are some functions to check out:

Computes the so-called *Mahalanobis distance* of points, to help identify outliers. You must restrict your data to only scalar variables.

Can be used to replace parts of the dataset with specific values (e.g. NA). This is an alternative to using the dplyr techniques we have discussed.

Can produce some basic graphs with error bar information. error.bars is also available.

Produces a matrix of graphs with information for the various variable interactions. Try pairs.panels(attitude, stars=TRUE) for an example.

Computes pairwise correlations and P-values for the associated hypothesis tests. The adjust parameter allows for a number of possible multiple-test adjustments. Try corr.test(attitude) %>% print(short=FALSE) for an example.

This function can be the basis for a function that produces a table with significant correlations flagged. We will show below how to write such a function.

Visually shows the information from corr.test. Try something like:

```
corPlot(Thurstone, stars=TRUE, numbers=TRUE, upper=FALSE, diag=FALSE)
```

outlier Shows a spider plot. See also radar. Example: spider(y=1,x=2:4,data=iris[,1:4])

mediate Provides some tools for mediation analysis.

The package also includes a number of datasets that might be useful in demonstrating psychometric topics. See data(package="psych") or the intro vignette for details.

The corrgram[1] package might also be of interest for visualizing correlation matrices.

Explicitly flagging significant correlations

We will now see how we can create a function that flags significant correlations in a correlation table. At the same time we will delve into some of the inner functioning of R.

We start with the corr.test function in the psych package, and store the result of its call:

```
cors <- corr.test(iris[,1:4])
```

We can see information about what this cors really is by using the str function (or by expanding the cors value in the Environment pane):

```
str(cors)
```

This output is a little hard to digest. First of all cors is a "list" of 9 items. They each appear following a dollar sign. And in fact we can access them based on the name next to that sign. For example:

```
cors$se
cors$adjust
```

We also see at the very bottom the line: – attr(*, "class")= chr [1:2] "psych" "corr.test". This tells us that the cors object has a "class" attribute with the two values "psych" and "corr.test". Many objects in R have similar outputs; for example the output of a linear model call typically has the class "lm". The class of an object changes how that object behaves for certain functions, like printing.

Let us take note of a couple more entries in this list:

- r contains the correlations. It has the "type" num [1:4, 1:4] which means it contains numeric values arranged in a 4 by 4 table. It also has a "dimnames" attribute, which contains the names of the two dimensions of the table.
- t contains corresponding t-values. It has the same structure as r.
- p contains the corresponding P-values, and it too has the same structure.
- ci contains confidence intervals, and it could be used to produce a graphical representation of the information.

As a quick example, let us use the ci information to draw quick plot of the different correlation confidence intervals.

```
cis <- cors$ci %>% rownames_to_column(var="pairs")
ggplot(cis) +
    aes(y=pairs, x=r, xmin=lower, xmax=upper) +
    geom_point() +
    geom_errorbarh()
```

[1] http://www.datavis.ca/papers/corrgram.pdf

Now let's see if we can flag the significant correlations:

```
p <- cors$p
stars <- case_when(
    p < .0001 ~ "****",
    p < .001  ~ "***",
    p < .01   ~ "**",
    p < .05   ~ "*",
    TRUE      ~ ""
)
dim(stars) <- dim(p)
dimnames(stars) <- dimnames(p)
stars
```

The dim (...) line set the dimensions of stars to match that of p, and the dimnames line added the names for the two dimensions.

This just shows us the different stars. We now need to merge them together with the correlation values, which are stored in cors$r. First we should cut off the amount of precision on the numbers, for printing purposes:

```
formattedRs <- format(cors$r, digits=2)    # Format the correlations
starredCors <- paste(formattedRs, stars)   # Merge with the stars
dim(starredCors) <- dim(p)
dimnames(starredCors) <- dimnames(p)
attr(starredCors, "class") <- "table"      # Define class a stable for nice printing
starredCors
```

We could manually repeat the above each time, but it would get tedious. A better idea is to create a function that does all these steps for us. When you define a function, you specify a list of *parameters* for it, and some can have default values. When we call a function we need to specify values for these parameters, as we have done many times in these labs. In our case our function will take a number of parameters:

- cors will be the first parameter, and it is expected to be the result of calling corr.test.
- digits will be the second parameter, specifying how many digits we should print. It will default to 2.
- max.stars will be the third parameter, specifying the maximum number of stars to show. For example, some people might not want to have the "4 star" case there. This will default to 4, but it could be set to 3 if someone didn't want the 4-star case to be shown separately.

Our function returns whatever the last thing is in its body. So we will conclude the function with the end result we want.

```
corrFlag <- function(cors, digits=2, max.stars=4) {
    p <- cors$p
    stars <- case_when(
        p < .0001 & max.stars >= 4 ~ "****",
        p < .001  & max.stars >= 3 ~ "***",
        p < .01   & max.stars >= 2 ~ "**",
        p < .05   & max.stars >= 1 ~ "*",
        TRUE      ~ ""
    )
    formattedRs <- format(cors$r, digits=digits)
    starredCors <- paste(formattedRs, stars)
    dim(starredCors) <- dim(p)
```

```
    dimnames(starredCors) <- dimnames(p)
    attr(starredCors, "class") <- "table"
    starredCors
}
# Example call:
iris[,1:4] %>% corr.test() %>% corrFlag()
```

An important thing to know about a function is that any variables you define within the function will not be added to your environment. So for example the formattedRs assignment we make within the function is only meaningful while that function runs, and is forgotten afterwards. This helps keep our environment clean.

And here is a different approach: We would like to start with the correlation table, then create a data frame of the pairs of variables and their corresponding correlations. Then we can add a column that has the star information. Let's take a look. First we use as.data.frame to turn the cors$r array into a data frame, then we use rownames_to_column to bring in the row names as a new column, then we gather the remaining columns into one:

```
corrsDf <- cors$r %>% as.data.frame() %>%
            rownames_to_column(var="var1") %>%
            gather(key="var2", value="r", -1)
corrsDf
```

Now we do the same thing for the t and p values:

```
corrsTs <- cors$t %>% as.data.frame() %>%
            rownames_to_column(var="var1") %>%
            gather(key="var2", value="t", -1)
corrsPs <- cors$p %>% as.data.frame() %>%
            rownames_to_column(var="var1") %>%
            gather(key="var2", value="p", -1)
```

Now we use inner_join to bring these three frames together:

```
df <- corrsDf %>% inner_join(corrsTs) %>% inner_join(corrsPs)
```

Now we want to restrict df to only contain each pair twice, and to not consider a variable and itself. We can simply compare the variable names for this, then arrange by the p-values:

```
df %>% filter(var1 < var2) %>%
    arrange(p)
```

We can now add a star column to this data frame. Instead of using case_when as above, we will use cut to end up with an ordered factor:

```
finalDf <- df %>%
    filter(var1 < var2) %>%
    arrange(p) %>%
    mutate(star=cut(p, breaks=c(0, .0001, .001, 0.01, 0.05, 1),
                       labels=c("****", "*** ", "**  ", "*   ", "    "),
                       include.lowest=TRUE))
finalDf
```

ANOVA output types

When considering the output of an ANOVA analysis, there are different kinds of "sum of squares" computations and tests that we can be performing. They all come down to which models we are trying to compare. As a running example, let us consider a small dataset that is used in the SAS documentation:

```
data<-tribble(
    ~Family, ~Gender, ~Height,
    1,"F",67,
    1,"F",66,
    1,"F",64,
    1,"M",71,
    1,"M",72,
    2,"F",63,
    2,"F",63,
    2,"F",67,
    2,"M",69,
    2,"M",68,
    2,"M",70,
    3,"F",63,
    3,"M",64,
    4,"F",67,
    4,"F",66,
    4,"M",67,
    4,"M",67,
    4,"M",69
)
data <- data %>% mutate(Family=factor(Family), Gender=factor(Gender))
data %>% count(Family, Gender)
```

We can see that this is an *unbalanced* design, with unequal cell sizes. A consequence of this lack of balance is that contrast-coded variables will be correlated. This results in a number of different SS computations, and disagreements about which one to use. A good reference for this discussion are the following two resources:

- Working with unbalanced cell sizes in multiple regression with categorical predictors[2] by Ista Zahn
- Anova – Type I/II/III SS explained[3]
- You can read here[4] a critique of some of the approaches described here.

The differences arise from the kind of hypothesis we want to test. We can roughly consider the following graphical depiction of the various models we have in our example:

ANOVA SS Types

The classification into three different types comes from SPSS and SAS and has become common practice. Briefly:

Type I In this type, factors are compared sequentially as their are being added to the model. so if the model is Height~Family+Gender+Family:Gender, then we consider the effect of Family compared to the null model, then we consider the effect of Gender in the presence of Family, and finally with consider the effect of the interaction term in the presence of both the other variables.

[2] http://psychology.okstate.edu/faculty/jgrice/psyc5314/SS_types.pdf
[3] https://mcfromnz.wordpress.com/2011/03/02/anova-type-iiiiii-ss-explained/
[4] http://www.stats.ox.ac.uk/pub/MASS3/Exegeses.pdf

Type II In this type, factors are considered against a model that contains all the other factors, and assuming no interactions.

Type III In this type, we consider factors against a model that contains all the other factors as well as the interactions.