Lab: Data Cleanup

Introduction

We can loosely split the "Data cleanup" part of a project in three stages:

- Loading the data from a spreadsheet file or some other kind of storage. The readr and readxl packages help us with this stage.
- "Tidying" the loaded data to ensure the data is structured in a way that facilitates analysis. This mainly means that each variable is in one column. For example instead of having measurements over 5 days split across 5 columns, there should instead be one column of 5 times as many observations, with a second column identifying the day index for each. The tidyr package provides methods that help with this stage.
- Manipulating the data in various ways to obtain summaries or prepare for graphing. The dplyr package offers a rich set of methods to help with that.

All the packages mentioned above are part of the "tidyverse[1]" package collection, and are automatically included in our hanoverbase package.

Importing

We start by importing the data from an external source. For this example, we will use the datasets/compression.xlsx spreadsheet. It might help to view the spreadsheet in Excel.

In terms of providing us access to the data, the sheet is somewhat messy: It has the data spread over multiple tabs/sheets, and sheets often contain non-data components like summaries and graphs.

We will start with reading the data from the first sheet, titled "Upper Arm Circ". You will need to first have the dataset imported into your project directory (you did create a new project with its own project directory, right?). You can then click on the dataset to open up the import dialog. We will not fully use the import dialog, but it is a good starting point to look at the data.

In the import dialog, you can select the sheet you want, here the "Upper Arm Circ", from the "Sheet" drop down in the "Import Option" section. Then you would want to copy the couple of lines on the right, which do the import:

```
library(readxl)
compression <- read_excel("~/statsLabPractice/compression/compression.xlsx",
    sheet = "Upper Arm Circ")
```

Then we would paste these lines in an r-chunk after canceling the import dialog. Remember that the first line loads a package for us, and only needs to be placed once.

Then we need to change the second line, to make sure we only try to load the first 8 data points, and also improve the stored data set name:

```
compression.uac <- read_excel("~/statsLabPractice/compression/compression.xlsx",
    sheet = "Upper Arm Circ",
    range = "A1:O9")
```

[1] https://www.tidyverse.org/

We learned about this extra parameter by reading the documentation for read_excel.

Let us also load another one of the sheets:

```
compression.lac <- read_excel("~/statsLabPractice/compression/compression.xlsx",
    sheet = "Lower Arm Circ",
    range = "A1:O9")
```

This shows the basics of importing data. You can explore the readr and readxl packages to find out more. Also check out the haven package for loading SPSS, Stata and SAS data. There are also ways to access a database directly; the package dbplyr[2] might be a good place to start for that.

Tidying

We will take various steps to "tidy" the data. Before we do that, let's discuss the data a bit. The first column represents of course the subject ids, and each row of data corresponds to a subject. The "CS..." variables correspond to the subject using a "Compression Sleeve" while the "NT..." variables correspond to the subject receiving "Pneumatic Compression" therapy. the "...UAC..." part represents that these are "Upper Arm Circulation" measurements, and finally the remaining bits "PR, PO, 1, 2, 3, ..." represent the time variable (pre-exercise, post-exercise, one day later etc).

The data is what is known as a "wide" form: In a "tidy" version of the data the kind of treatment, the quantity being measured, and the time of observation would all be three separate factor variables and we would have a single measurement column. In effect all the measurements currently present in the sheet will be placed in one column, and various information that is currently coded in the variable names will instead be stored in different variables/columns specifically created for that purpose.

We will now convert the data sheet in this form. The first step in doing so is to isolate the CSU columns as well as the subject column. We will use the subset method for that, which picks out a subset of the variables. There are various ways to specify which variables to select, and you can look at the documentation or cheatsheets for that.

```
compression.uac %>%
    select(X__1, CSUACPR:CSUAC5)
```

We could have stored this result in a dataset, but we will instead add another process step in the pipeline, which does the transformation. It is called gather, and it simply collects some columns and puts them together into one column (the –X__1 tells it to combine all the other columns except X__1):

```
sleeve.uac <- compression.uac %>%
    select(X__1, CSUACPR:CSUAC5) %>%
    gather(key="time", value="value", -X__1)
View(sleeve.uac)    # Do in console
```

Notice how the new time column contains the variable names, and we now have 7 times as many rows, one for each combination of a subject and a variable.

Next we need to split the variable time three parts:

- The fact that it is a "compression sleeve" treatment
- The fact that it is the upper-arm

[2] https://dbplyr.tidyverse.org/articles/dbplyr.html

- The timing (pre, post, day 1 etc)

The separate method helps us with this. Its syntax will look a bit weird. The second argument (into=) specifies the names to give to the three parts, while the third argument (sep=) specifies the cutoff points (after the first two letters, after the first 5 letters). This effectively splits each value like "CSUAPR" in three parts: the first two letters, the next three letters, and the rest.

```
sleeve.uac2 <- sleeve.uac %>%
        separate(time,
                 into=c("treatment", "measurement", "time"),
                 sep=c(2,5))
View(sleeve.uac2)
```

Finally, we will fix the first column: First we want to change its name to "subject", with the method "rename":

```
sleeve.uac2 %>%
    rename(subject=X__1)
```

And then change the strings "sub 1" into the numbers themselves, using mutate. For this to work, we also need to load a string-manipulation library called stringr :

```
sleeve.uac3 <- sleeve.uac2 %>%
    rename(subject=X__1) %>%
    mutate(subject = str_extract(subject, "\\d+"))
```

We will likely need to repeat all these steps for each part of the dataset, and they will be more or less the same steps, so let's see if we can make a function out of it:

```
convert.to.long <- function(df, middleLetters) {
    df %>%
        gather(key="time", value="value", -X__1) %>%
        separate(time,
                 into=c("treatment", "measurement", "time"),
                 sep=c(2, 2 + middleLetters)) %>%
        rename(subject=X__1) %>%
        mutate(subject = str_extract(subject, "\\d+"))
}
Notice that we used another variable, "middleLetters", to represent the number of letters use
```

Then we could simply have done the following:

```
sleeve.uac <- compression.uac %>%
    select(X__1, CSUACPR:CSUAC5) %>%
    convert.to.long(3)
```

We will now do the same for the other part, the NT columns:

```
pneumatic.uac <- compression.uac %>%
    select(X__1, NTUACPR:NTUAC5) %>%
    convert.to.long(3)
```

We will repeat this for the "lower-arm-circulation" variables:

```
sleeve.lac <- compression.lac %>%
    select(X__1, CSLACPR:CSLAC5) %>%
    convert.to.long(3)
pneumatic.lac <- compression.lac %>%
    select(X__1, NTLACPR:NTLAC5) %>%
    convert.to.long(3)
```

Now we have four data sets that all look similar in terms of the columns they contain etc. We would like to merge them together by simply placing them below each other and matching the columns. We can use the bind_rows method for this:

```
alldata <- bind_rows(sleeve.uac, pneumatic.uac, sleeve.lac, pneumatic.lac)
```

Now it is time to more properly code our factor variables, using mutate. In R, there are different "modes" for vectors/variables. One mode is "numeric" for scalar variables, another is "character" for strings. There is also "factor" which is what should be used for factor variables. A factor typically uses numbers for the underlying codes/levels, and it also contains a "value label" for each code/level. For example, if we take the treatment variable and turn it into a factor:

```
subject$treatment
subject$treatment %>% factor()
```

You will see the two results looking a bit differently. The important thing to know is that in order to do some of the standard statistical analysis steps, we need variables to be coded as "factors":

```
alldata <- alldata %>% mutate(
    subject = factor(subject),
    treatment = recode_factor(treatment, CS="Compression Sleeve", NT="Pneumatic Compression"),
    measurement = recode_factor(measurement, UAC="Upper Arm Circ", LAC="Lower Arm Circ"),
    time = recode_factor(time, PR="Pre", PO="Post", `1`="Day 1", `2`="Day 2",
                                `3`="Day 3", `4`="Day 4", `5`="Day 5")
    )
```

We are now ready to do some work with this data. Here's some examples of graph we can produce, you'll see more about this sort of graph in the graphing section.

```
ggplot(alldata) +
    aes(x=time, y=value, group=subject, color=subject) +
    geom_path() +
    facet_grid(measurement~treatment)

ggplot(alldata) +
    aes(x=time, y=value, group=treatment, col=treatment) +
    stat_summary(geom="pointrange", fun.data=mean_se, fatten=3, alpha=0.6,
        position=position_dodge(width=0.1)) +
    geom_line(stat="summary", fun.y=mean, position=position_dodge(width=0.1)) +
    facet_grid(measurement~.)
```

Let us move on to doing some numerical work with this data. For instance we may want to compute the means and standard deviations for each day, for each treatment and for each type of measurements. In order to perform such computations, we need to first "group" the data by the grouping variables we want to use. The group_by method helps us with that. Then we can use the summarize function to produce numerical summaries per group.

```
summaries <- alldata %>%
    group_by(treatment, measurement, time) %>%
    summarize(mean=mean(value),
              sd=sd(value),
              median=median(value),
              q1=quantile(value, 0.25))
```

We could then visualize those summaries in some way:

```
ggplot(summaries) +
    aes(x=time, y=mean, color=treatment, group=treatment) +
    geom_line() + facet_grid(measurement~.)
```

Let us try a more complicated transformation. Since different subjects start from different values at the pre-stage (you may have noticed that in the first graph you created earlier), we could try to normalize that initial amount, and compute the other values as relative to that that initial amount. What we need to do here is again group by the subject, treatment and measurement, but not time, and now within each category we need to mutate our values instead of summarizing:

```
relativedata <- alldata %>%
    group_by(subject, treatment, measurement) %>%
        mutate(relChange=value/first(value))
```

We'll also want to remove the "pre" entries, as they are no longer really applicable:

```
relativedata <- alldata %>%
    group_by(subject, treatment, measurement) %>%
        mutate(relChange=value/first(value)) %>%
        filter(time != "Pre")
```

Let's repeat our earlier graphs now. For the second graph we'll adjust the y-axis to show relative increase (more about this stuff no the graphics session):

```
ggplot(relativedata) +
    aes(x=time, y=relChange-1) +
    geom_path(aes(group=subject)) +
    facet_grid(measurement~treatment) +
    labs(y="relative change from pre") +
    scale_y_continuous(labels = scales::percent)

ggplot(relativedata) +
    aes(x=time , y=relChange-1, group=treatment, col=treatment) +
    stat_summary(geom="pointrange", fun.data=mean_se, fatten=3, alpha=0.6,
        position=position_dodge(width=0.1)) +
    geom_line(stat="summary", fun.y=mean, position=position_dodge(width=0.1)) +
    facet_grid(measurement~.) +
    labs(y="relative change from pre") +
    scale_y_continuous(labels = scales::percent)
```