Workshop part 2.0

Intro to Linear Modeling in R

Working with data

We always need to select parts of the dataset—maybe we only want data from the subjects who performed well on a behavioral task, or who are of a certain age or gender. To do this, you want to create a new dataframe that is a subset of the original dataframe.

Make sure R is open, and that you have loaded Duncan.text into a variable.

```
duncan=read.table(“Data/Duncan.txt”, header=TRUE)
```

You can select parts of the dataframe by accessing them directly or by name, or using subset.

```
duncan$type
```

This returns the whole “type” column.

```
duncan[,1]
```

This is all rows of the first column.

```
duncan[2,]
```

This is all columns of the second row.

But if we want the data only from particular rows, say the blue collar jobs (where type is set to “bc”):

```
subset(duncan, type == “bc”)
```

The == double equals indicates that it is a logical statement, and the results are the subset of the dataframe where the statement evaluates to TRUE.

Summary() works on the output of subset(), as do many other functions. You might want to save it as a new dataframe just to be able to use it in an analysis, for example.

```
bc_data = subset(duncan, type == “bc”)

bc.mod = lm(bc_data$prestige ~ bc_data$income)
```

Quick digression: the use of c() – the concatenate function c() shows up everywhere. By default, it creates a vector of all the objects passed to it. See what the outcome is of each of these commands:

```
c(1,2,3,4,5)

c("type", "income")

c(1,2,3,4)/2

c(1,2,3,4) + c(6,8,10,12)

log(c(10,100,1000), base=10)

x=c("bob", "jane")

y=c("grace", "eric")

c(x,y)
```

Using c(), you can use subset to select multiple columns from your dataset as for example:

```
Subset(duncan, select = c(“type”, “income”))
```

Or combine both approaches to choose subsets of both rows and columns:

```
Subset(duncan, prestige > 25, select = c(“type”, “income”))
```

Data Transformations

Adding columns to a dataframe: We often need to do transformations of a variable, and you might want to keep it with the original data. It is easy to add a new column to a dataframe—just give it a name and the right number of values for your dataset. For example, let’s add some random values in a new column. The Duncan dataframe has 45 entries, so we need 45 random values

```
Nonsense = runif(45)
```

To add a new column called SillyData to the Duncan dataframe with these values,

```
Duncan$SillyData = Nonsense
```

How can you check that it worked?

Turning a numerical variable into a factor: Suppose for some reason you have men and women subjects in your dataset coded as 0 and 1. By default, R will assume these are numbers. You have several options—you can recode the data to be labeled by strings (e.g., “men” and “women”), or you can tell R to treat the numbers as factors. Let’s load an example dataset into a dataframe called Stressdata:

```
Stressdata = read.csv(“data/dataset_anova_badcode_interactions.csv”, header = true)
```

What’s in it?

```
summary(Stressdata)
```

Let’s make Gender into a factor rather than a number and see what’s different:

```
Stressdata$Gender = as.factor(Stressdata$Gender)

summary(Stressdata)
```

Note that we overwrote the original column data.

Relabeling a number into a string: We’ve made Gender into a grouping variable or a factor by using as.factor(). But that’s not ideal: Preferably we don’t want to have to remember that 1 = male and 2 = female. This can be done by changing the labels on the “levels” of the factor.

Let’s see what levels there are for this factor:

```
levels(Stressdata$Gender)
```

Then reset the values and check it did what you wanted:

```
levels(Stressdata$Gender) = c(“male”, “female”)

summary(Stressdata)
```

Z-scoring a variable: We often need to de-mean or center a variable, by subtracting the mean from all the values; or we may need to nomalize or z-score it by subtracting the mean and dividing by the variance. The command to do either of these is scale().

Let’s try centering the StressReduction values and saving it in a new column in the dataframe:

```
Stressdata$StressReduction.cent = scale(Stressdata$StressReduction, center = TRUE, scale = FALSE)
```

Actually, let’s z-score it too—that’s a very similar command:

```
Stressdata$StressReduction.z = scale(Stressdata$StressReduction, center = TRUE, scale = TRUE)
```

What happened to Stressdata?

Make a histogram of the centered values, and another of the z-scored values to check.

Working with incomplete data: Sometimes data is incomplete, and you have to specify what you want to do with missing data. You can throw samples out that are incomplete, which may be necessary to get a dataset where no data are missing; or you can tell the functions you are using to drop cases as needed. Let’s see an example—this is the same as the previous dataset, but with a cognitive score and some missing data:

```
missdata = read.csv("Data/dataset_anova_missing.csv", header=TRUE)
```

```
summary(missdata)
```

Note the NA’s in the StressReduction variable.

Some functions fail in the presence of missing data, but some don’t:

```
median(missdata$StressReduction)

lm(missdata$StressReduction ~ missdata$Treatment)
```

Tell median what to do with the missing data:

```
median(missdata$StressReduction, na.rm = TRUE)
```

When a function returns NA, definitely check its help file to see whether it has a na.omit or na.rm or something similar in its options, that you can use.

Binning a quantitative variable: cut() is a standard function that will identify the number of intervals you specify. But then you have to create a new variable based on those intervals, and save it in a meaningful way.

```
tmp = cut(Stressdata$StressReduction, 3)
```

Recode with labels, Low Medium High--

```
tmp2 = cut(Stressdata$StressReduction, 3,

labels = c('Low', 'Medium', 'High'))
```

Add these new variables and labels to the data frame:

```
Stressdata$ReductionLevel = tmp

Stressdata$ReductionLevelGroup = tmp2
```

And remember to look at the data and see what’s changed!

Looking at your data: plot() is a very smart function that does its best to make sense of whatever you give it. It is not as powerful as scatterplot() or boxplot() but it’ll do simple boxplots, scatterplots, etc.

Writing your data out to disk: write.table() –give it the dataframe, and the file name. Use “” for the file name if you want to dump the output to the screen to make sure you are formatting it as you want to.

```
write.table(Stressdata, file = "NewData.csv", sep = ",", row.names = FALSE)
```

Your exercise!

1) Read in one of the files in the Data directory

2) Select a dependent variable and two independent variables and use subset() to make a smaller dataframe

3) z score the dependent variable and save it as a new variable in the data frame

4) run a linear model on the smaller dataset using the z-scored dependent variable--either multiple regression or anova using lm() – and examine the results.

4) write your new dataframe out to a new file.

5) make a histogram of one of the data variables