Lab 4 – Descriptive Statistics

Questions? montwe@ualberta.ca or isaacren@ualberta.ca

In this lab, we'll cover how you can calculate descriptive statistics that we discussed in class. We also learn how to summarize larger databases efficiently, and we will again cover how to export the results as a CSV file. Such a table of summary statistics can be the basis for graphical representation of quantitative data that we cover in the subsequent labs.

4.1. Simple, descriptive functions in R

Create a "Lab 4" folder. Then download the aspen dataset from the class website (aspen.csv). It has a unique ID as well as response variables DBH and VOL (diameter at breast height and volume).

Set your working directory either by doing it manually (save script in the same folder as the data, then Session > Set Working Directory > To Source File Location) or by coding it with `setwd()`. Then import the aspen.csv file into R and save the table as an object named "asp":

```
setwd("C:/YourPathHere/Lab4Folder")
asp = read.csv("aspen.csv")
head(asp)
```

Now calculate the mean of DBH using the "mean" function by specifying the table and column name in the brackets. Remember the following format from an earlier lab: table$columnname

```
mean(asp$DBH)
```

Try out the following R functions that will give you summary statistics. Insert the response variable name (DBH or VOL) that you are interested in between the brackets as you did in the line above.

```
mean()  median()  max()  min()  range()  which.max()  which.min()
var()  sd()  quantile()
```

Percentiles can be calculated with the quantile function by specifying the percentiles you want with a vector (you will recall that you can create a vector using the `c()` function):

```
quantile(asp$DBH, c(0.025,0.05,0.95,0.975))
```

There are also these handy functions to calculate multiple statistics:

```
fivenum(), summary()
```

4.2. Multi-variable data summaries in R

Using the same steps from above, now load the lentil-protein dataset from the website (you also created this in Lab 2).

```
dat1=read.csv("lentils_with_protein.csv")
head(dat1)
```

Now, we want to do some more complex data summaries. For example, we may want to know the mean and standard deviation of yield and protein content for each lentil variety at each farm. With the formulas above, this would require a lot of programming to subset the data and calculate the statistics of interest. Thankfully there is a nice R-package "`plyr`" that does that automatically for us, but you need to install it first:

```
install.packages("plyr")
```

As described in the last lab, the first thing you always have to do with additional packages is to load them into the computer memory. You must rerun this "library" command every time you re-open R. (Only the base-functionality is loaded by default, not the additional packages that you may install.)

```
library(plyr)
```

We will also need ggplot2, so let's install and load this now as well:

```
install.packages("ggplot2")
library(ggplot2)
```

Now we are ready to calculate some summaries. The function `ddply()` first needs to be told which dataset to work with (`dat1`), then given a list of independent variables to make the summaries by (in parenthesis and preceded by a period). Because you must specify the dataset right at the beginning in `ddply()`, you can skip the typical table$columnname format – you only need to specify the column name when using `ddply()`. Here we use the "summarize" function of `ddply()` (but it can do other stuff). You provide the function or formula that should be applied to the data. Try:

```
ddply(dat1,.(FARM), summarise, mYIELD=mean(YIELD))

ddply(dat1,.(FARM,VARIETY), summarise, mYIELD=mean(YIELD))

ddply(dat1,.(FARM), summarise, mPROTEIN=mean(PROTEIN))
```

The result from the last command may strike you as odd. However, this is how many functions work by default. If you have missing values, the result of the function will be a missing value, unless you add a comment that removes missing values before the calculation.

```
ddply(dat1,.(FARM), summarise, mPROTEIN=mean(PROTEIN, na.rm=T))
```

This is true and worth to keep in mind for all statistical functions. Try:

`mean(dat1$PROTEIN)`	versus	`mean(dat1$PROTEIN, na.rm=T)`
`sd(dat1$PROTEIN)`	versus	`sd(dat1$PROTEIN, na.rm=T)`
`range(dat1$PROTEIN)`	versus	`range(dat1$PROTEIN, na.rm=T)`

To top things off, let's calculate the mean and standard error of yield and protein content. We have not covered standard errors in class yet, but I want to demonstrate that you can use `ddply()` with any formula of interest. To get the standard error, we divide the standard deviation by the square root of the number of observations.

For the protein dataset, we can get the total number of observation with the function `length()`:

```
length(dat1$PROTEIN)
```

You should get a length of 24. (You can verify this manually by running `dat1$PROTEIN` and counting the number of observations in the output).

For protein content, we need the number of *non-missing observations*, which we get with:

```
length(dat1$PROTEIN [!is.na(dat1$PROTEIN)])
```

You should get a length of 6. If you're unsure what the `is.na()` function does, it might help to run a subset of the code:

```
is.na(dat1$PROTEIN)
```

The output is true/false, so the previous line of code specifies that you want values only where "is.na" is false (i.e., it is not a missing value).

Now let's combine those principles into code with `ddply()`, writing the results into a new table (`dat2`) and then export the result as a CSV.

```
dat2 = ddply(dat1,.(FARM,VARIETY), summarise,
mYIELD    = mean(YIELD),
seYIELD   = sd  (YIELD)/sqrt(length(YIELD)),
mPROTEIN  = mean(PROTEIN, na.rm=T),
sePROTEIN = sd  (PROTEIN, na.rm=T)/sqrt(length (PROTEIN[!is.na(PROTEIN)])))

head(dat2)
write.csv(dat2,"lentil_summary.csv",row.names=F)
```

This may be a little complicated if you are not used to programming, but you see that it is a powerful way to quickly summarize your data. It's the numerical equivalent to:

```
ggplot(dat1,aes(x=FARM, y=YIELD, fill=VARIETY))+
  geom_boxplot()
```

Just for the record, the R base package has a function that is similar to `ddply()`, but its output is ugly, it loses the names of group variables, and it can't do complex functions. The general syntax is:

`aggregate(`variable(s)`, by=list(`class variable(s)`), FUN=`statistic of your choice`)`

For example:

```
aggregate(dat1$YIELD, by=list(dat1$FARM,dat1$VARIETY), FUN=mean)
```

Now try a two-variable example:

```
aggregate(dat1[,c("YIELD", "PROTEIN")], by=list(dat1$FARM,dat1$VARIETY),
FUN=mean, na.rm=T)
```

There is no reason to use this instead of the `plyr` package, however. This is just another tool for your R toolbox.

4.4. Excel Pivot Tables

Excel is very useful in its "pivot table" functionality, which produces the types of summaries that we have just programmed with the `plyr` package in R. If you just want some quick summary statistics from a database, pivot tables are often easier to use and you have more flexibility in arranging your table of summary statistics. I highly recommend exploring this functionality with your own data.

Let’s try this out:

- Open the lentils_with_protein.csv in Excel
- Select your entire data range
- Choose the ribbon “Insert” > “Pivot Table” (hit “ok” to the dialogue box that comes up asking where you want to put your pivot table)
- To calculate means for “FARM” and “VARIETY” combinations:
    - From the “PivotTable Field List” in the top-right corner, drag “YIELD” into the “Values” field in the lower-left corner.
    - You will notice the sum has now appeared in your new worksheet. “Sum” is the default in pivot tables, but that is less meaningful for us.
    - To change from “Sum” to other functions like “Average”: double-click the “Sum of YIELD” in the “Values” field and choose “Value field settings”. From the drop-down list, then select “Average”.
    - Arrange the table yield means in various ways by dragging “FARM” and “VARIETY” (in the “PivotTable Field List) into the “Rows”/”Columns”/”Filter” fields.

“FARM” and “VARIETY” can be both in the “Rows” field, which gives you something similar to the `plyr` for a compact table. Very handy!

If you move independent variables into the “Filter” field, you get a drop-down list, where you can choose subsets (e.g. remove Farm1 data by unchecking the box). This is handy to quickly subset your data, for example to remove a certain treatment from the summary statistics.

You can further drag “YIELD” a second time into the “Value” field, and this time select “standard deviation from the “Value field settings” dialog. Similarly, you can drag “PROTEIN” there as a third item, etc.

The `plyr` package is of course still useful if the calculation of summary statistics is part of a larger program, if the function of interest is not available in Excel Pivot Tables (for example Standard Errors), or if you want to calculate statistics for many dependent variables simultaneously (for example you will quickly get tired of dragging variables to the value field if you have 365 daily measurements over the course of a year).