Lab 2: Data Tables & Basic Data Management

Questions? david.montwe@ualberta.caor isaacren@ualberta.ca

In this lab, we will learn how to create and manage data tables for analysis. We work with a very simple example, so it is easy to see what the code does. In your own projects, the same code can apply to manage and modify large datasets. You may need to merge multiple data sheets, subset data tables to focus your analysis on treatments or locations, fix errors in your datasets, transform or modify variables, or re-arrange data that has not been entered in a format fit for analysis.

It is good to do all these operations with scripts that leave a trail of changes and corrections that you did. If you change your mind later, none of your original data has been touched. You simply change a bit of code and execute your script again to do a slightly modified analysis (e.g. with or without outliers).

2.1. Creating a data table

Let's practice creating and importing a data table into R. Here is an experiment, where three lentil varieties (A, B, C) are tested at two farms. What you see below are the lentil yield in kg/ha that were observed in randomized experimental plots at each of two farms. This is the field-layout of the plots:

Farm 1

720 (A)	515 (B)	545 (B)
540 (C)	760 (A)	502 (C)
492 (B)	480 (B)	740 (A)
505 (C)	690 (A)	510 (C)

Farm 2

163 (A)	405 (B)	389 (B)
375 (C)	163 (A)	381 (C)
387 (B)	375 (B)	176 (A)
385 (C)	168 (A)	400 (C)

- What are the Populations, Samples, Units, Independent Variables, Dependent Variables, Data? (possibly not all of the terms apply)
- How does this information needs to be arranged into a data table, then?
- Enter your data table in the correct format into Excel.

2.2. Import data to R

To recap what you learned in the first lab, write code that imports your new data table into R.

- If you have difficulty setting yourself up to do this please let me or the TA know!
- If you have designed your data table in the right format, R will immediately know how to analyze it correctly. As a check, call for an Analysis of Variance in R. The sums of square for "Farm" should be 435243 returned from the command:

```
summary(aov(dat$YIELD ~ dat$FARM * dat$VARIETY))
```

- Note that your variable names and capitalization may be different

- Also try out the head() command as a quick check if your import worked correctly, the str() command to check your variable types, and if necessary do the appropriate variable type conversions with the as.factor() command.
- For example, if you entered "1" and "2" under the "FARM" heading, then R will think the FARM variable data type is an integer (which it is not – it is a category or factor). That can be changed through this code:

  ```
  dat$FARM <- as.factor(dat$FARM)
  ```

- Note that you can use the $ symbol to specify in which data table R is supposed to look for the variable FARM (the general format is table$variable).

2.3. Subset data tables: selecting rows and columns in R

In many cases, you don't want to analyze your entire dataset in one go, you can select particular rows and columns and get rid of the rest of the data if you want to focus your analysis in a particular aspect of interest. In R, you ask for rows, columns, and individual cells of a table with this extremely important and widely used command: [,]. We will use this a lot, so remember what these simple brackets specify:

data_table_name [row(s) , column(s)]

Let's start by calling individual cells of a data table, where I assume that the data table name is "dat". Change "dat" to whatever name you use:

- `dat[1,2]` should return the value of the first row and the second column of the "dat" table.
- `dat[5,3]` returns the value of the fifth row and the third column.
- You can also assign this value to a variable: `A=dat[5,3]`, then call A

We can also call entire rows or columns by leaving one of the fields between brackets blank:

- `dat[4,]` will return the fourth row,
- `dat[,3]` will return the third column,
- and `Y=dat[,3]`, will create a vector from the third column that you can recall later: Y

Third, we can get sections of data by specifying an array:

- `dat[1:3,1:3]` will return the first three rows of the first three columns
- `farm2=dat[13:24,]`, will return rows 13-24, which probably is the data for farm 2, if you entered the data in this order. We can now use this subset of farm2 for analysis.
- `dat[,2:3]` will return the second and third column.

- A more transparent, and therefore better option is to specify the variable names instead of the column numbers. For example `dat[,c("ID","YIELD")]` will return those two columns.

- Finally, you can use the minus sign to drop rows or columns. Try `dat[-20,]` or you can drop multiple rows that perhaps represent outliers: `cleandat=dat[-c(2,7,15),]`.

Now, what if your data is not arranged in a nice way to subset it by using row numbers or column numbers? For example, how would we remove a lentil variety from the analysis? You can use conditional statements to select rows:

The conditional operators are:

==	Equal to. Note: A double "=" sign is reserved for the "is it equal?" question, while a single "=" is reserved for the command "make it equal to"
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Lesser than or equal to

You may combine or stack several conditional statements with:

&	And
|	Or
!	Not

These can be combined using brackets () to determine the hierarchy of conditional statements:

- For example, `dat[dat$VARIETY!="C",]` will remove variety C from the dataset. Note that you need to put the "C" in quotation marks. That is necessary for character/factor variables. For numerical variables, you would leave the quotation marks out.

- `f1vB=dat[(dat$FARM=="Farm1") & (dat$VARIETY=="B"),]` will just select variety B from farm 1

- Remembering the general format of table$variable, you could now check, say, the mean of yield for this subset `mean(f1vB$YIELD)`.

2.4. Calculations and transformations in data tables

Although there are other options in R, carrying out any type of calculation on your data table should be done with the transform command, which has the syntax:

transform(table, operation1, operation2, …)

For example, we may want to change the units from kg/acre to kg/hectares or make a log-transformation of our YIELD variable:

- `transform(dat, YIELD=YIELD*0.4)` transforms the units of yield from kg/acre to kg/hectares and over-writes your original variable values.
- You could also create a new variable with a different unit and retain the old one:
  `transform(dat, YIELD_HA=YIELD*0.4)`
- You can also do two or more operations in one transform statement:
  `transform(dat, YIELD_HA=YIELD*0.4, LOG_YIELD=log(YIELD))`
- You can make this change permanent by over-writing your original dataset
  `dat=transform(dat, YIELD_HA=YIELD*0.4, LOG_YIELD=log(YIELD))`
- But generally, it's good practice to assign the results of transform operation to a new dataset, especially if you over-write variables. I like to number the resulting datasets consecutively, which allows me to go back and fix errors along the way if I do lots of modifications:
  `dat2=transform(dat, YIELD_HA=YIELD*0.4, LOG_YIELD=log(YIELD))`
- Check your new table with by typing dat2

You can also do transformations with a conditional command, which has the general syntax:

variable=ifelse(condition, "value or operation if true", "value or operation if false")

To give an imaginary example where this is useful: farmer 1 may have entered the data in kg/acre and farmer 2 in kg/ha and then send the data to you for analysis.

- Here is how you would correct this by changing the units to kg/ha only in farm1 data:

```
transform(dat, YIELD=ifelse(FARM=="Farm1",YIELD*0.4,YIELD))
```

You can do many fancy things with if-else statements, such as generating class variables from continuous data, by stacking multiple if-else statements.

- An example that creates a new class variable, indicating low, medium, and high yields:

```
transform(dat, YIELDCLASS=ifelse(YIELD<300, "Low", ifelse(YIELD<500,
"Medium", "High")))
```

Another application is the automated deletion of outliers. For example, we may know that yields cannot reasonably exceed 3000 kg/ha. If there are values larger than this, we may assume that they are typos or other errors and then decide to remove them automatically from the database.

- Use the fix() command to edit the data table and create an unreasonable yield value. Then, clean the database with this command, where NA means "missing" or "not applicable":

  `transform(dat, YIELD=ifelse(YIELD>3000,NA,YIELD))` (Note: This can be verified by checking the mean after you add the outlier, then again after you remove it).

2.5. Merging multiple data tables

One reason why you always want to use a unique ID for your research subjects (or rows in the data table) is that you can use IDs to reliably add new measurements to an existing database. You can do that by creating your own column with a unique identifier, for example:

```
dat$ID=1:24
```
(Where you know the length of each column in your data frame)

or

```
dat$ID<-1:dim(dat)[1]
```
(Where you want R to figure out the dimensions for you)

For future reference, you could also create a unique identifier based on other columns, for example:

```
dat$ID2=paste("FARM",dat$FARM,dat$VARIETY,dat$ID,sep= "_")
```

For example, in addition to the yield measurement, you may also get protein content for your lentil varieties. However, this is an expensive measurement and you could only afford to pay a lab for 6 data points (two for each variety from Farm 1). Here is the lab report:

ID:	1	2	5	6	9	10
PROTEIN:	23	43	66	46	51	29

You can add these data as columns in Excel, then:

- Import these data as a second data table to R, called “newdat”.
- Then, merge it into the existing database with the following command:

```
dat3=merge(dat, newdat, by="ID", all.x=T, all.y=F
```

- The options “all.x=T, all.y=F” means that you want to keep all observations from the first dataset (x refers to dat, T means TRUE) but only matching observations from the second dataset (y is newdat, F means FALSE). This is the most common “Left-merge” operation, where you have a master dataset on the left and a secondary dataset to add on the right side of the table as additional column(s). Check the results with head(dat2).
- Similarly, you can use the merge command to add additional information into your data table. For example, we might need additional data about the two farms in the analysis. Try a left-merge command where your master is the previous data file, and the table to be merged is a spreadsheet “farminfo.csv” that contains the following information:

	Owner	Lat	Long	Elev
Farm 1:	Smith	53.43	115.33	680
Farm 2:	Thomas	52.12	116.12	920

- Finally, you may want to save your newly created master file. To export your data to your default directory, the command is:

```
write.csv(merged, "MyMergedData.csv")
```

What you have been doing in this section is essentially a conversion of a relational database into a master flat file. Relational databases consist of multiple tables (here: lentil data, protein essays, farm info) that are linked by means of common variables (here: lentil plot ID and farm number). We have merged these all together into a master table with all information that can now be used for statistical analysis.

What you have done here in a very simple example can be applied to large databases with dozens of tables, hundreds of variables, and hundreds of thousands of rows without any issues in R.