Starting with data

Data Carpentry contributors

Learning Objectives

- load external data (CSV files) in memory using the survey table (`surveys.csv`) as an example
- explore the structure and the content of the data in R
- understand what are factors and how to manipulate them

Data importing

When you start thinking about loading your Excel files and spreadsheets into R, you need to make sure that your data is well prepared to be imported.

Keep in mind:

- The first row of the spreadsheet is usually reserved for the header, while the first column is used to identify the sampling unit;
- Avoid names, values or fields with blank spaces, otherwise each word will be treated as a separate variable, resulting in larger number of elements per line in your data set.

After preparing our file we can start loading it into R.

Loading spreadsheets can be done by using basic commands: `read.csv()`, `read.csv2()` and **and** `read.table()`

read.csv() and read.csv2()

Theses functions are often used to read spreadsheets saved with the extension .csv or comma separated values.

To read .csv files that use a comma as separator symbol, you can use the `read.csv()` function. like this:

```r
# Read csv file
df <- read.csv(file = "data/spreadsheet_messy_comma.csv",
               header = TRUE,
               quote="\"")
```

```r
# Read csv file
df <- read.csv(file = "data/spreadsheet_messy_comma.csv",
               header = TRUE,
               quote="\"",
               col.names= c("A", "B", "C", "D", "E", "F"),
               na.strings = ""
               )
```

There might be another separator symbol used in your file instead of comma, for exapmle a semicolon. To load the files where fields are separated by a semicolon, you can use `read.csv2()` function.

```r
df <- read.csv2(file = "data/spreadsheet_messy_semicol.csv",
                header = TRUE,
                quote = "\"",
                dec = ",")
```

read.table()

To get more control over the options how you want to read your file, use read.table()

```r
df <- read.table("data/spreadsheet_messy_semicol.csv",
                 sep=";",
                 header = TRUE)

df <- read.table("data/spreadsheet_messy_comma.csv",
                 sep=",",
                 header = TRUE)
```

Saving data

We can save our files in .csv using the following finction:

```r
# Write .csv in R
write.csv(df, file = "results/mydata.csv")
```

Most of the time you don't want to include row names in the CSV. To omit rownames add row.names = FALSE.

```r
# Write .csv in R, omit rownames
write.csv(df, file = "results/mydata.csv", row.names = FALSE)
```

Substitute NAs with white spaces.

```r
# Write .csv in R, omit rownames
write.csv(df, file = "results/mydata.csv", row.names = FALSE, na = "")
```

Alternatively we could save our data using `write.table()` function to save data in .csv and .txt format.

```r
# Save in .csv format
write.table(df, file = "results/mydata.csv", sep = ",", row.names = FALSE, col.names = TRUE, quote = FA
```

```r
# Save in .txt (tab delimited) format
write.table(df, file = "results/mydata.txt", sep = "\t", row.names = FALSE, col.names = TRUE, quote = F
```

Presentation of the Survey Data

We are studying the species and weight of animals caught in plots in our study area. The dataset is stored as a `csv` file: each row holds information for a single animal, and the columns represent:

Column	Description
record_id	Unique id for the observation
month	month of observation
day	day of observation
year	year of observation
plot_id	ID of a particular plot
species_id	2-letter code
sex	sex of animal ("M", "F")
hindfoot_length	length of the hindfoot in mm
weight	weight of the animal in grams
genus	genus of animal
species	species of animal
taxa	e.g. Rodent, Reptile, Bird, Rabbit
plot_type	type of plot

We are going to use the R function `download.file()` to download the CSV file that contains the survey data from figshare, and we will use `read.csv()` to load into memory (as a `data.frame`) the content of the CSV file.

To download the data into the `data/` subdirectory, do:

```r
download.file("http://files.figshare.com/2236372/combined.csv",
              "data/portal_data_joined.csv")
```

You are now ready to load the data:

```r
surveys <- read.csv('data/portal_data_joined.csv')
```

This statement doesn't produce any output because assignment doesn't display anything. If we want to check that our data has been loaded, we can print the variable's value: `surveys`

Alternatively, wrapping an assignment in parentheses will perform the assignment and display it at the same time.

```r
(surveys <- read.csv('data/portal_data_joined.csv'))
```

Wow… that was a lot of output. At least it means the data loaded properly. Let's check the top (the first 6 lines) of this `data.frame` using the function `head()`:

```r
head(surveys)
```

```
##   record_id month day year plot_id species_id sex hindfoot_length weight
## 1         1     7  16 1977       2         NL   M              32     NA
## 2        72     8  19 1977       2         NL   M              31     NA
## 3       224     9  13 1977       2         NL                  NA     NA
## 4       266    10  16 1977       2         NL                  NA     NA
## 5       349    11  12 1977       2         NL                  NA     NA
## 6       363    11  12 1977       2         NL                  NA     NA
##     genus  species   taxa plot_type
## 1 Neotoma albigula Rodent   Control
## 2 Neotoma albigula Rodent   Control
## 3 Neotoma albigula Rodent   Control
## 4 Neotoma albigula Rodent   Control
## 5 Neotoma albigula Rodent   Control
## 6 Neotoma albigula Rodent   Control
```

An important feature of a `data.frame`is that each column is a vector, with the same type of data. We can see this when inspecting the ___str___ucture of a `data.frame` with the function `str()`:

```
str(surveys)
```

```
## 'data.frame':    34786 obs. of  13 variables:
##  $ record_id      : int  1 72 224 266 349 363 435 506 588 661 ...
##  $ month          : int  7 8 9 10 11 11 12 1 2 3 ...
##  $ day            : int  16 19 13 16 12 12 10 8 18 11 ...
##  $ year           : int  1977 1977 1977 1977 1977 1977 1977 1978 1978 1978 ...
##  $ plot_id        : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ species_id     : Factor w/ 48 levels "AB","AH","AS",..: 16 16 16 16 16 16 16 16 16 16 ...
##  $ sex            : Factor w/ 6 levels "","F","M","P",..: 3 3 1 1 1 1 1 1 3 1 ...
##  $ hindfoot_length: int  32 31 NA NA NA NA NA NA NA NA ...
##  $ weight         : int  NA NA NA NA NA NA NA NA 218 NA ...
##  $ genus          : Factor w/ 26 levels "Ammodramus","Ammospermophilus",..: 13 13 13 13 13 13 13 13
##  $ species        : Factor w/ 40 levels "albigula","audubonii",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ taxa           : Factor w/ 4 levels "Bird","Rabbit",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ plot_type      : Factor w/ 5 levels "Control","Long-term Krat Exclosure",..: 1 1 1 1 1 1 1 1 1 1
```

Challenge

Based on the output of `str(surveys)`, can you answer the following questions?

- What is the class of the object `surveys`?
- How many rows and how many columns are in this object?
- How many species have been recorded during these surveys?

As you can see, many of the columns consist of integers, however, the columns `species` and `sex` are of a special class called a `factor`. Before we learn more about the `data.frame` class, we are going to talk about factors. They are very useful but not necessarily intuitive, and therefore require some attention.

Factors

Factors are used to represent categorical data. Factors can be ordered or unordered and are an important class for statistical analysis and for plotting.

Factors are stored as integers, and have labels associated with these unique integers. While factors look (and often behave) like character vectors, they are actually integers under the hood, and you need to be careful when treating them like strings.

Once created, factors can only contain a pre-defined set values, known as *levels*. By default, R always sorts *levels* in alphabetical order. For instance, if you have a factor with 2 levels:

```
sex <- factor(c("male", "female", "female", "male"))
```

R will assign `1` to the level `"female"` and `2` to the level `"male"` (because `f` comes before `m`, even though the first element in this vector is `"male"`). You can check this by using the function `levels()`, and check the number of levels using `nlevels()`:

```
levels(sex)
```

```
## [1] "female" "male"
```

```
nlevels(sex)
```

```
## [1] 2
```

Sometimes, the order of the factors does not matter, other times you might want to specify the order because it is meaningful (e.g., "low", "medium", "high") or it is required by particular type of analysis. Additionally, specifying the order of the levels allows us to compare levels:

```
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)
```

```
## [1] "high"   "low"    "medium"
```

```
food <- factor(food, levels=c("low", "medium", "high"))
levels(food)
```

```
## [1] "low"    "medium" "high"
```

```
min(food) ## doesn't work
```

```
## Error in Summary.factor(structure(c(1L, 3L, 2L, 3L, 1L, 2L, 3L), .Label = c("low", : 'min' not meanin
```

```
food <- factor(food, levels=c("low", "medium", "high"), ordered=TRUE)
levels(food)
```

```
## [1] "low"    "medium" "high"
```

```
min(food) ## works!
```

```
## [1] low
## Levels: low < medium < high
```

In R's memory, these factors are represented by numbers (1, 2, 3). They are better than using simple integer labels because factors are self describing: `"low"`, `"medium"`, `"high"`" is more descriptive than `1`, `2`, `3`. Which is low? You wouldn't be able to tell with just integer data. Factors have this information built in. It is particularly helpful when there are many levels (like the species in our example data set).

Converting factors

If you need to convert a factor to a character vector, simply use `as.character(x)`.

Converting a factors where the levels appear as numbers (such as in concentration levels) to a numeric vector is however a little trickier, and you have to go via a character vector. Compare:

```
f <- factor(c(1, 5, 10, 2))
as.numeric(f)               ## wrong! and there is no warning...
```

```
## [1] 1 3 4 2
```

```
as.numeric(as.character(f)) ## works...
```

```
## [1]  1  5 10  2
```

```
as.numeric(levels(f))[f]    ## The recommended way.
```

```
## [1]  1  5 10  2
```

Notice that in this last approach, three important steps have happened * We have obtained the factor levels using `levels(f)` * We have converted these levels to numeric values using `as.numeric(levels(f))` * We have then accessed these numeric values using the underlying integers of the vector `f` inside the square brackets

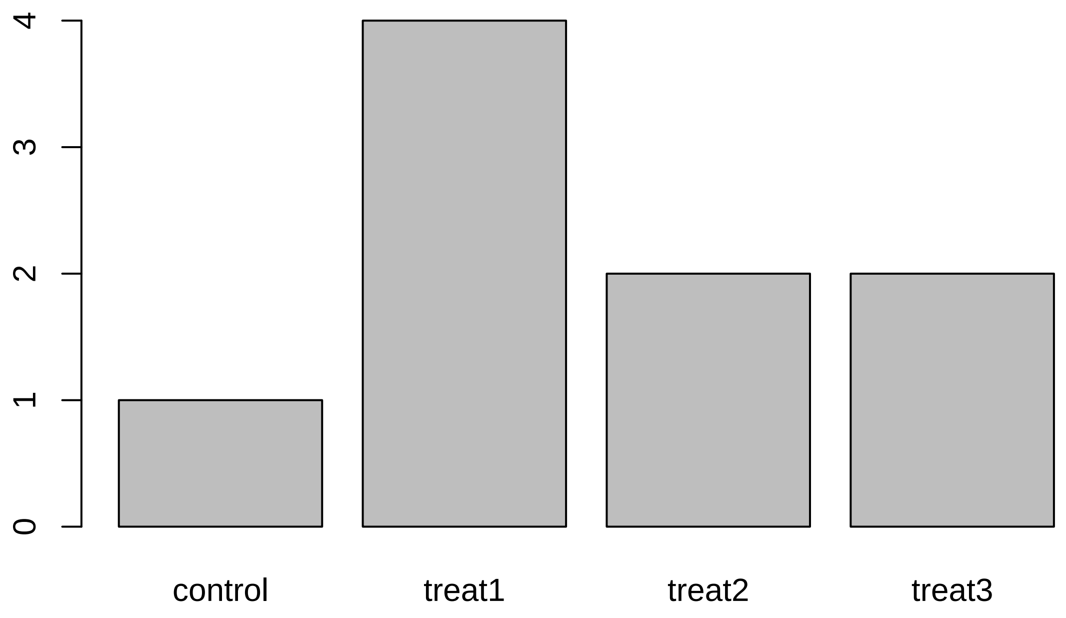
Challenge

The function `table()` tabulates observations and can be used to create bar plots quickly. For instance, the code below gives you a barplot of the number of observations. How can you recreate this plot with "control" listed last instead of first?

```
## Challenge
##
## How can you recreate this plot with "control" listed
## last instead of first?
exprmt <- factor(c("treat1", "treat2", "treat1", "treat3", "treat1", "control",
                   "treat1", "treat2", "treat3"))
table(exprmt)
```

```
## exprmt
## control  treat1  treat2  treat3
##       1       4       2       2
```

```
barplot(table(exprmt))
```


0
1
2
3
4
control
treat1
treat2
treat3

—>