Starting with data

DC contributors and Dmytro Fishman

Learning Objectives

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

Presentation of the Global Temperature Data

We are studying the temperature. The dataset is stored as a CSV file: each row holds information for a single measurement starting from 18 centuary and until now, and the columns represent:

Column	Description
record_id	Unique id for the measurement
month	month of measurement
day	day of measurement
year	year of measurement
AverageTemperatureFahr	regional average land temperature
AverageTemperatureUncertaintyFahr	the 95% confidence interval around the average
City	city where temperature was measured
country_id	three first capitalised letters in the name of the country
Country	country where temperature was measured
Latitude	angular distance, of a point north or south from equator
Longitude	angular distance, of a point east or west from Greenwich

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("https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/4938964/temperature.csv",
              "data/temperature.csv")
```

You are now ready to load the data:

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

This statement doesn't produce any output because, as you might recall, assignment doesn't display anything. If we want to check that our data has been loaded, we can print the variable's value: `temperature`. Alternatively, wrapping an assignment in parentheses will perform the assignment and display it at the same time.

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

```
#>   record_id month day year AverageTemperatureFahr
#> 1    474376     1   1 1853                     NA
#> 2    474377     2   1 1853                     NA
#> 3    474378     3   1 1853                     NA
#> 4    474379     4   1 1853                     NA
#> 5    474380     5   1 1853                     NA
#> 6    474381     6   1 1853                51.9062
#>   AverageTemperatureUncertaintyFahr     City country_id     Country
#> 1                                NA Auckland        NEW New Zealand
#> 2                                NA Auckland        NEW New Zealand
#> 3                                NA Auckland        NEW New Zealand
#> 4                                NA Auckland        NEW New Zealand
#> 5                                NA Auckland        NEW New Zealand
#> 6                           36.9572 Auckland        NEW New Zealand
#>   Latitude Longitude
#> 1   36.17S   175.03E
#> 2   36.17S   175.03E
#> 3   36.17S   175.03E
#> 4   36.17S   175.03E
#> 5   36.17S   175.03E
#> 6   36.17S   175.03E
```

A `data.frame` is the representation of data in the format of a table where the columns are vectors that all have the same length. Because each column is a vector, they all contain the same type of data. We can see this when inspecting the ___str___ucture of a `data.frame` with the function `str()`:

```r
str(temperature)
```

Challenge

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

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

As you can see, many of the columns consist of integers, however, the columns `City`, `Country` and `country_id` are of a special class called a `factor`. Before we learn more about the `data.frame` class, let's 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 understanding them is necessary 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 of 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)
nlevels(sex)
```

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 a particular type of analysis. Additionally, specifying the order of the levels allows for level comparison:

```
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)
food <- factor(food, levels=c("low", "medium", "high"))
levels(food)
min(food) ## doesn't work
```

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

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

In R's memory, these factors are represented by integers (1, 2, 3), but are more informative than integers 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 just from the integer data. Factors, on the other hand, 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, you use `as.character(x)`.

Converting factors where the levels appear as numbers (such as concentration levels) to a numeric vector is a little trickier. One method is to convert factors to characters and then numbers. Another method is to use the `levels()` function. Compare:

```
f <- factor(c(1, 5, 10, 2))
as.numeric(f)               ## wrong! and there is no warning...
as.numeric(as.character(f)) ## works...
as.numeric(levels(f))[f]    ## The recommended way.
```

Notice that in the `levels()` approach, three important steps occur:

- We obtain all the factor levels using `levels(f)`
- We convert these levels to numeric values using `as.numeric(levels(f))`
- We then access these numeric values using the underlying integers of the vector `f` inside the square brackets

Challenge

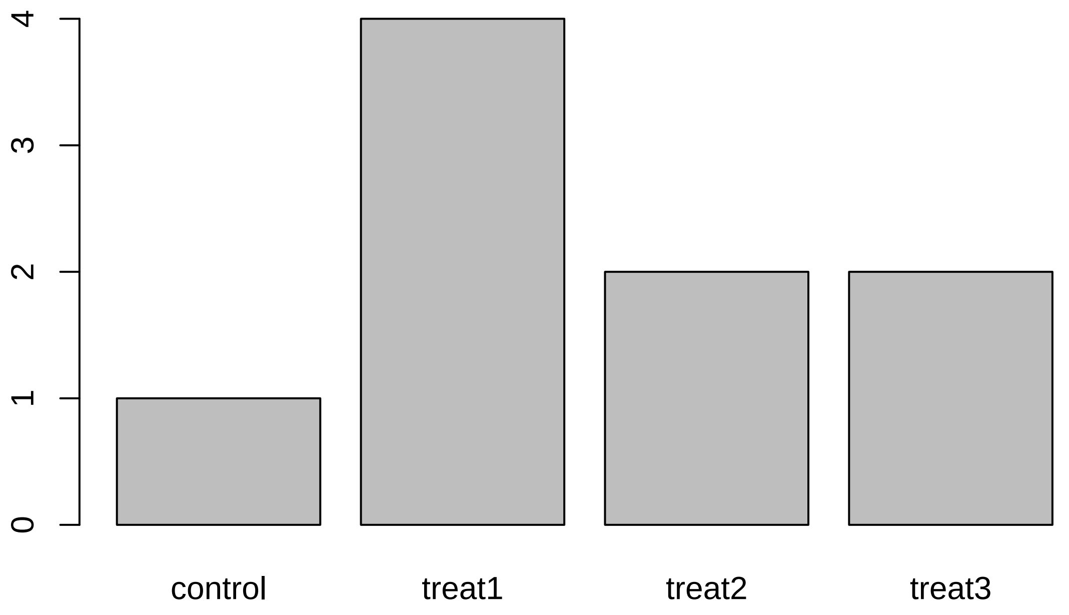
The function `table()` tabulates observations and can be used to create bar plots quickly. For instance, the code below gives you a barplot of the number of observations.

- In which order are the treatments listed?
- How can you recreate this plot with "control" listed last instead of first?

```r
exprmt <- factor(c("treat1", "treat2", "treat1", "treat3", "treat1", "control",
                   "treat1", "treat2", "treat3"))
table(exprmt)
```

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

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



```r
#Answers

# * The treatments are listed in alphabetical order because they are factors.
# * By redefining the order of the levels
#exprmt <- factor(exprmt, levels=c("treat1", "treat2", "treat3", "control"))
#barplot(table(exprmt))
```