Introduction to Data Frames (DataCamp lesson)

What's a data frame?

You may remember from the chapter about matrices that all the elements that you put in a matrix should be of the same type. Back then, your dataset on Star Wars only contained numeric elements.

When doing a market research survey, however, you often have questions such as:

- 'Are you married?' or 'yes/no' questions (`logical`)
- 'How old are you?' (`numeric`)
- 'What is your opinion on this product?' or other 'open-ended' questions (`character`)
- …

The output, namely the respondents' answers to the questions formulated above, is a dataset of different data types. You will often find yourself working with datasets that contain different data types instead of only one.

A data frame has the variables of a dataset as columns and the observations as rows. This will be a familiar concept for those coming from different statistical software packages such as SAS or SPSS.

Quick, have a look at your dataset

Working with large datasets is not uncommon in data analysis. When you work with (extremely) large datasets and data frames, your first task as a data analyst is to develop a clear understanding of its structure and main elements. Therefore, it is often useful to show only a small part of the entire dataset.

So how to do this in R? Well, the function `head()` enables you to show the first observations of a data frame. Similarly, the function `tail()` prints out the last observations in your dataset.

Both `head()` and `tail()` print a top line called the 'header', which contains the names of the different variables in your dataset.

Have a look at the structure

Another method that is often used to get a rapid overview of your data is the function `str()`. The function `str()` shows you the structure of your dataset. For a data frame it tells you:

- The total number of observations (e.g. 32 car types)
- The total number of variables (e.g. 11 car features)
- A full list of the variables names (e.g. `mpg`, `cyl` …)
- The data type of each variable (e.g. `num`)
- The first observations

Applying the `str()`] function will often be the first thing that you do when receiving a new dataset or data frame. It is a great way to get more insight in your dataset before diving into the real analysis.

Creating a data frame

Since using built-in datasets is not even half the fun of creating your own datasets, the rest of this chapter is based on your personally developed dataset. Put your jet pack on because it is time for some space exploration!

As a first goal, you want to construct a data frame that describes the main characteristics of eight planets in our solar system. According to your good friend Buzz, the main features of a planet are:

- The type of planet (Terrestrial or Gas Giant).
- The planet's diameter relative to the diameter of the Earth.
- The planet's rotation across the sun relative to that of the Earth.
- If the planet has rings or not (TRUE or FALSE).

After doing some high-quality research, you feel confident enough to create the necessary vectors: `name`, `type`, `diameter`, `rotation` and `rings`; these vectors have already been coded up in the editor. The first element in each of these vectors corresponds to the first observation.

You construct a data frame with the `data.frame()` function. As arguments, you pass the vectors from before: they will become the different columns of your data frame. Because every column has the same length, the vectors you pass should also have the same length. But don't forget that it is possible (and likely) that they contain different types of data.

The `planets_df` data frame should have 8 observations and 5 variables.

Selection of data frame elements

Similar to vectors and matrices, you select elements from a data frame with the help of square brackets `[ ]`. By using a comma, you can indicate what to select from the rows and the columns respectively. For example:

- `my_df[1,2]` selects the value at the first row and second column in `my_df`.
- `my_df[1:3,2:4]` selects rows 1, 2, 3 and columns 2, 3, 4 in `my_df`.

Sometimes you want to select all elements of a row or column. For example, `my_df[1, ]` selects all elements of the first row. Let us now apply this technique on `planets_df`!

Instead of using numerics to select elements of a data frame, you can also use the variable names to select columns of a data frame.

Suppose you want to select the first three elements of the `type` column. One way to do this is

```
planets_df[1:3,2]
```

A possible disadvantage of this approach is that you have to know (or look up) the column number of `type`, which gets hard if you have a lot of variables. It is often easier to just make use of the variable name:

```
Planets_df[1:3,"type"]
```

Only planets with rings

You will often want to select an entire column, namely one specific variable from a data frame. If you want to select all elements of the variable `diameter`, for example, both of these will do the trick:

```
planets_df[,3]
planets_df[,"diameter"]
```

However, there is a short-cut. If your columns have names, you can use the `$` sign:

```
planets_df$diameter
```

You probably remember from high school that some planets in our solar system have rings and others do not. Unfortunately you can not recall their names. Could R help you out?

If you type `rings_vector` in the console, you get:

```
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

This means that the first four observations (or planets) do not have a ring (`FALSE`), but the other four do (`TRUE`). However, you do not get a nice overview of the names of these planets, their diameter, etc. Let's try to use `rings_vector` to select the data for the four planets with rings.

Only planets with rings but shorter

So what exactly did you learn in the previous exercises? You selected a subset from a data frame (`planets_df`) based on whether or not a certain condition was true (rings or no rings), and you managed to pull out all relevant data. Pretty awesome! By now, NASA is probably already flirting with your CV ;-).

Now, let us move up one level and use the function `subset()`. You should see the `subset()` function as a short-cut to do exactly the same as what you did in the previous exercises.

```
subset(my_df, subset = some_condition)
```

The first argument of `subset()` specifies the dataset for which you want a subset. By adding the second argument, you give R the necessary information and conditions to select the correct subset.

The code below will give the exact same result as you got in the previous exercise, but this time, you didn't need the `rings_vector`!

```
subset(planets_df, subset = rings)
```

Sorting

Making and creating rankings is one of mankind's favorite affairs. These rankings can be useful (best universities in the world), entertaining (most influential movie stars) or pointless (best 007 look-a-like).

In data analysis you can sort your data according to a certain variable in the dataset. In R, this is done with the help of the function `order()`.

`order()` is a function that gives you the ranked position of each element when it is applied on a variable, such as a vector for example:

```
a <- c(100, 10, 1000)
```

```
order(a)
```

```
[1] 2 1 3
```

10, which is the second element in `a`, is the smallest element, so 2 comes first in the output of `order(a)`. 100, which is the first element in `a` is the second smallest element, so 1 comes second in the output of `order(a)`.

This means we can use the output of `order(a)` to reshuffle `a`:

```
a[order(a)]
```

```
[1]   10  100 1000
```

Sorting your data frame

Alright, now that you understand the `order()` function, let us do something useful with it. You would like to rearrange your data frame such that it starts with the smallest planet and ends with the largest one. A sort on the `diameter` column.