5: Data frames

Introduction to R

This assignment consists of six parts:

- Intro to Basics
- Vectors
- Matrices
- Factors
- Data frames (this document)
- Lists

If you haven't already, download the answer sheet for this document (**Note:** right-click and save the file with the `.Rmd` extension to your `hw02` folder.)

After you complete each exercise, push the R Notebook to your remote repo. See Part 0 for instructions. Do *not* push this document.

5.1 What is 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 data set 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 data set of different data types. You will often find yourself working with data sets that contain different data types instead of only one.

A data frame has the variables of a data set 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.

Instructions

Run the code below. The data from the built-in example data frame `mtcars` will be printed.

```
# Print out built-in R data frame
mtcars
```

5.2 Quick, have a look at your data set

Wow, that is a lot of cars!

Working with large data sets is not uncommon in data analysis. When you work with data sets 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 data set.

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 data set.

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

Instructions

Call `head()` on the `mtcars` data set to have a look at the header and the first observations.

```
# Call head() on mtcars


#
```

5.3 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 data set. 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 data set or data frame. It is a great way to get more insight in your data set before diving into the real analysis.

Instructions

Investigate the structure of `mtcars`. Make sure that you see the same numbers, variables and data types as mentioned above.

```
# Investigate the structure of mtcars


#
```

5.4 Creating a data frame

Since using built-in data sets is not even half the fun of creating your own data sets, the rest of this chapter is based on your personally developed data set. 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 on Wikipedia, you feel confident enough to create the necessary vectors: `name`, `type`, `diameter`, `rotation` and `rings`; these vectors have already been coded up below. The first element in each of these vectors correspond 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.

Instructions

Use the function `data.frame()` to construct a data frame. Pass the vectors `name`, `type`, `diameter`, `rotation` and `rings` as arguments to `data.frame()`, in this order. Call the resulting data frame `planets_df`.

```
# Definition of vectors
name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
          "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)

# Create a data frame from the vectors


#
```

5.5 Creating a data frame (2)

The `planets_df` data frame should have 8 observations and 5 variables. It is in your workspace now, so you do not have to recreate it again.

Instructions

Use `str()` to investigate the structure of the new `planets_df` variable.

```
# Check the structure of planets_df


#
```

5.6 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`!

Instructions

- From `planets_df`, select the diameter of Mercury: this is the value at the first row and the third column. Simply print out the result.
- From `planets_df`, select all data on Mars (the fourth row). Simply print out the result.

```
# The planets_df data frame from the previous exercise is pre-loaded

# Print out diameter of Mercury (row 1, column 3)


# Print out data for Mars (entire fourth row)


#
```

5.7 Selection of data frame elements (2)

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"]
```

Instructions

Select and print out the first 5 values in the `"diameter"` column of `planets_df`.

```
# The planets_df data frame from the previous exercise is available

# Select first 5 values of diameter column


#
```

5.8 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
```

Instructions

- Use the `$` sign to select the rings variable from `planets_df`. Store the vector that results as `rings_vector`.
- Print out `rings_vector` to see if you got it right.

```
# planets_df is pre-loaded in your workspace

# Select the rings variable from planets_df


# Print out rings_vector


#
```

5.9 Only planets with rings (2)

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.

Instructions

The code below selects the `name` column of all planets that have rings. Adapt the code so that instead of only the `name` column, all columns for planets that have rings are selected.

```
# planets_df and rings_vector are pre-loaded in your workspace

# Adapt the code to select all columns for planets with rings
planets_df[name, ]
```

5.10 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 data set 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)
```

Instructions

Use `subset()` on `planets_df` to select planets that have a diameter smaller than Earth. Because the `diameter` variable is a relative measure of the planet's diameter w.r.t that of planet Earth, your condition is `diameter < 1`.

```
# planets_df is pre-loaded in your workspace

# Select planets with diameter < 1


#
```

5.11 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 data set. 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
```

More often, you might save the order to a variable, and then use the variable to set the order, as in this example. Here, I am using the `head()` function to show only the first 8 rows of the data frame.

```
mpg_rank <- order(mtcars$mpg)
head(mtcars[mpg_rank, ], 8)
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
```

Clearly, the Cadillac and Lincoln are good gas guzzlers. What if you want a more economical option?

Instructions

- Read and run this example. What does the argument `decreasing = TRUE` do?

```r
mpg_rank <- order(mtcars$mpg, decreasing = TRUE)
head(mtcars[mpg_rank, ], 8)
```

5.12 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.

Instructions

- Call `order()` on `planets_df$diameter` (the diameter column of planets_df). Store the result as `positions`.
- Sort `planets_df` with the positions vector as row indexes inside square brackets. Keep all columns. Simply print out the result.
- Call `order()` again on `planets_df$rotation`, with the highest value first. Store the result in `rot_position`.
- Sort `planets_df` as you did above, but use `rot_position` as the row index. Print out the result.

```r
# planets_df is pre-loaded in your workspace

# Use order() to create positions


# Use positions to sort planets_df


# Use order() to create rotation positions


# Use rot_position to sort planets_df


#
```