> Note on the data used in this howto: This data can be downloaded from http://piketty.pse.ens.fr/files/capital21c/en/xls/, but the excel format is a bit difficult to parse at it is meant to be human readable, with multiple header rows etc. For that reason, I've extracted csv files for some interesting tables that I've uploaded to https://github.com/vanatteveldt/learningr/tree/master/data. If you're accessing this tutorial from the githup project, these files should be in your 'data' sub folder automatically.

Organizing data in R

This hands-on demonstrates reading, writing, and manipulating data in R. As before, we will continue using the data from Piketty's 'Capital in the 21st Century'

```
income = read.csv("data/income_toppercentile.csv")
```

Saving and loading data

So far, we've used the `read.csv` command to read data from a CSV file. As can be guessed, there is also a `write.csv` command that writes data into a CSV file:

```
write.csv(income, file="test.csv")
test = read.csv("test.csv")
head(test)
```

X	Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
1	1900	NA	NA	NA	NA	NA	NA	NA	NA	NA
2	1901	NA	NA	NA	NA	NA	NA	NA	NA	NA
3	1902	NA	NA	NA	NA	NA	NA	NA	NA	NA
4	1903	NA	NA	NA	0.162	NA	NA	NA	NA	NA
5	1904	NA	NA	NA	NA	NA	NA	NA	NA	NA
6	1905	NA	NA	NA	NA	NA	NA	NA	NA	NA

A new column was created because by default `write.csv` also writes the row numbers (you can check this by opening test.csv in excel). Since this row number column has no header, it is given the variable name `X`. You can suppress this by adding `row.names=F` to the write.csv function:

```
write.csv(income, file="test.csv", row.names=F)
```

On european computers, excel produces (and expects) csv files to be delimited with semicolons rather then commas by default, using the comma as a decimal separator (instead of period). To facilitate this, R provides a pair of functions `read.csv2`/`write.csv2` that use this format.

If you open a CSV file using the wrong function, you will only see a single column with all the values in it. For example, if we use `read.csv2` to open the file we just created we get the following:

```
d = read.csv2("test.csv")
head(d)
```

```
##   Year.Canada.Australia.New.Zealand.Denmark.Italy.Holland.Spain.France.US
## 1                                          1900,NA,NA,NA,NA,NA,NA,NA,NA,NA
## 2                                          1901,NA,NA,NA,NA,NA,NA,NA,NA,NA
## 3                                          1902,NA,NA,NA,NA,NA,NA,NA,NA,NA
## 4                                      1903,NA,NA,NA,0.162,NA,NA,NA,NA,NA
## 5                                          1904,NA,NA,NA,NA,NA,NA,NA,NA,NA
## 6                                          1905,NA,NA,NA,NA,NA,NA,NA,NA,NA
```

The bottom line is: when using CSV data, always check your results, and use the 'European' version of the commands when appropriate.

Apart from writing csv files, R can also write to a native file format, which has the advantage of correctly storing all types of data (including numbers and date columns) and of storing multiple variables in one file.

For example, the following code stores the incomep and a new `x` variable in a file called `mydata.rdata`:

```
x = 12
save(income, x, file="mydata.rdata")
```

Now, you can clear the data from your environment, using the Clear button in RStudio or by issuing the somewhat cryptic command `rm(list=ls())`

```
rm(list=ls())
head(income)
```

```
## Error in head(income): object 'income' not found
```

And if you load the file, the variables will appear again:

```
load("mydata.rdata")
head(income)
```

Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
1900	NA	NA	NA	NA	NA	NA	NA	NA	NA
1901	NA	NA	NA	NA	NA	NA	NA	NA	NA
1902	NA	NA	NA	NA	NA	NA	NA	NA	NA
1903	NA	NA	NA	0.162	NA	NA	NA	NA	NA
1904	NA	NA	NA	NA	NA	NA	NA	NA	NA
1905	NA	NA	NA	NA	NA	NA	NA	NA	NA

Note that you do not load the file into a specific variable, as the file can contain multiple variables. The load command will automatically create those variables with their original names.

Subsetting data

The data we have downloaded into `income` contains income series from 1900 to 2010 for a number of countries. We can use hard brackets `[rows, columns]` to subset this dataset, for example to select only the first 10 rows or to only select the US and Franch data.

```
income[1:10, ]
```

```
##    Year Canada Australia New.Zealand Denmark Italy Holland Spain France US
## 1  1900     NA        NA          NA      NA    NA      NA    NA     NA NA
## 2  1901     NA        NA          NA      NA    NA      NA    NA     NA NA
## 3  1902     NA        NA          NA      NA    NA      NA    NA     NA NA
## 4  1903     NA        NA          NA   0.162    NA      NA    NA     NA NA
## 5  1904     NA        NA          NA      NA    NA      NA    NA     NA NA
## 6  1905     NA        NA          NA      NA    NA      NA    NA     NA NA
## 7  1906     NA        NA          NA      NA    NA      NA    NA     NA NA
## 8  1907     NA        NA          NA      NA    NA      NA    NA     NA NA
## 9  1908     NA        NA          NA   0.165    NA      NA    NA     NA NA
## 10 1909     NA        NA          NA      NA    NA      NA    NA     NA NA
```

```
subset = income[, c("US", "France")]
head(subset)
```

US	France
NA	NA
NA	NA
NA	NA
NA	NA
NA	NA
NA	NA

A more common use case is that we want to select based on specific criteria. Suppose that we are now only interested in the series for the US, and France since 1945. We can place an expression in the rows selector to subset the data like that:

```
subset = income[income$Year > 1945, c("Year", "US", "France")]
head(subset)
```

	Year	US	France
47	1946	0.133	0.092
48	1947	0.120	0.092
49	1948	0.122	0.088
50	1949	0.117	0.090
51	1950	0.128	0.090
52	1951	0.118	0.090

An easy 'shortcut' to subset data by selecting only certain rows is by using the `subset` function. This function allows you to specify criteria using column names directly, i.e. without using the `dataset$` prefix:

```r
postwar = subset(income, Year > 1945)
head(postwar)
```

	Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
47	1946	0.107	0.095	0.075	0.106	NA	0.129	0.113	0.092	0.133
48	1947	0.110	0.106	0.077	0.107	NA	NA	0.100	0.092	0.120
49	1948	0.104	0.108	0.077	0.099	NA	NA	0.097	0.088	0.122
50	1949	0.107	0.113	0.080	0.097	NA	NA	0.096	0.090	0.117
51	1950	0.109	0.121	0.094	0.094	NA	0.121	0.088	0.090	0.128
52	1951	0.100	0.091	0.079	0.093	NA	NA	0.082	0.090	0.118

Ordering data

The easiest way to order data is using the `arrange` command from the `plyr` package. If you haven't installed `plyr` yet, you can install it with:

```r
install.packages("plyr")
```

Now, you can use the arrange command which, like the subset command, allows you to use columns directly. For example, this code orders the income dataset by ascending inequality for France:

```r
library(plyr)
income = arrange(income, France)
head(income)
```

Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
1983	0.082	0.047	0.057	0.053	0.063	NA	0.077	0.070	0.116
1984	0.083	0.048	0.056	0.053	0.065	NA	0.076	0.070	0.120
1982	0.085	0.047	0.055	0.052	0.064	NA	0.078	0.071	0.108
1985	0.082	0.050	0.055	0.052	0.068	0.059	0.078	0.072	0.127
1986	0.082	0.054	0.049	0.052	0.071	NA	0.082	0.074	0.159
1945	0.101	0.084	0.069	0.114	NA	NA	0.118	0.075	0.125

So, the years with the most equal income distribution in France are in the 1980's, after which inequality was again on the increase.

You can also specify multiple columns by just adding extra arguments. The named argument `decreasing=TRUE` makes it easy to sort in descending order. Finally, note that you can always using `plyr::rearrange` without first using the `library(plyr)` command. Thus, the following code sorts by decreasing inequality in Canada and then by decreasing year:

```r
income = plyr::arrange(income, Canada, Year, decreasing=T)
head(income)
```

Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
1938	0.184	0.104	0.073	0.133	NA	0.157	NA	0.143	0.158
1933	0.180	0.103	0.109	0.139	NA	0.142	0.139	0.150	0.165

Year	Canada	Australia	New.Zealand	Denmark	Italy	Holland	Spain	France	US
1932	0.177	0.093	NA	0.135	NA	0.144	NA	0.148	0.156
1921	0.176	0.116	0.113	0.128	NA	0.183	NA	0.173	0.156
1936	0.175	0.113	0.107	0.144	NA	0.148	NA	0.147	0.193
1934	0.175	0.104	0.104	0.144	NA	0.140	0.139	0.153	0.164

Calculating columns

We saw earlier that you can store the result of a calculation in a new variable. You can also create a new column by storing the result of a calculation in a column. For example, we could create an column for the average of US and French inequality:

```
subset$average = (subset$US + subset$France) / 2
head(subset)
```

	Year	US	France	average
47	1946	0.133	0.092	0.1125
48	1947	0.120	0.092	0.1060
49	1948	0.122	0.088	0.1050
50	1949	0.117	0.090	0.1035
51	1950	0.128	0.090	0.1090
52	1951	0.118	0.090	0.1040

It is also possible to replace part of a column. For example, we can set the average to NA when the French value is lower than 0.09 like so:

```
subset$average[subset$France < 0.09] = NA
head(subset)
```

	Year	US	France	average
47	1946	0.133	0.092	0.1125
48	1947	0.120	0.092	0.1060
49	1948	0.122	0.088	NA
50	1949	0.117	0.090	0.1035
51	1950	0.128	0.090	0.1090
52	1951	0.118	0.090	0.1040

What you are doing there is in fact assigning `NA` to a subset of the column, selected using the France column. Becoming good at R for a large part means becoming good at using the subsetting and assignment operations, so take some time to understand and play around with this code.

Dealing with Missing Values

Finally, a useful function is `is.na`. This function is true when it's argument is NA (i.e., missing):

```
is.na(subset$average)
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

As you can see, it is true for the thrid row and for most rows past the 23d. In fact, an expression lik `subset$average > 3` also returns such a vector of logical values:

```
subset$US > .11
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

This result is `TRUE` for those years where the income inequality in the US is larger than .11. Just as we can use `subset$France < 0.09` to selectively replace certain cells, we can do so with `is.na`:

```
subset$average[is.na(subset$average)] = 0
head(subset)
```

	Year	US	France	average
47	1946	0.133	0.092	0.1125
48	1947	0.120	0.092	0.1060
49	1948	0.122	0.088	0.0000
50	1949	0.117	0.090	0.1035
51	1950	0.128	0.090	0.1090
52	1951	0.118	0.090	0.1040

This command tells R to replace every cell in the average column where the average is missing with zero. Since sometimes NA values are really zero, this is quite a useful command. We can also use this to remove NA rows, similar to the na.omit command used earlier but more flexible. Let's first introduce our NA's again:

```
subset$average[subset$France < 0.09] = NA
head(subset)
```

	Year	US	France	average
47	1946	0.133	0.092	0.1125
48	1947	0.120	0.092	0.1060
49	1948	0.122	0.088	NA
50	1949	0.117	0.090	0.1035
51	1950	0.128	0.090	0.1090
52	1951	0.118	0.090	0.1040

And now use `!is.na` to select certain rows in the data frame (an exclamation mark (read as NOT) inverts a selection)

```
subset.nomissing = subset[!is.na(subset$average), ]
head(subset.nomissing)
```

	Year	US	France	average
47	1946	0.133	0.092	0.1125
48	1947	0.120	0.092	0.1060
50	1949	0.117	0.090	0.1035
51	1950	0.128	0.090	0.1090
52	1951	0.118	0.090	0.1040
53	1952	0.108	0.092	0.1000

As you can see, row 49 is gone. Note the trailing comma in the subset command. Although we only want to select on rows (and not on columns), we still need to place a comma after the row selection to complete the `[rows, columns]` pattern.

In fact, you can also use selections on a whole data frame, allowing you to replace all values under a certain condition.

```
subset[subset < .11] = NA
head(subset, n=10)
```

	Year	US	France	average
47	1946	0.133	NA	0.1125
48	1947	0.120	NA	NA
49	1948	0.122	NA	NA
50	1949	0.117	NA	NA
51	1950	0.128	NA	NA
52	1951	0.118	NA	NA
53	1952	NA	NA	NA
54	1953	NA	NA	NA
55	1954	NA	NA	NA
56	1955	0.111	NA	NA

Note that here the trailing comma is not given since the selection is based on the whole data set, not just on certain rows. Similarly, the is.na function can be used to globally replace NA values in a data frame:

```
subset[is.na(subset)] = 0
head(subset)
```

	Year	US	France	average
47	1946	0.133	0	0.1125
48	1947	0.120	0	0.0000
49	1948	0.122	0	0.0000
50	1949	0.117	0	0.0000
51	1950	0.128	0	0.0000
52	1951	0.118	0	0.0000