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.

Merging and transforming data in R

Let's have a closer look at the relation at the accumulation of capital in various countries. In chapter 5, separate values are introduced for private and public capital:

```
private.capital = read.csv("data/private_capital.csv")
public.capital = read.csv("data/public_capital.csv")
```

Combining data using cbind

First, let's combine the two data sets into a single set. This is done using `cbind`, which binds multiple columns (vectors or data frames) into a single data frame. It only works if the data is alinged correctly and has the same amount of rows, so first we check this:

```
nrow(private.capital)
```

```
## [1] 41
```

```
nrow(public.capital)
```

```
## [1] 41
```

```
table(private.capital$Year == public.capital$Year)
```

```
##
## TRUE
##   41
```

So they both have 41 rows and in all rows the Year variables are identical. Thus, we can use cbind here, omitting the first column of public.capital.

```
capital = cbind(private.capital, public.capital[-1])
head(capital)
```

```
##   Year U.S. Japan Germany France U.K. Italy Canada Australia Spain U.S.
## 1 1970 3.42  2.99    2.25   3.10 3.06  2.39   2.47      3.30    NA 0.61
## 2 1971 3.41  3.28    2.20   3.04 3.28  2.45   2.52      3.38    NA 0.64
## 3 1972 3.49  3.73    2.22   3.07 3.54  2.58   2.51      3.44    NA 0.64
## 4 1973 3.39  4.04    2.18   3.05 3.40  2.53   2.46      3.47    NA 0.65
## 5 1974 3.21  3.96    2.20   3.03 3.37  2.82   2.39      3.48    NA 0.74
## 6 1975 3.20  3.86    2.29   3.17 3.01  3.21   2.42      3.49    NA 0.77
##   Japan Germany France U.K. Italy Canada Australia Spain
## 1  0.61    0.88   0.41 0.59  0.20   0.37      0.61    NA
## 2  0.67    0.89   0.43 0.69  0.16   0.39      0.65    NA
## 3  0.71    0.88   0.45 0.79  0.11   0.40      0.68    NA
## 4  0.75    0.88   0.46 0.93  0.10   0.40      0.71    NA
## 5  0.81    0.90   0.48 1.11  0.11   0.44      0.78    NA
## 6  0.82    0.87   0.53 1.02  0.05   0.46      0.84    NA
```

Renaming variables

As could be seen in the `head` output, the column names are identical for both data sets, which causes a problem. So, let's prepend the last 10 column names with `public.` using the `paste` function, which pastes two texts together.

```r
colnames(capital)[11:19] = paste("public", colnames(capital[11:19]), sep=".")
head(capital)
```

```
##   Year U.S. Japan Germany France U.K. Italy Canada Australia Spain
## 1 1970 3.42  2.99    2.25   3.10 3.06  2.39   2.47      3.30    NA
## 2 1971 3.41  3.28    2.20   3.04 3.28  2.45   2.52      3.38    NA
## 3 1972 3.49  3.73    2.22   3.07 3.54  2.58   2.51      3.44    NA
## 4 1973 3.39  4.04    2.18   3.05 3.40  2.53   2.46      3.47    NA
## 5 1974 3.21  3.96    2.20   3.03 3.37  2.82   2.39      3.48    NA
## 6 1975 3.20  3.86    2.29   3.17 3.01  3.21   2.42      3.49    NA
##   public.U.S. public.Japan public.Germany public.France public.U.K.
## 1        0.61         0.61           0.88          0.41        0.59
## 2        0.64         0.67           0.89          0.43        0.69
## 3        0.64         0.71           0.88          0.45        0.79
## 4        0.65         0.75           0.88          0.46        0.93
## 5        0.74         0.81           0.90          0.48        1.11
## 6        0.77         0.82           0.87          0.53        1.02
##   public.Italy public.Canada public.Australia public.Spain
## 1         0.20          0.37             0.61           NA
## 2         0.16          0.39             0.65           NA
## 3         0.11          0.40             0.68           NA
## 4         0.10          0.40             0.71           NA
## 5         0.11          0.44             0.78           NA
## 6         0.05          0.46             0.84           NA
```

/Note that using `merge` rather than `cbind` would be more natural and would prevent these problems, but it is good to know how cbind and renaming works in any case/

In order to rename variables more easily in a 'manual' fashion, the `rename` command from the `plyr` package can be used.

Note, if you get an error about missing a package, you can install it using the install.packages function (or you can use the 'Packages' panel in the bottom right in RStudio):

```r
install.packages("plyr")  # only needed the first time
```

Now, we can use `library(plyr)` to load the package, and use the rename command:

```
library(plyr)
capital = rename(capital, c("U.K."="UK", "U.S."="US", "public.U.K."="public.UK", "public.U.S."="public.U
head(capital)
```

```
##   Year   US Japan Germany France   UK Italy Canada Australia Spain
## 1 1970 3.42  2.99    2.25   3.10 3.06  2.39   2.47      3.30    NA
## 2 1971 3.41  3.28    2.20   3.04 3.28  2.45   2.52      3.38    NA
## 3 1972 3.49  3.73    2.22   3.07 3.54  2.58   2.51      3.44    NA
## 4 1973 3.39  4.04    2.18   3.05 3.40  2.53   2.46      3.47    NA
## 5 1974 3.21  3.96    2.20   3.03 3.37  2.82   2.39      3.48    NA
## 6 1975 3.20  3.86    2.29   3.17 3.01  3.21   2.42      3.49    NA
##   public.US public.Japan public.Germany public.France public.UK
## 1      0.61         0.61           0.88          0.41      0.59
## 2      0.64         0.67           0.89          0.43      0.69
## 3      0.64         0.71           0.88          0.45      0.79
## 4      0.65         0.75           0.88          0.46      0.93
## 5      0.74         0.81           0.90          0.48      1.11
## 6      0.77         0.82           0.87          0.53      1.02
##   public.Italy public.Canada public.Australia public.Spain
## 1         0.20          0.37             0.61           NA
## 2         0.16          0.39             0.65           NA
## 3         0.11          0.40             0.68           NA
## 4         0.10          0.40             0.71           NA
## 5         0.11          0.44             0.78           NA
## 6         0.05          0.46             0.84           NA
```

Variables to cases using melt

Now assume that we want to calculate the total capital (private plus public) per country. One way to do that would be to simply sum the relevant parts of the data frame:

```
total.capital = capital[, 2:10] + capital[, 11:19]
head(total.capital)
```

```
##     US Japan Germany France   UK Italy Canada Australia Spain
## 1 4.03  3.60    3.13   3.51 3.65  2.59   2.84      3.91    NA
## 2 4.05  3.95    3.09   3.47 3.97  2.61   2.91      4.03    NA
## 3 4.13  4.44    3.10   3.52 4.33  2.69   2.91      4.12    NA
## 4 4.04  4.79    3.06   3.51 4.33  2.63   2.86      4.18    NA
## 5 3.95  4.77    3.10   3.51 4.48  2.93   2.83      4.26    NA
## 6 3.97  4.68    3.16   3.70 4.03  3.26   2.88      4.33    NA
```

However, a more general way that will afford more flexibility later is to first transform both datasets from wide into long formats (e.g. convert the variables to cases), and then merge the two sets. The `reshape2` package has a function `melt` that transforms variables to cases based on a list of index variables that identify the cases.

```
library(reshape2)
public.long = melt(public.capital, id.vars="Year")
private.long = melt(private.capital, id.vars="Year")
head(public.long)
```

```
##   Year variable value
## 1 1970     U.S.  0.61
## 2 1971     U.S.  0.64
## 3 1972     U.S.  0.64
## 4 1973     U.S.  0.65
## 5 1974     U.S.  0.74
## 6 1975     U.S.  0.77
```

As you can see, we now have one row per year per country. We can now use `merge` to merge the two data frames. By default, `merge` merges data on identical column names. Since the `value` column is also shared between the data frames, and we don't want to merge on that column, we need to specify the `by=` argument:

```
capital = merge(public.long, private.long, by=c("Year", "variable"))
head(capital)
```

```
##   Year  variable value.x value.y
## 1 1970 Australia    0.61    3.30
## 2 1970    Canada    0.37    2.47
## 3 1970    France    0.41    3.10
## 4 1970   Germany    0.88    2.25
## 5 1970     Italy    0.20    2.39
## 6 1970     Japan    0.61    2.99
```

So now we have our data ready. Let's rename the column names to something more sensible, and then compute the total capital. We also save the file for later use using the `save` command.

```
colnames(capital) = c("Year", "Country", "Public", "Private")
capital$Total = capital$Private + capital$Public
save(capital, file="data/capital.rdata")
```

Variables to cases

The counterparg of melting is called casting. The `dcast` function works by using a 'casting formula' `rows ~ columns` to specify which variables to place in the rows and columns of the resulting data frame.

```
d = dcast(capital, Year ~ Country, value.var="Total")
head(d)
```

```
##   Year U.S. Japan Germany France U.K. Italy Canada Australia Spain
## 1 1970 4.03  3.60    3.13   3.51 3.65  2.59   2.84      3.91    NA
## 2 1971 4.05  3.95    3.09   3.47 3.97  2.61   2.91      4.03    NA
## 3 1972 4.13  4.44    3.10   3.52 4.33  2.69   2.91      4.12    NA
## 4 1973 4.04  4.79    3.06   3.51 4.33  2.63   2.86      4.18    NA
## 5 1974 3.95  4.77    3.10   3.51 4.48  2.93   2.83      4.26    NA
## 6 1975 3.97  4.68    3.16   3.70 4.03  3.26   2.88      4.33    NA
```

So now we have a data frame that we can plot, e.g. using a for loop as above. `cast` can also work to make aggregate functions. Suppose we would want to make a plot per decade rather than per year, we first make a new 'decade' value:

```
capital$Decade = floor(capital$Year / 10) * 10
```

A good way to check whether a recode such as this succeeded is to tabulate decade and year:

```
table(capital$Decade, capital$Year)
```

```
##
##         1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982
##    1970    9    9    9    9    9    9    9    9    9    9    0    0    0
##    1980    0    0    0    0    0    0    0    0    0    0    9    9    9
##    1990    0    0    0    0    0    0    0    0    0    0    0    0    0
##    2000    0    0    0    0    0    0    0    0    0    0    0    0    0
##    2010    0    0    0    0    0    0    0    0    0    0    0    0    0
##
##         1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
##    1970    0    0    0    0    0    0    0    0    0    0    0    0    0
##    1980    9    9    9    9    9    9    9    0    0    0    0    0    0
##    1990    0    0    0    0    0    0    0    9    9    9    9    9    9
##    2000    0    0    0    0    0    0    0    0    0    0    0    0    0
##    2010    0    0    0    0    0    0    0    0    0    0    0    0    0
##
##         1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008
##    1970    0    0    0    0    0    0    0    0    0    0    0    0    0
##    1980    0    0    0    0    0    0    0    0    0    0    0    0    0
##    1990    9    9    9    9    0    0    0    0    0    0    0    0    0
##    2000    0    0    0    0    9    9    9    9    9    9    9    9    9
##    2010    0    0    0    0    0    0    0    0    0    0    0    0    0
##
##         2009 2010
##    1970    0    0
##    1980    0    0
##    1990    0    0
##    2000    9    0
##    2010    0    9
```

Now, we use the new Decade variable in the casting formula, and specify an aggregation function `mean`:

```
d = dcast(capital, Decade ~ Country, value.var="Total", fun.aggregate=mean)
head(d)
```

```
##   Decade  U.S. Japan Germany France  U.K. Italy Canada Australia Spain
## 1   1970 4.001 4.452   3.134  3.615 4.030 2.872  2.857     4.197    NA
## 2   1980 4.175 6.055   3.529  3.751 4.459 3.493  2.801     4.358    NA
## 3   1990 4.191 7.112   3.552  3.678 4.658 4.681  2.972     4.737 4.436
## 4   2000 4.920 6.273   3.876  5.028 5.248 5.570  3.576     5.665 6.771
## 5   2010 4.310 6.150   4.160  6.060 5.280 6.080  4.120     5.850 7.600
```

Aggregate vs. cast

Instead of `cast`, we can also use the `normal'`aggregate`command. Cast and aggregate have slightly different applications. Aggregate allows multiple columns to be aggregated` `in one command (but all using the same aggregation function), but produces a 'long' data format, while`cast‘ can only aggregate a single variable, but can directly create a tabular data format. For example, the following aggregates the public, private and total capital per decade and country:

```
aggregated = aggregate(capital[c("Public", "Private", "Total")], by=capital[c("Decade", "Country")], FU
head(aggregated)
```

```
##   Decade Country Public Private Total
## 1   1970    U.S.  0.681   3.320 4.001
## 2   1980    U.S.  0.603   3.572 4.175
## 3   1990    U.S.  0.267   3.924 4.191
## 4   2000    U.S.  0.455   4.465 4.920
## 5   2010    U.S.  0.210   4.100 4.310
## 6   1970   Japan  0.734   3.718 4.452
```

Note: the `by=` argument in aggregate needs to be a list or data frame. Since `capital[, "Decade"]` returns a vector rather than a data frame, you must use either `capital["Decade"]` (ommitting the comma) or `capital[, "Decade", drop=F]`. For example, the first attempt below to aggregate the recent values per country fails because of this problem:

```
aggregated = aggregate(capital["Total"], by=capital[, "Country"], FUN=mean)
```

```
## Error in aggregate.data.frame(capital["Total"], by = capital[, "Country"], : 'by' must be a list
```

```
aggregated = aggregate(capital["Total"], by=capital[, "Country", drop=F], FUN=mean)
head(aggregated)
```

```
##   Country    Total
## 1    U.S. 4.321463
## 2   Japan 5.977317
## 3 Germany 3.538293
## 4  France 4.067805
## 5    U.K. 4.615366
## 6   Italy 4.200976
```