R syntax in a nutshell – Part III

- 1.10 Factors
- 1.11 Data input/output, files

1.10 Factors

Factors are vector objects used for discrete classification (grouping) of components in other vectors of the same length, matrices or data frames. In statistical applications, these often serve to store categorical variables.

1.10.1 Basic usage of factors

factor(x)

The (unordered) factors are set by the function `factor` where `x` is a vector of data, usually containing a small number of discrete values (known as `levels`). In this case the levels are stored in alphabetical order. For instance:

In [16]:
```
x <- c("Pl","Bt","Pl","Pl","Kfs","Pl","Bt","Pl",NA)
x.un <- factor(x)
print(x.un)
```

```
[1] Pl   Bt   Pl   Pl   Kfs  Pl   Bt   Pl   <NA>
Levels: Bt Kfs Pl
```

ordered(x, levels)

This function defines a special type of factor in which the order of levels is defined explicitly using the namesake parameter:

In [17]:
```
x.or <- ordered(x,levels=c("Pl","Kfs","Bt"))
print(x.or)
```

```
[1] Pl   Bt   Pl   Pl   Kfs  Pl   Bt   Pl   <NA>
Levels: Pl < Kfs < Bt
```

levels(x)

Returns all possible values (levels) of the factor `x`.

In [18]:
```
levels(x.un)
levels(x.or)
```

'Bt' · 'Kfs' · 'Pl'

'Pl' · 'Kfs' · 'Bt'

 The data frame `ToothGrowth` portrays response in the teeth length of 10 guinea pigs to each of three dose levels of Vitamin C (0.5, 1, and 2 mg) supplied by two delivery

methods (orange juice, `OJ` or ascorbic acid, `VC`). It contains 60 observations on 3 variables.

In [19]:
```
data(ToothGrowth)
head(ToothGrowth) # print just a few observations
```

A data.frame: 6 × 3

	len	supp	dose
	<dbl>	<fct>	<dbl>
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5

We can define a factor `method` that shows the Vitamin C supplement method.

In [20]:
```
method <- factor(ToothGrowth[,"supp"])
print(method)
```

```
 [1] VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC VC
[26] VC VC VC VC VC OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
[51] OJ OJ OJ OJ OJ OJ OJ OJ OJ OJ
Levels: OJ VC
```

Possible values of the factor method are displayed directly using the function `levels`:

In [21]:
```
levels(method)
```

'OJ' · 'VC'

1.10.2 Calculations using factors

The factor can be now used for instance to calculate the mean tooth length for each group ("OJ" and "VC") separately. This is done using the function:

tapply (x, index, fun):

where `x` is a vector, `index` a factor (or list of two factors) and `fun` a function to be applied.

In [22]:
```
ee <- tapply(ToothGrowth[,"len"],method,mean)
print(ee)
```

```
      OJ       VC
20.66333 16.96333
```

In [23]:
```
dose <- factor(ToothGrowth[,"dose"])
ee <- tapply(ToothGrowth[,"len"],list(suppl_method=method,mg=dose),mean)
print(ee)
```

```
            mg
suppl_method   0.5     1     2
          OJ 13.23 22.70 26.06
          VC  7.98 16.77 26.14
```

1.10.2 Classification: conversion of numeric vectors to factors

In some cases it is advantageous to divide the total range of a numeric vector `x` into a certain number of discrete ranks (groups), and classify the values in `x` according to the rank they fall into. If each of these ranks is labelled by the identifying text, the result is a factor of the same length as the original vector.

cut (x, breaks, labels)

The function splits the numeric vector `x` into given number of ranks and codes its items according to the rank they fall into. The parameter `breaks` either defines the cut off points or specifies the desired number of intervals. Parameter `labels` may provide optional names for individual ranks.

In [24]:
```
print(max(ToothGrowth[,"len"]))
```

```
[1] 33.9
```

In [25]:
```
# So let's split into 4 groups, 0-10, 10-20, 20-30, 30-40
tooth.length <- cut(ToothGrowth[,"len"],breaks=10*(0:4),labels=c("Short","Normal",'
print(tooth.length)
```

```
 [1] Short     Normal    Short     Short     Short     Short     Normal
 [8] Normal    Short     Short     Normal    Normal    Normal    Normal
[15] Long      Normal    Normal    Normal    Normal    Normal    Long
[22] Normal    Superlong Long      Long      Superlong Long      Long
[29] Long      Long      Normal    Long      Normal    Short     Normal
[36] Short     Short     Short     Normal    Short     Normal    Long
[43] Long      Long      Normal    Long      Long      Long      Normal
[50] Long      Long      Long      Long      Long      Long      Superlong
[57] Long      Long      Long      Long
Levels: Short Normal Long Superlong
```

1.10.3 Frequency tables

table(f1,f2)

The function `table` allows frequency tables to be calculated from equal-length factors `f1` and `f2`.

We can now define two factors, e.g. a factor `method`, reflecting the Vitamin C supplement type, and a factor `teeth`, classifying the teeth length:

In [26]:
```
method <- factor(ToothGrowth[,"supp"])
```

Finally we can generate a frequency table showing the distribution of teeth lengths depending on the Vitamin C supplement method:

In [27]:
```
table(suppl_method=method,length=tooth.length)
```

```
            length
suppl_method Short Normal Long Superlong
          OJ     5      7   17         1
          VC     7     13    8         2
```

1.11 Data input/output, files

1.11.1 Reading data

The tools for data handling and editing available in R are fairly limited and thus it is a good idea to prepare them beforehand in a dedicated application, such as a spreadsheet or database program. Several packages are available on CRAN to help communicate with databases using the SQL language or the ODBC (Open Database Connectivity Application Programming Interface, API).

Moreover there is a package interfacing to Windows applications (including MS Excel) via the DCOM interface. If you require any of these sophisticated tools, see the "R Data Import/Export" pdf file in documentation.

In many situations it will be sufficient to import plain text files. The most powerful of the functions available for this purpose is:

read.table(filename, header = FALSE, sep = "", na.strings = "NA", check.names = TRUE, quote = "'", dec = ".", fill = !blank.lines.skip)

This function imports a data file specified by `filename`, in which the individual items are separated by separator `sep`. The common separators are `","` – comma, `"\t"` – tab, and `"\n"` – new line. The parameter `dec` specifies a character interpreted as a decimal point. Note that the result is a data frame (and not a matrix), even if the file contains only numerical values.

If matrix operations are to be employed, the data object must be thus explicitly converted.

Unless the full path is specified, the file is searched in the current working directory. The directory can be queried with the `getwd()` command and set with the `setwd(dir)` function or the menu option `File|Change dir...`

A parameter worth resetting to `FALSE` is `check.names` as it determines whether the row and column names are to be syntactically checked to be valid R names. When `TRUE`, R will replace e.g. accented characters and slashes (`/`) with dots. There is a useful convention; if the first row in the data file has one item less than the following ones, it is interpreted as column names and every first item in subsequent rows as a respective row name. The file might look as follows:

SiO2 → TiO2 → Al2O3 → Fe2O3 → FeO

Li1 → 51.73 → 1.48 → 16.01 → 1.03 → 7.06

Li2 → 51.88 → 1.48 → 15.93 → 0.99 → 6.85 ...

In order to read a text file in which the lengths of rows are all the same, but column names are present, one can employ `header = TRUE, row.names = 1`.

Parameter `na.strings` specifies text strings to be interpreted as missing values, e.g., `na.strings=c("b.d.","-","NA")`. It is fairly common for a file exported from a spreadsheet such as MS Excel to have all trailing empty fields and their separators omitted. To read such files set `fill = TRUE` or simply copy and paste the data from spreadsheet to your text editor directly using the Windows clipboard.

In [28]:
```
sazava  <-  read.table("data/sazava.data",sep="\t")
head(sazava)
```

A data.frame: 6 × 49

	Intrusion	Locality	Petrology	Outcrop	Symbol	Colour	SiO2	TiO2	Al2O3	FeO	..
	<chr>	<chr>	<chr>	<chr>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	..
Sa-1	Sazava	Mrac	bi-amph quartz diorite	working quarry	10	4	59.98	0.63	16.42	5.46	..
Sa-2	Sazava	Mrac	bi-amph quartz diorite	working quarry	10	4	55.17	0.71	17.00	5.26	..
Sa-3	Sazava	Mrac	bi-amph quartz diorite	working quarry	10	4	55.09	0.75	17.59	5.81	..
Sa-4	Sazava	Mrac	bi-amph quartz diorite	working quarry	10	4	50.72	0.83	17.57	7.65	..
Sa-7	Sazava	Teletín	bi-amph tonalite	disused quarry	10	1	57.73	0.95	18.82	5.43	..
SaD-1	basic	Teletín	bi-amph quartz diorite	disused quarry	8	1	52.90	1.35	18.23	7.24	..

readClipboard()

In its simplest form, this function reads the text from the Windows clipboard.

1.11.2 Sample data sets

R and its packages contain numerous sample datasets that can be attached to the current session using the function `data(…)`. For instance:

In [29]:
```
data(islands)
```

Then documentation is available using the help command `?islands`.

1.11.3 Saving data

write.table(x, file = "", append = FALSE, sep = " ", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE)

This function writes an object `x` (a matrix, a data frame, or an object that can be converted to such) to the specified `file`, separating the individual items by `sep`. As for `read.table`, one can specify the strings representing the missing values and the decimal point. Moreover, there are logical parameters determining whether row and/or column names are to be stored (`row.names, col.names`) and whether to append the data without erasing those possibly already present.

writeClipboard(str)

Writes the text specified by the character vector `str` to the Windows clipboard.