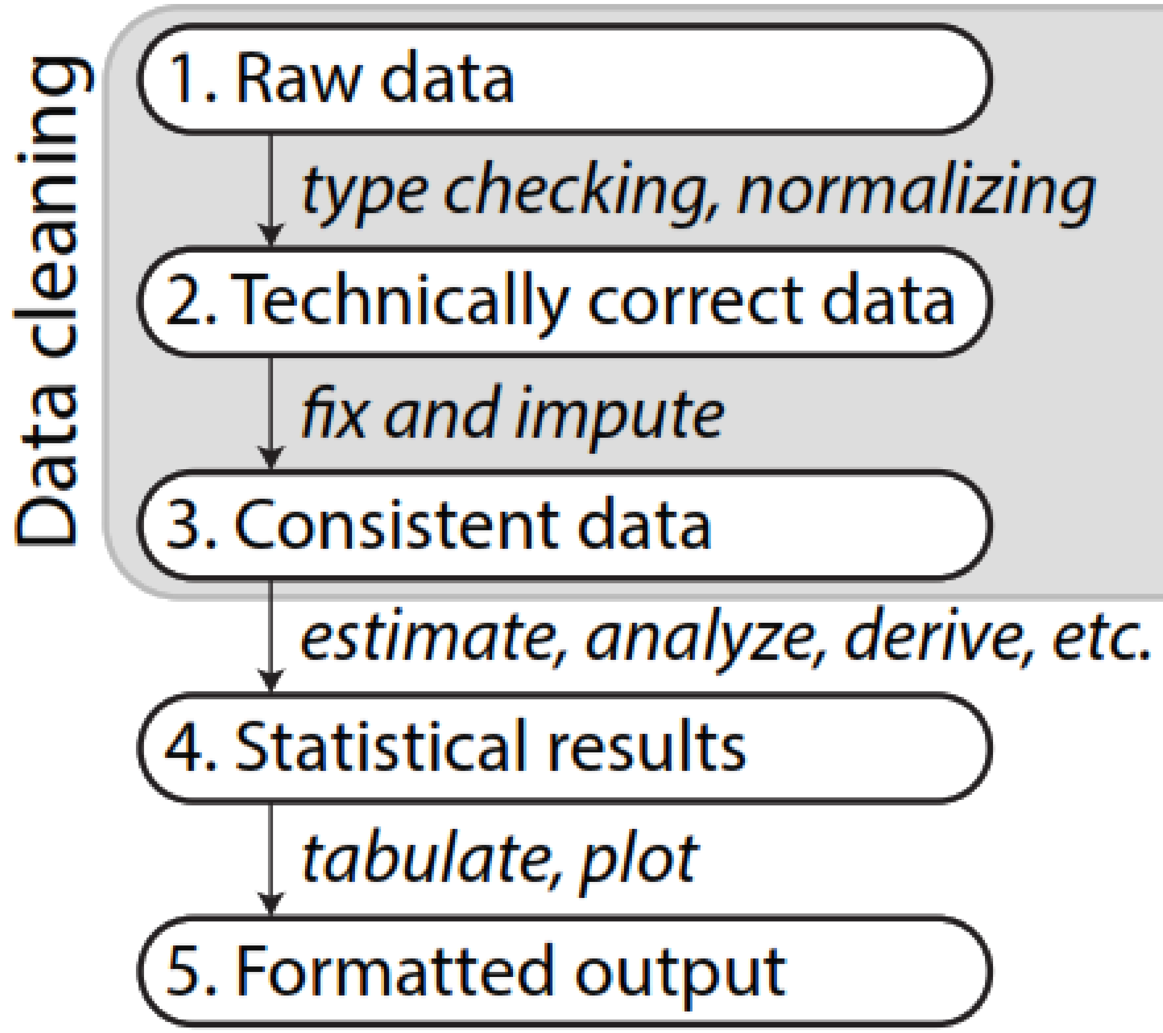
Data Cleaning in R

5 Steps of Data Analysis



Raw Data

- May lack headers
- Contain wrong data types (e.g., numbers stored as strings)
- Wrong category labels, unknown or unexpected
- Character encoding and so on.

Technically correct data

- For example, an age variable may be reported negative, an under-aged person may be registered to possess a driver's license, or data may simply be missing.
- Such inconsistencies obviously depend on the subject matter that the data pertains to, and they should be ironed out before valid statistical inference from such data can be produced.

Consistent data

- The data is ready for statistical inference. It is the data that most statistical theories use as a starting point.
- Ideally, such theories can still be applied without taking previous data cleaning steps into account.
- In practice however, data cleaning methods like imputation of missing values will influence statistical results and so must be accounted for in the following analyses or interpretation thereof.

Steps of Data Cleaning

- Loading data into R from databases, spreadsheets, or other formats
- Shaping data into a form appropriate for analysis
- Checking variable types: Are the variables of the type that you expect?
- Managing bad values: What do you do with NaNs or values that are out of range or invalid (Outliers)
- Anticipating future novel categorical values (values that were not present in training data)
- Re-encoding categorical values with too many levels: How do you use such variables in an analysis efficiently?

RAW Data formatting

- Header at time of reading Data
- Type Conversion
- Recoding
- Converting Dates
- String Normalization

RAW to Technically correct data

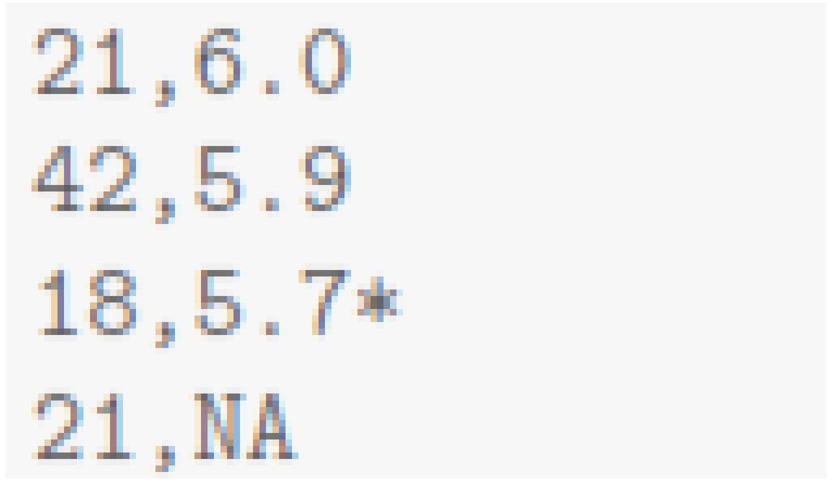
- Limiting ourselves to “rectangular" data sets read from a text-based format, technically correct data in R
  - is stored in a data.frame with suitable columns names, and
  - each column of the data.frame is of the R type that adequately represents the value domain.
- The second demand implies that numeric data should be stored as numeric or integer, textual data should be stored as character and categorical data should be stored as a factor or ordered vector, with the appropriate levels.

Header at time of Reading Data

- read.table() and similar functions below will read a text le and return a data.frame.
- The other read-functions below all eventually use read.table() with some fixed parameters and possibly after some preprocessing.
- Specially
  - read.csv() for comma separated values with period as decimal separator.
  - read.csv2() for semicolon separated values with comma as decimal separator.
  - read.delim() tab-delimited les with period as decimal separator.
  - read.delim2() tab-delimited les with comma as decimal separator.
  - read.fwf() data with a predetermined number of bytes per column.

Header Issue: Example

- Except for read.table() and read.fwf(), each of the above functions assumes by default that the first line in the text file contains column headers. For example type the following data in a text file and save it to desktop as unnamed.txt:



```
21,6.0
42,5.9
18,5.7*
21,NA
```

Header Issue: Example

```
fn.data<-"C:/Users/d.singh/Desktop/unnamed.txt"
>person<-read.csv(fn.data)
>person
## X21 X6.0
## 1 42 5.9
## 2 18 5.7*
## 3 21 <NA>
# instead, use header = FALSE and specify the column names
>person<-read.csv(file= fn.data ,header=FALSE ,col.names=c("age","height") )
>person
## age height
## 1 21 6.0
## 2 42 5.9
## 3 18 5.7*
## 4 21 <NA>
```

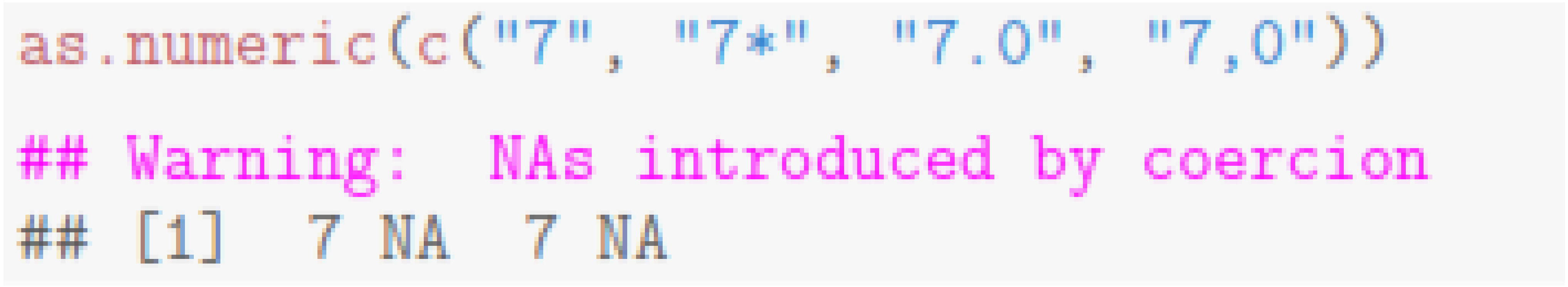
- Best practice: A freshly read data.frame should always be inspected with functions like head(), str(), and summary().

Type conversion

- as.numeric
- as.integer
- as.character
- as.logical
- as.factor
- as.ordered

Type Conversion" Example

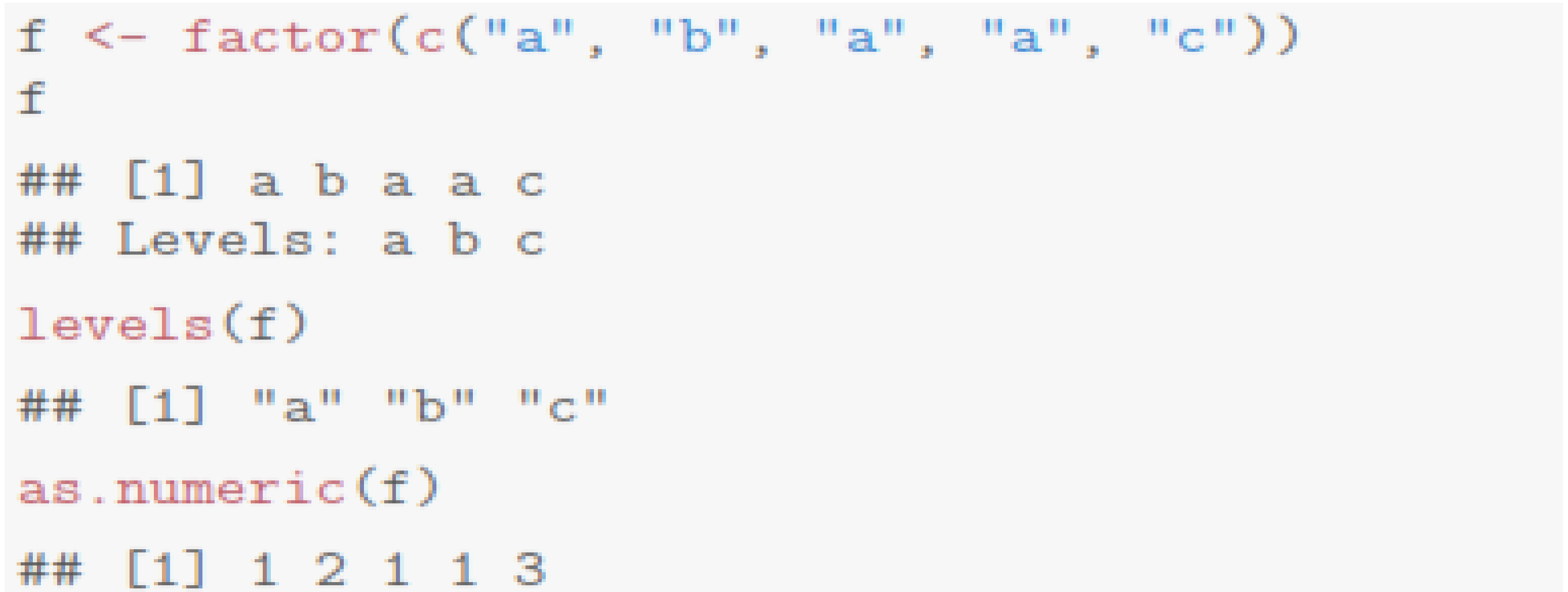
- Each of these functions takes an R object and tries to convert it to the class specified behind the "as.".
- By default, values that cannot be converted to the specied type will be converted to a NA value while a warning is issued.



```
as.numeric(c("7", "7*", "7.0", "7,0"))

## Warning:  NAs introduced by coercion
## [1]  7 NA  7 NA
```

Recoding factors

- In R, the value of categorical variables is stored in factor variables.
- A factor is an integer vector endowed with a table specifying what integer value corresponds to what level.
- The values in this translation table can be requested with the levels function.



```r
f <- factor(c("a", "b", "a", "a", "c"))
f
```

```
## [1] a b a a c
## Levels: a b c
```

```r
levels(f)
```

```
## [1] "a" "b" "c"
```

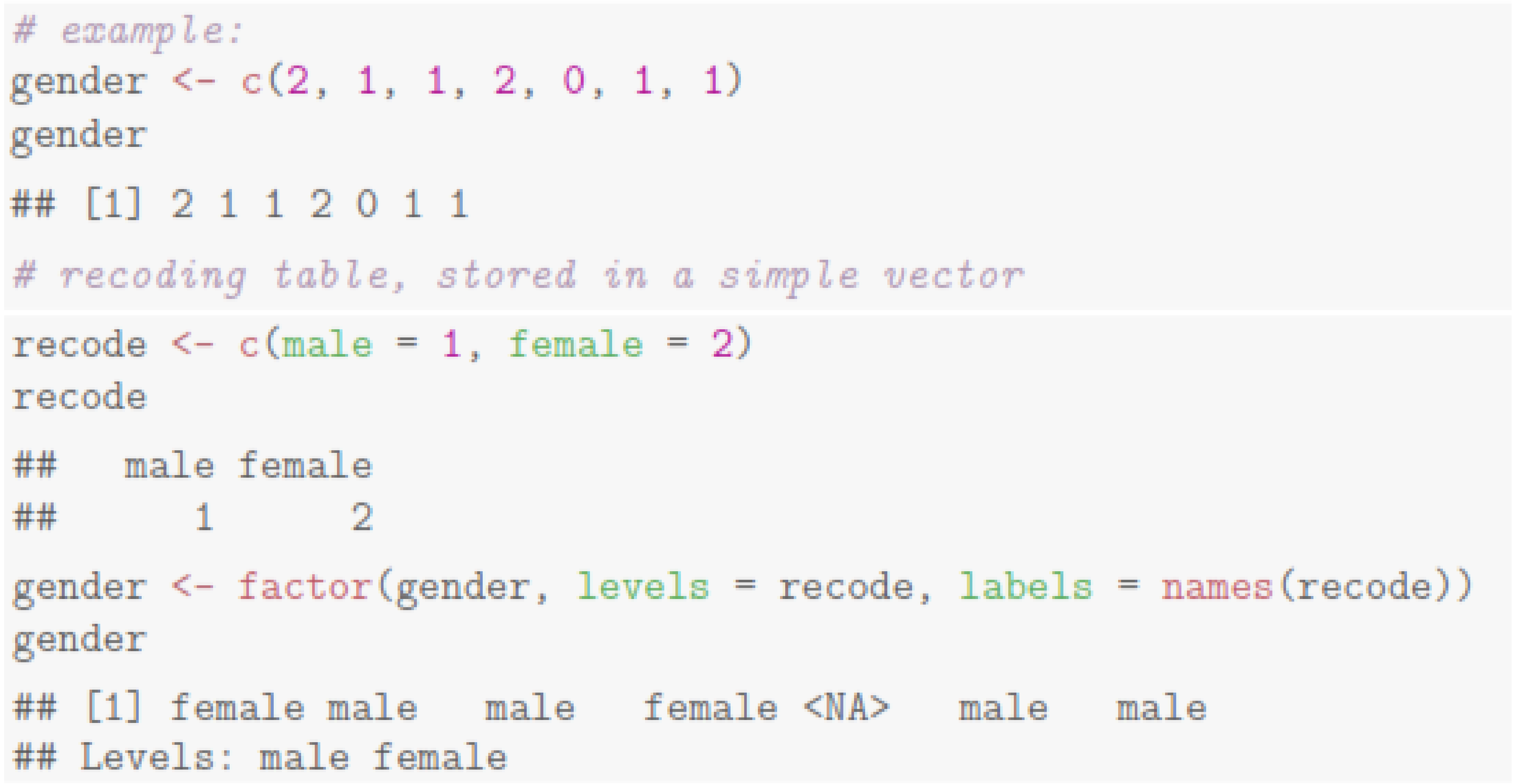
```r
as.numeric(f)
```

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

Recoding factors: Example

- For example, suppose we read in a vector where 1 stands for male, 2 stands
- for female and 0 stands for unknown.

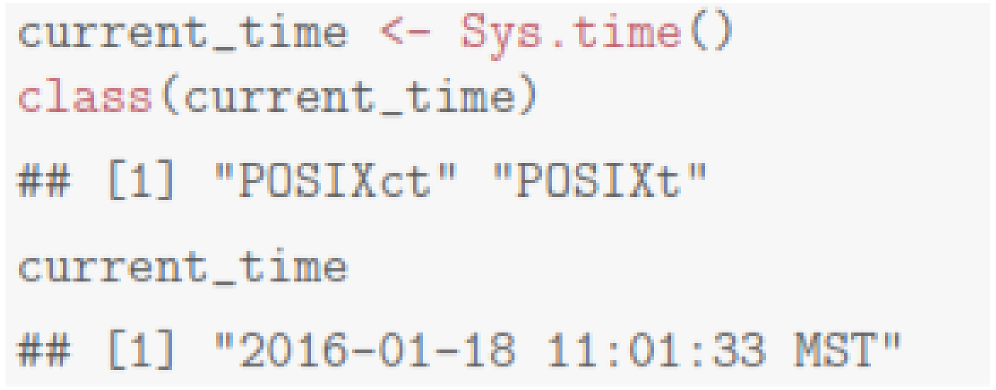
Recoding factors: Example



```
# example:
gender <- c(2, 1, 1, 2, 0, 1, 1)
gender

## [1] 2 1 1 2 0 1 1

# recoding table, stored in a simple vector
recode <- c(male = 1, female = 2)
recode

##   male female
##      1      2

gender <- factor(gender, levels = recode, labels = names(recode))
gender

## [1] female male   male   female <NA>   male   male
## Levels: male female
```

Converting dates

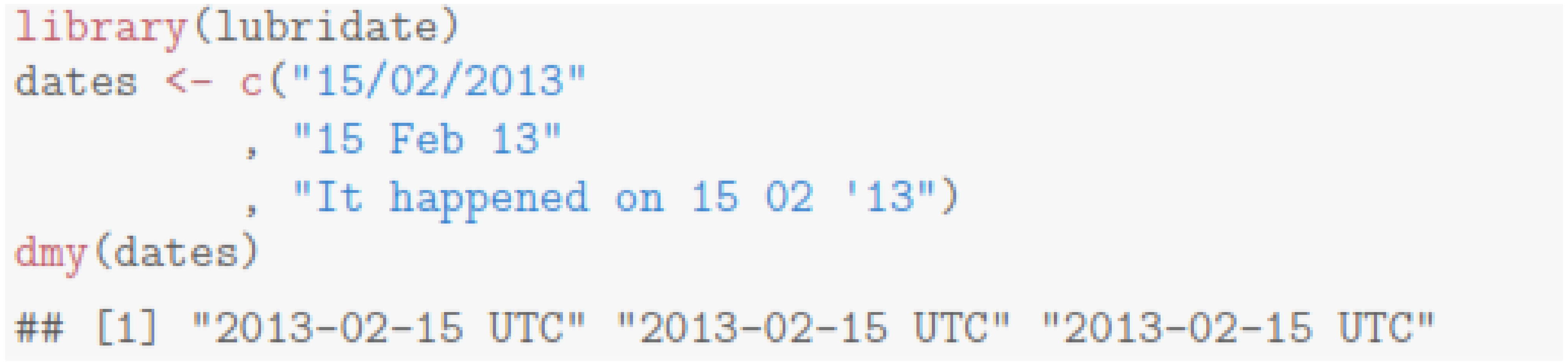
- The base R installation has three types of objects to store a time instance:
  - Date, POSIXlt, and POSIXct.
  - The Date object can only be used to store dates,
  - Other two store date and/or time.
- Here, we focus on converting text to POSIXct objects since this is the most portable way to store such information.



```
current_time <- Sys.time()
class(current_time)
## [1] "POSIXct" "POSIXt"
current_time
## [1] "2016-01-18 11:01:33 MST"
```

Converting dates: Example

- The lubridate package contains a number of functions facilitating the conversion of text to POSIXct dates. As an example, consider the following code.

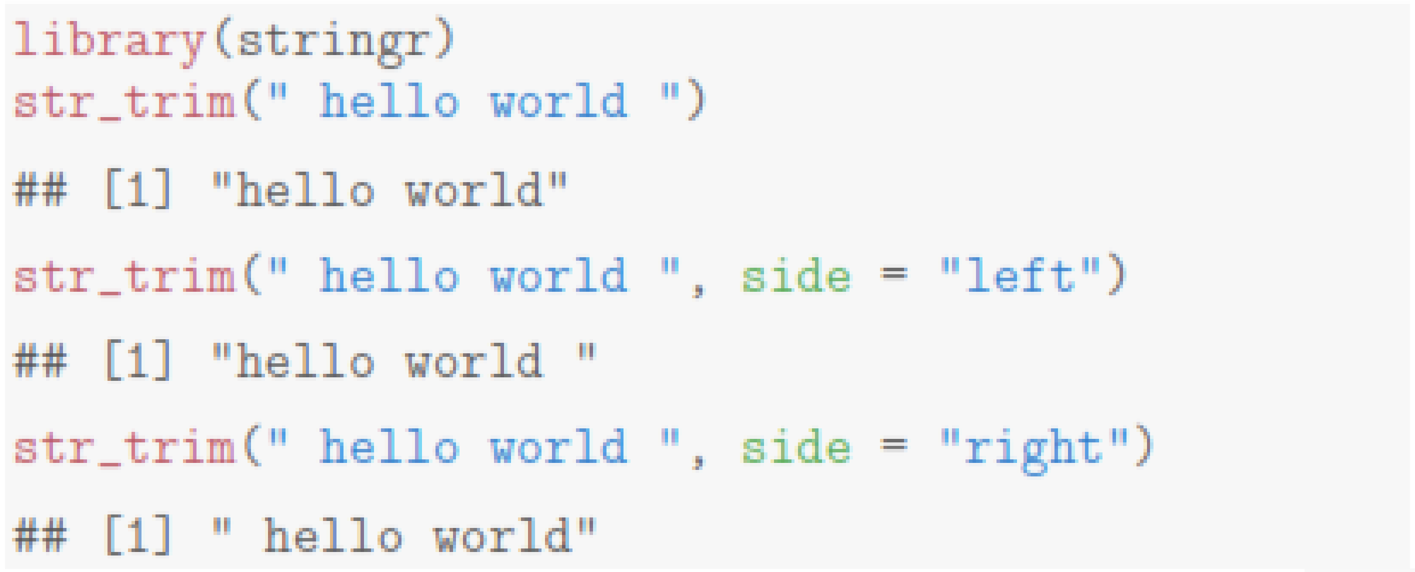


```
library(lubridate)
dates <- c("15/02/2013"
         , "15 Feb 13"
         , "It happened on 15 02 '13")
dmy(dates)

## [1] "2013-02-15 UTC" "2013-02-15 UTC" "2013-02-15 UTC"
```

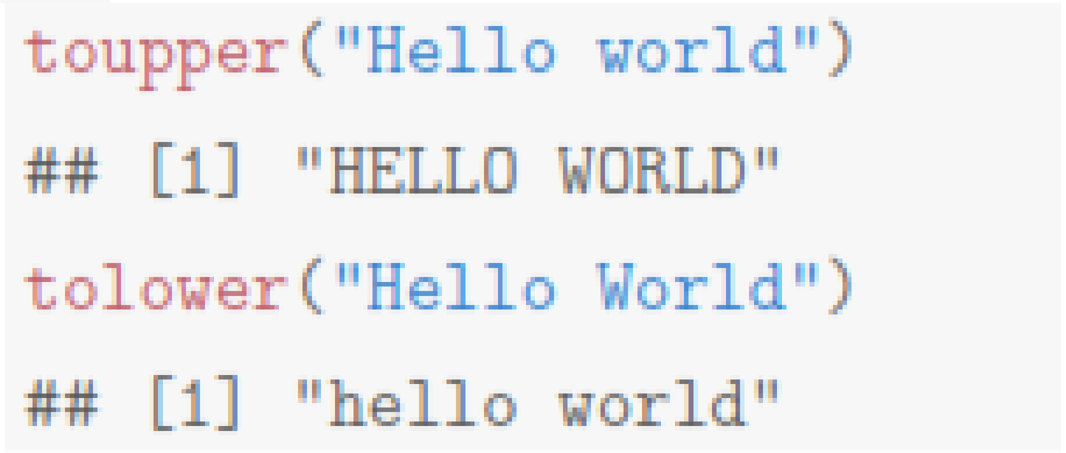
String normalization

- String normalization techniques are aimed at transforming a variety of strings to a smaller set of string values which are more easily processed.
- The stringr package offers a number of functions that make some some string manipulation tasks a lot easier than they would be with R's base functions.
- For example, extra white spaces at the beginning or end of a string can be removed using str_trim().

Str_trim & toupper/tolower



```r
library(stringr)
str_trim(" hello world ")

## [1] "hello world"

str_trim(" hello world ", side = "left")

## [1] "hello world "

str_trim(" hello world ", side = "right")

## [1] " hello world"
```



```r
toupper("Hello world")

## [1] "HELLO WORLD"

tolower("Hello World")

## [1] "hello world"
```

Technically correct data to consistent data

- Consistent data are technically correct data that are t for statistical analysis.
- They are data in which missing values, special values, (obvious) errors and outliers are either removed, corrected, or imputed.
- The data are consistent with constraints based on real-world knowledge about the subject that the data describe.

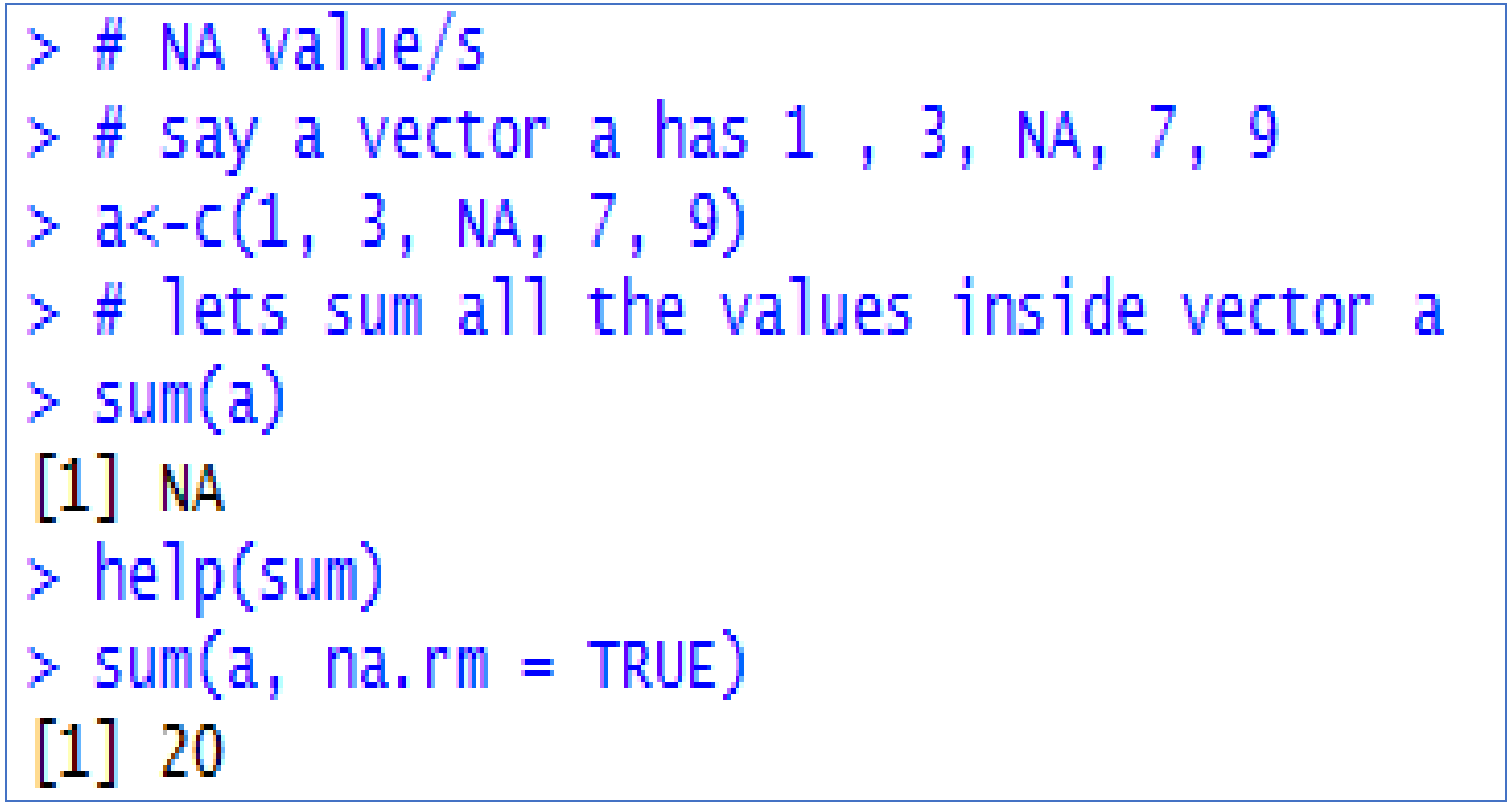
Main Issues with consistency of Data

- Special values and value-checking functions
  - NA
  - NULL
  - Infinity
  - NaN
- Outliers

NA: Not Available

- NA Stands for “not available".
- NA is a placeholder for a missing value.
- All basic operations in R handle NA without crashing and mostly return NA as an answer whenever one of the input arguments is NA.
- If you understand NA, you should be able to predict the result of the following R statements.

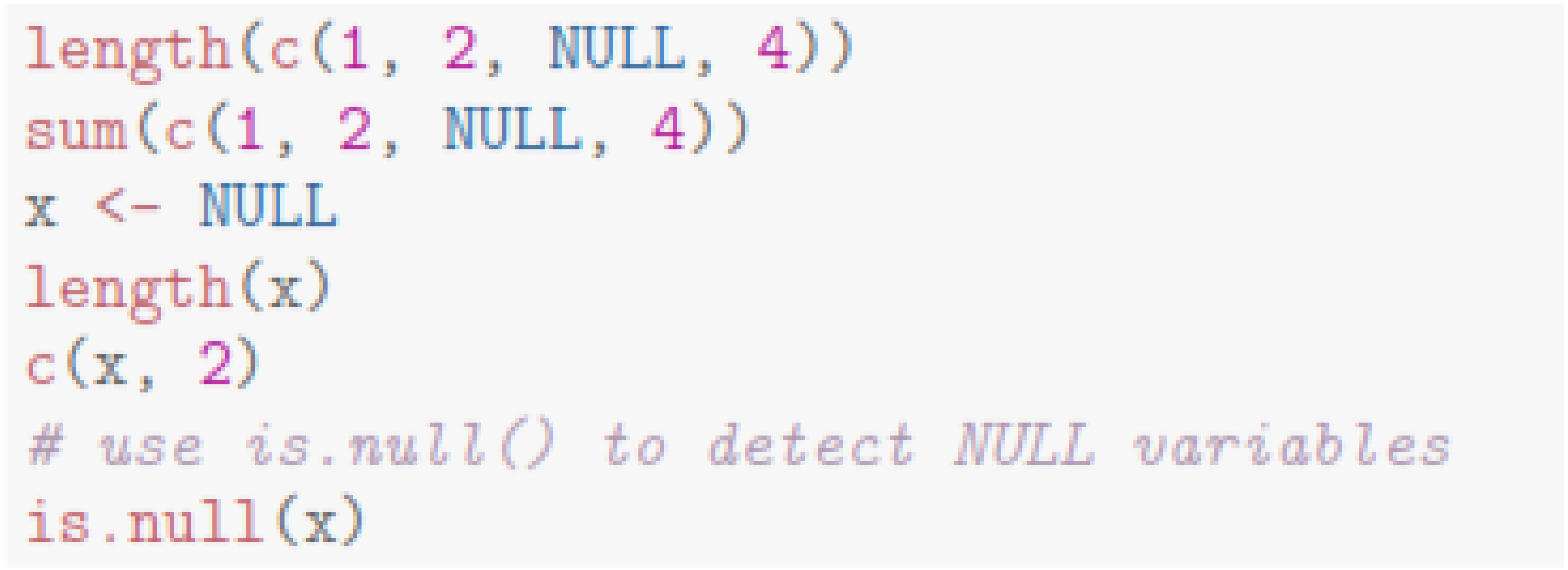
NA value



```
> # NA value/s
> # say a vector a has 1 , 3, NA, 7, 9
> a<-c(1, 3, NA, 7, 9)
> # lets sum all the values inside vector a
> sum(a)
[1] NA
> help(sum)
> sum(a, na.rm = TRUE)
[1] 20
```

NULL

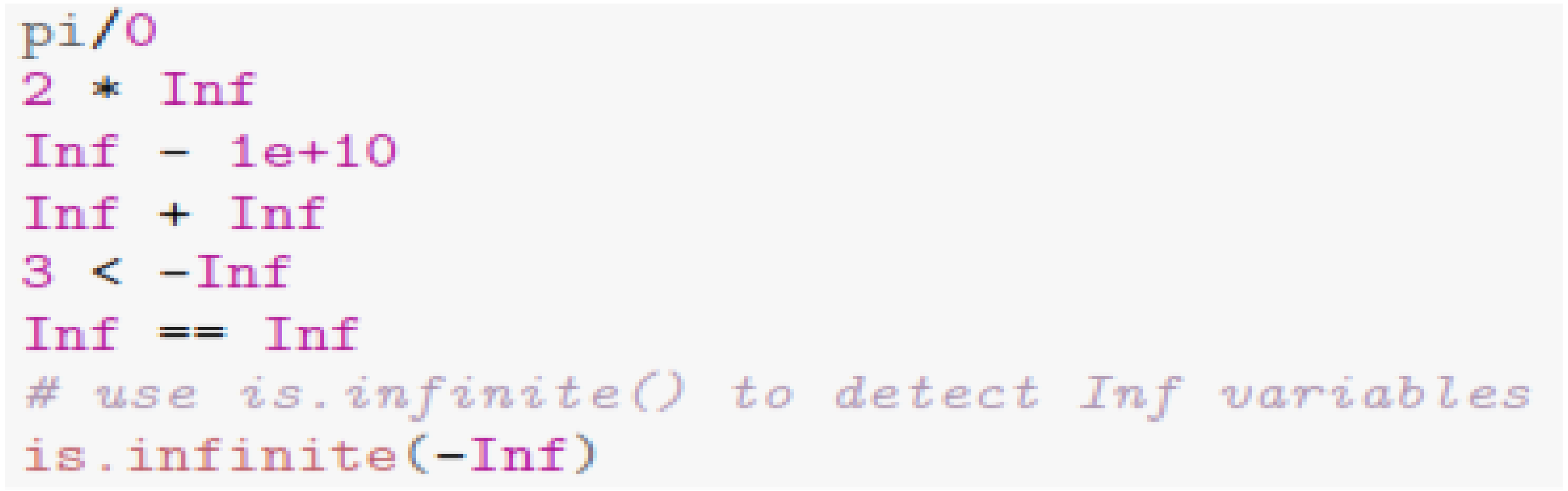
- NULL Think of NULL as the empty set from mathematics; it has no class (its class is NULL) and has length 0 so it does not take up any space in a vector.



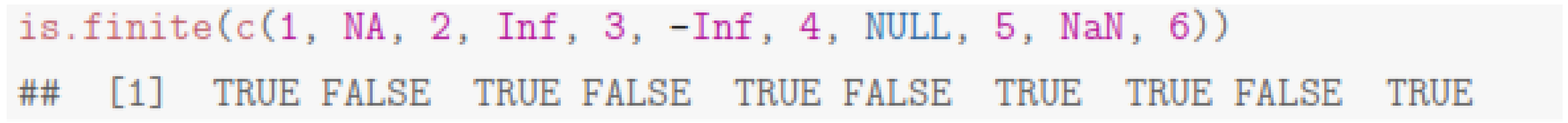
```r
length(c(1, 2, NULL, 4))
sum(c(1, 2, NULL, 4))
x <- NULL
length(x)
c(x, 2)
# use is.null() to detect NULL variables
is.null(x)
```

Infinity

- Inf Stands for “infity" and only applies to vectors of class numeric (not integer). Technically, Inf is a valid numeric that results from calculations like division of a number by zero.



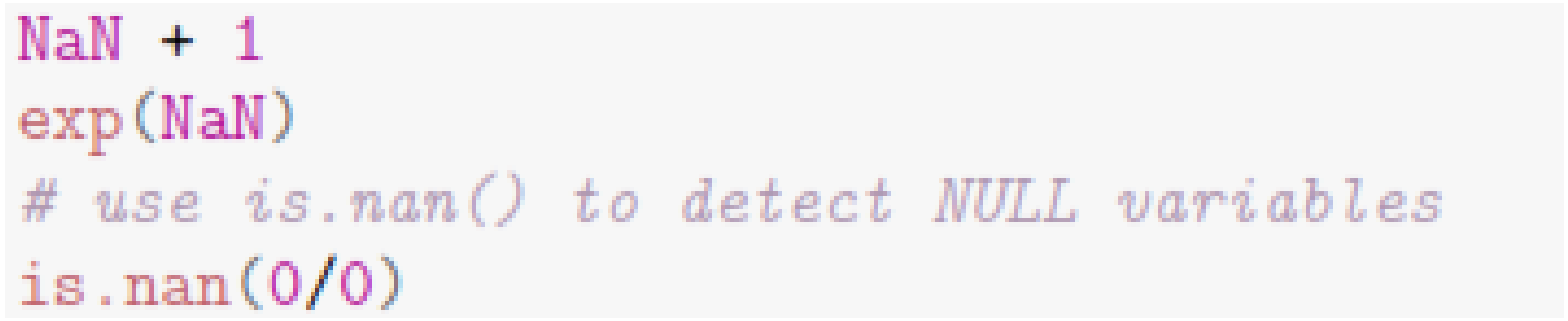
```
pi/0
2 * Inf
Inf - 1e+10
Inf + Inf
3 < -Inf
Inf == Inf
# use is.infinite() to detect Inf variables
is.infinite(-Inf)
```



```
is.finite(c(1, NA, 2, Inf, 3, -Inf, 4, NULL, 5, NaN, 6))
## [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
```

NaN: Not a Number

- NaN Stands for “not a number". This is generally the result of a calculation of which the result is unknown, but it is surely not a number. In particular operations like 0=0, Inf-Inf and Inf=Inf result in NaN.
- Technically, NaN is of class numeric, which may seem odd since it is used to indicate that something is not numeric. Computations involving numbers and NaN always result in NaN.



```
NaN + 1
exp(NaN)
# use is.nan() to detect NULL variables
is.nan(0/0)
```

Outliers

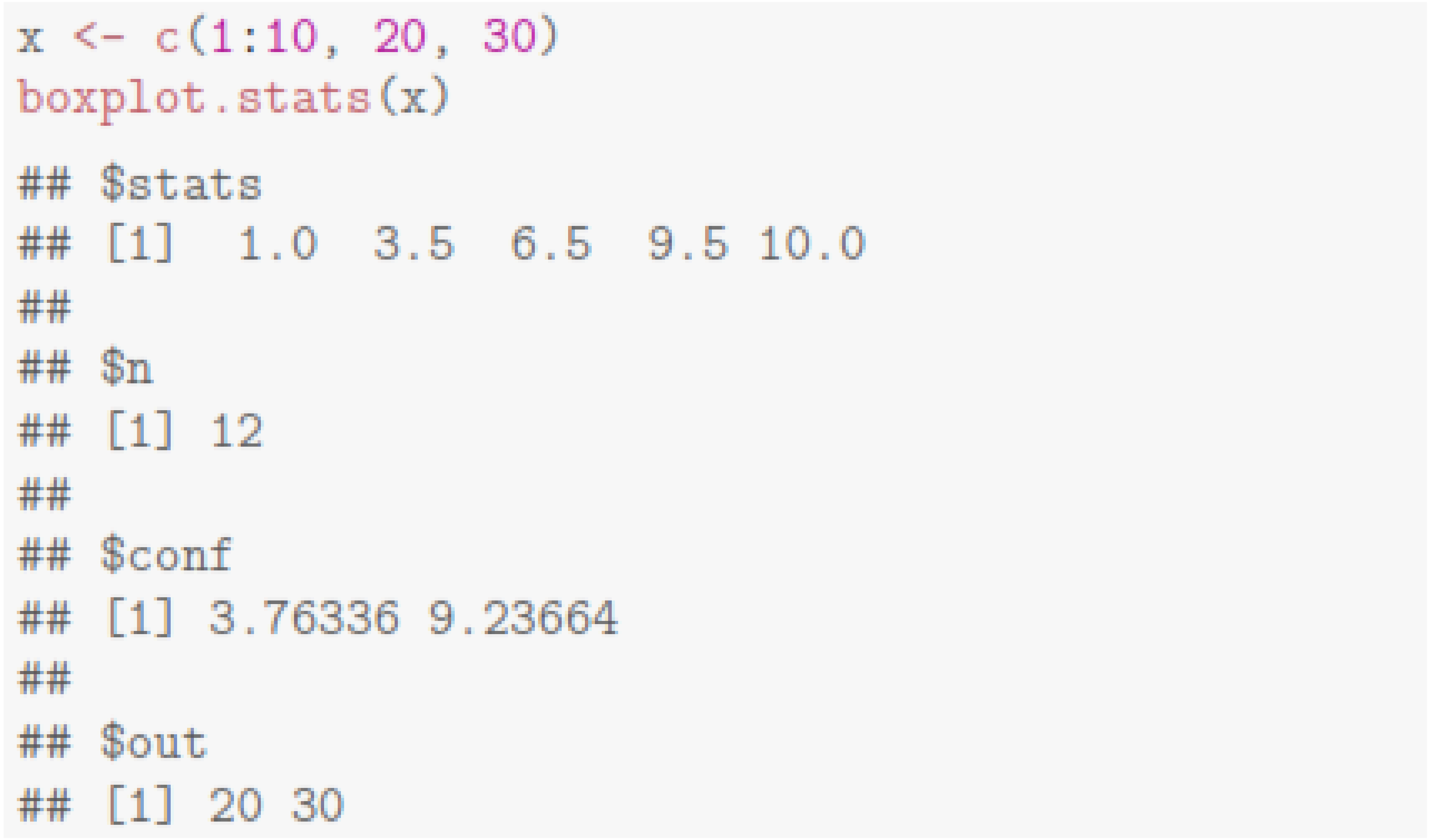
- A general definition by Barnett and Lewis defines:

"An outlier in a data set as an observation (or set of observations) which appear to be inconsistent with that set of data."

Some Details

- Outliers do not equal errors.
- They should be detected, but not necessarily removed. Their inclusion in the analysis is a statistical decision.
- For more or less unimodal and symmetrically distributed data, Tukey's box and-whisker method for outlier detection is often appropriate.
- In this method, an observation is an outlier when it is larger than the so-called “whiskers" of the set of observations. The upper whisker is computed by adding 1.5 times the interquartile range to the third quartile and rounding to the nearest lower observation. The lower whisker is computed likewise.

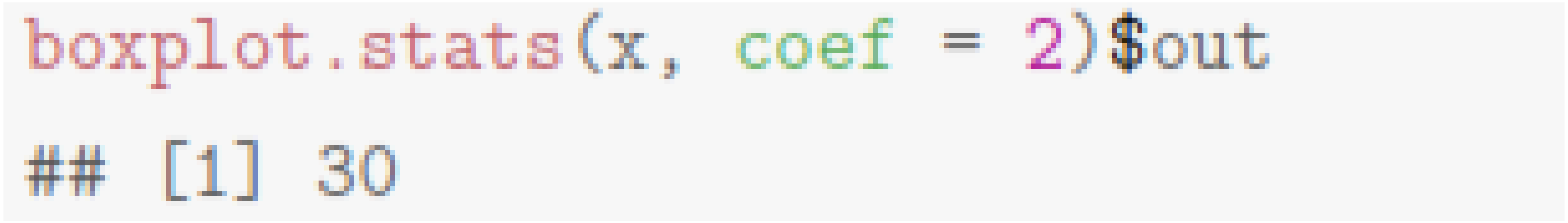
Boxplot.stats



```
x <- c(1:10, 20, 30)
boxplot.stats(x)

## $stats
## [1]  1.0  3.5  6.5  9.5 10.0
##
## $n
## [1] 12
##
## $conf
## [1] 3.76336 9.23664
##
## $out
## [1] 20 30
```

Boxplot.stats: coef

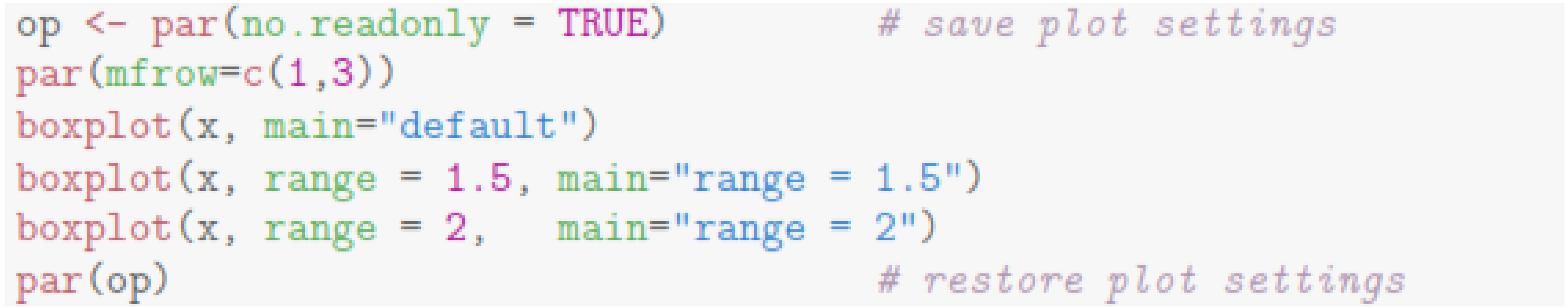
- The factor 1.5 used to compute the whisker is to an extent arbitrary and it can be altered by setting the coef option of boxplot.stats(). A higher coefficient means a higher outlier detection limit (so for the same dataset, generally less upper or lower outliers will be detected).



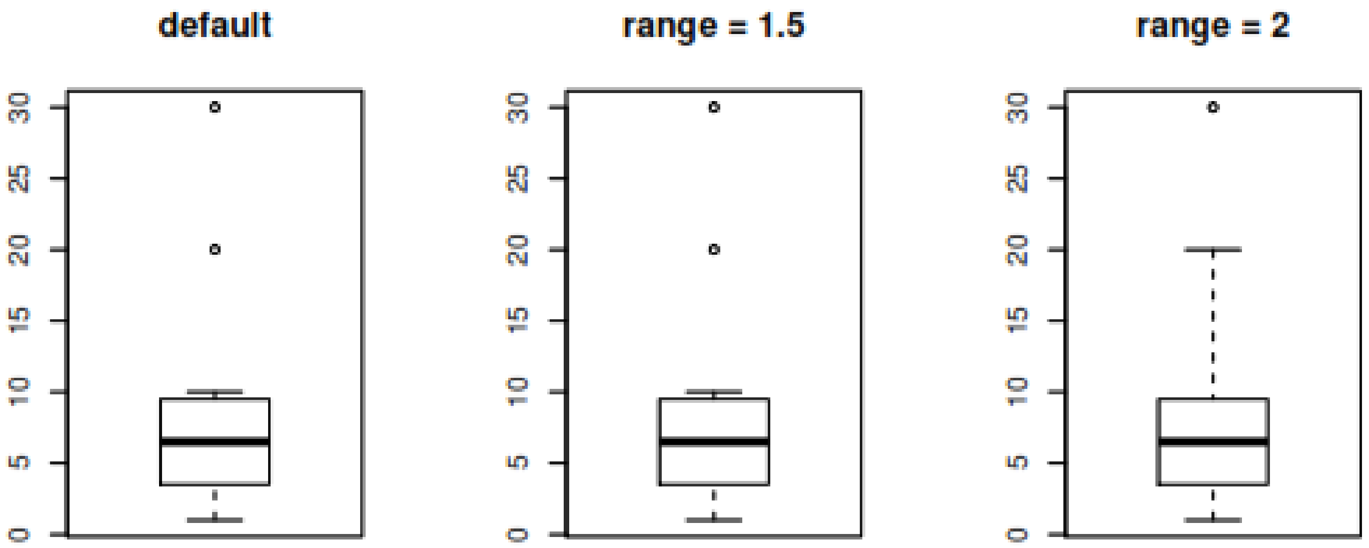
```
boxplot.stats(x, coef = 2)$out
## [1] 30
```

Box-and-whisker plot

- The box-and-whisker method can be visualized with the box-and-whisker plot, where the box indicates the interquartile range and the median, the whiskers are represented at the ends of the box-and-whisker plots and outliers are indicated as separate points above or below the whiskers.



```r
op <- par(no.readonly = TRUE)          # save plot settings
par(mfrow=c(1,3))
boxplot(x, main="default")
boxplot(x, range = 1.5, main="range = 1.5")
boxplot(x, range = 2,   main="range = 2")
par(op)                                # restore plot settings
```


default
range = 1.5
range = 2
0
5
10
15
20
25
30

Thanks