R Course for the NSOs in the Arab countries
Part II: R Basics

Valentin Todorov[1]

[1]United Nations Industrial Development Organization, Vienna

18-20 May 2015

Outline

1. R basic operations
2. Data types, modes and classes
3. R vectors
4. Statistical functions
5. Data structures
6. Missing data
7. Summary

Operator precedence in R

- The precedence of the basic oprators of R is as follows (highest first):
  - ^ exponent
  - - + unary minus and plus
  - : sequence operator (e.g. 1:10)
  - %/% %% integer division, remainder (to be discused later0
  - * / multiplication, division
  - + - addition, subtraction
- Operations with higher precedence take place before those with lower precidence.
- Operations with equal precedence will be evaluated from left to right (except for exponentiation, which takes place right-to-left).

R expressions

```
> ## Evaluation order (exponentiation)
> 2^3^2
[1] 512
> (2^3)^2
[1] 64
> ## Simple expressions
> 5+2
[1] 7
> 5^2
[1] 25
> 5 * 5
[1] 25
> 2 + 3 * 3
[1] 11
```

R functions

```
> rnorm(10)

 [1]  0.7244630 -1.0810779 -1.2142282 -1.3000762  0.1717139 -0.5960454
 [7] -0.6713543 -1.6860146 -0.5387339 -0.5362302

> 1:10

 [1]  1  2  3  4  5  6  7  8  9 10

> mean(1:10)

[1] 5.5

> sqrt(25)

[1] 5

> log(10)

[1] 2.302585

>
```

Variables and assignment

- R variables do not need to be declared in advance
- Can hold any of the R data types (to come later in this lecture) or an object or a function.
- Assignment: giving the value a name or
- Assignment: storing a value into a variable.
- Thus: a variable is defined as a value-name pair.
- The following three expressions are equivalent:

```
> a <- 10
> a=10
> 10 -> a
> ## Let us see what is in a?
> a
[1] 10
```

Variables and assignment

```
> a <- b <- 10
> a
[1] 10
> b
[1] 10
```

- Variable name can contain any alphanumeric character, period (.) and underscore (_)
- Variable name cannot start with a number or an underscore
- Find what variables do we have (function `ls()`):

```
> ls()
[1] "a"          "b"          "filename"
```

Variable names

- Variable names are case sensitive:
  - `myvar` is different from `MyVar`
- Like C, C++, Java and
- unlike VB, SQL

Variables and assignment

Removing variables

```
> a

[1] 10

> rm(a)

> a

Error: object 'a' not found
```

Data types in R

- All R objects have a type or storage mode
- The possible primitive types of data are:
  - numeric
  - character
  - logical
  - complex
- integer is a subclass of numeric
- double is a subclass of numeric
- Use function `mode()` to display an object's mode (e.g. numeric, character, logical)
- Use function `typeof()` to display an object's type (e.g. numeric, integer, double, logical)
- other modes are:
  - list
  - function

Numeric data

- R numeric data
  - Similar to float and double in other languages
  - Can handle integers and decimals, both negative and positive and zero.
  - Numeric is the default data time - if you store a variable, it is automatically numeric:

```
> x <- 10
> x
[1] 10
> is.numeric(x)
[1] TRUE
> mode(x)
[1] "numeric"
> typeof(x)
[1] "double"
```

Numeric data

- Another (not that often used) type is integer

```
> x <- 10L
> is.numeric(x)
[1] TRUE
> is.integer(x)
[1] TRUE
> mode(x)
[1] "numeric"
> typeof(x)
[1] "integer"
```

Character data

- R has two primary ways of handling character data: character and factor
- Although they seem similar, they are treated completely differently

```
> country <- "Oman"
> country
[1] "Oman"
> is.character(country)
[1] TRUE
> ct <- factor("Oman")
> ct
[1] Oman
Levels: Oman
```

- To find the length of a characters string we use `nchar()`:

```
> nchar(country)
[1] 4
```

- More about factors later.

Dates and time

- We skip this for now

Logicals

R vectors

- R vectors
  - In R the 'base' type is a vector, not a scalar.
  - A vector is an indexed set of values that are all of the same type.
  - The type of the entries determines the class of the vector.
  - A vector cannot contain elements of different modes.
  - The most common way to create a vector is the function c().

```
> ##  This is a vector with five integer elements
> ##  containing the elements 1,3,5,7,9 in this order:
>
> v <- c(1,3,5,7,9)
> v

[1] 1 3 5 7 9

> ## This is a character vector:
> cv <- c("Oman", "Qatar", "Bahrain")
> cv

[1] "Oman"    "Qatar"   "Bahrain"
```

Creating vectors

- We can create a vector with the `c()` function:

```
> v <- c(1,2,3,4,5,6,7,8,9,10)
```

- The values in the vector can be viewed by simply typing its name:

```
> v
 [1]  1  2  3  4  5  6  7  8  9 10
```

- Another way to create a vector is using the `sequence` operator:

```
> w <- 1:10
> w
 [1]  1  2  3  4  5  6  7  8  9 10
> x <- 3:-3
> x
[1]  3  2  1  0 -1 -2 -3
```

Combining vectors

- The function `c()` can be used also to combine vectors:

```
> v <- c(1,2,3,4)
> c(v, 100)
```

```
[1]   1   2   3   4 100
```

```
> c(v,v)
```

```
[1] 1 2 3 4 1 2 3 4
```

- Actually the number 10 is stored in R also as a vector with length 1—there are no simple numbers (scalars) in R, everything is a vector.

Vector operations

- Since everything is a vector, we can do:

```
> x <- c(1,2,3,4)
> 2*x+1
[1] 3 5 7 9
> sqrt(x)
[1] 1.000000 1.414214 1.732051 2.000000
```

- Let us have two vectors of the same length

```
> x1 <- 1:5
> x2 <- -2:2
> x1
[1] 1 2 3 4 5
> x2
[1] -2 -1  0  1  2
```

Vector operations

```
> x1 + x2          # add the two vectors
[1] -1  1  3  5  7
> x1 * x2          # multiply them (element-wise)
[1] -2 -2  0  4 10
> length(x1)       # check the length of x1
[1] 5
> length(x1+x2)    # check the length of their sum
[1] 5
> length(c(x1, x2))   # check the length of the combined vect
[1] 10
```

Vector operations

Vector operations

- The advantage of vector operations in R is that no loops are necessary.

Vectors with different length

- Recycling (the shorter vector will be recycled)

```
> x1
[1] 1 2 3 4 5
> x1 + c(1,2,3)
[1] 2 4 6 5 7
```

More on the recycling rule: see Ross Ihaka

Comparing vectors

- Vectors can be compared using logical operators. The result is a vector of the same length, containing TRUE or FALSE values for each element.

```
> x1 <= 3
[1]  TRUE  TRUE  TRUE FALSE FALSE
> x1 <= x2
[1] FALSE FALSE FALSE FALSE FALSE
```

- The functions `all()` and `any()` can be used to check the result of the comparison

```
> any(x1 <= 3)
[1] TRUE
> all(x1 <= 3)
[1] FALSE
```

Accessing the elements of vectors

- To access individual elements of a vector use the square brackets operator [].
  - Retrieve the first element of a vector:

    ```
    > x1[1]
    ```

    ```
    [1] 1
    ```
  - Retrieve the last element of a vector (use the function `length()`:

    ```
    > x1[length(x1)]
    ```

    ```
    [1] 5
    ```
  - Retrieve several subsequent elements of a vector (use the sequence operator :):

    ```
    > x1[1:2]
    ```

    ```
    [1] 1 2
    ```
  - Retrieve several non-subsequent elements of a vector (use the function `c()`:

    ```
    > x1[c(1,4)]
    ```

    ```
    [1] 1 4
    ```

Vector names

- The elements of any vector can be named:

```
> x <- c(1,2,3,4)
> names(x) <- c("A", "B", "C", "D")
> x
```

```
A B C D
1 2 3 4
```

- The names of a vector can be extracted by the `names()` function:

```
> names(x)
```

```
[1] "A" "B" "C" "D"
```

Some statistical functions

- Functions make the code reusable
- Almost any step in R is calling a function

```
> x <- c(117.8, 119.9, 115.9, 121.9, 121.9, 118.9, 118.9,
+         123.9, 125.9, 113.9)
> mean(x)      # mean (average) of x
[1] 119.89
> sd(x)        # standard deviation
[1] 3.627197
> median(x)    # median
[1] 119.4
> var(x)       # variance
[1] 13.15656
> max(x)
[1] 125.9
> min(x)
[1] 113.9
```

Some statistical functions (cont.)

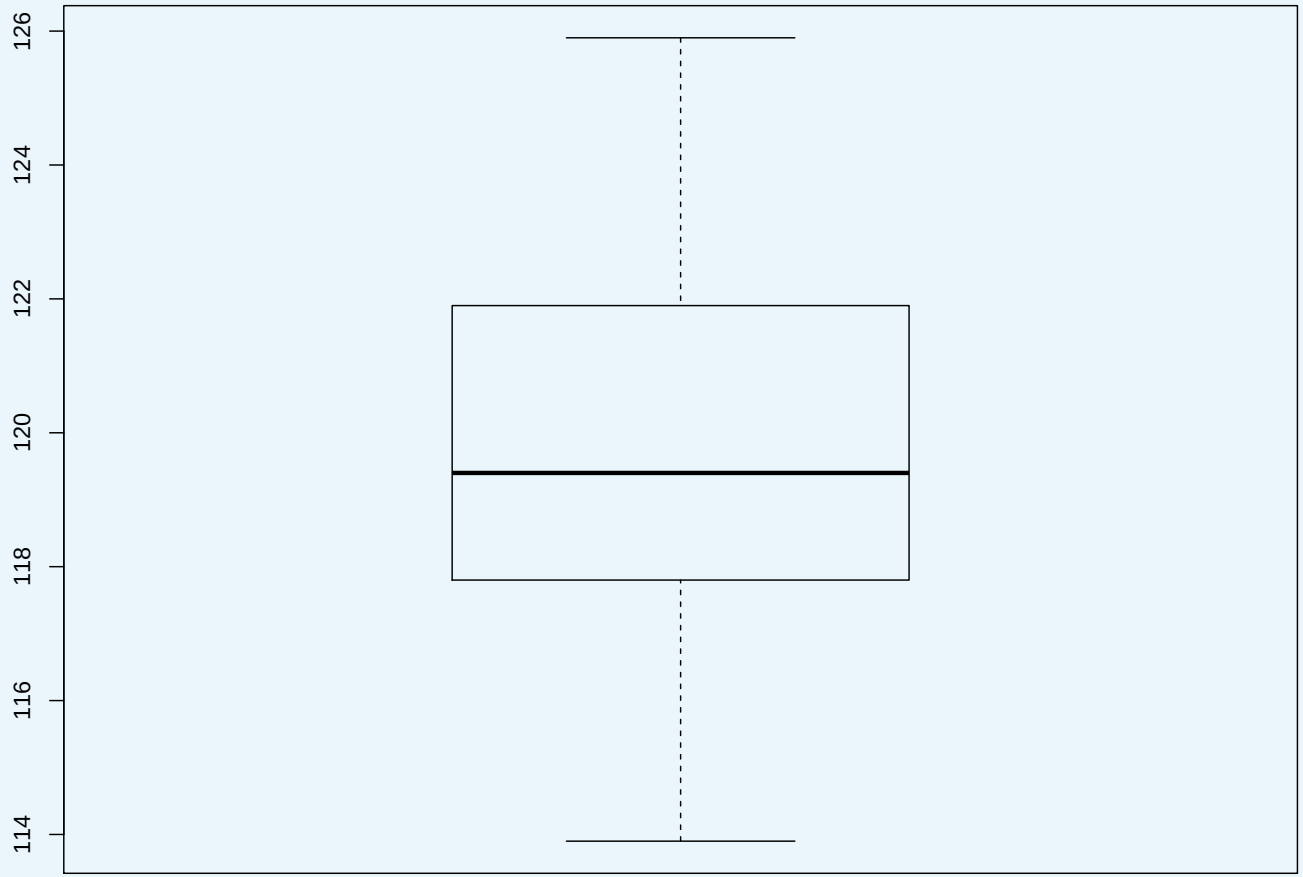
```
> range(x)          # what do you expect here?
[1] 113.9 125.9
> IQR(x)            # interquartile range = quantile(x, 3/4) - quantile(x, 1/4)
[1] 3.825
> sum(x)            # sum of the elements of x
[1] 1198.9
> length(x)         # count of the elements of x
[1] 10
> sd(x)/sqrt(length(x))   # standard error
[1] 1.14702
```

Nore statistical functions

```
> summary(x)       # summary statistics for x (some of them we already saw

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  113.9   118.1   119.4   119.9   121.9   125.9

> t.test(x)        # perform one sample t-tests on x

One Sample t-test

data:  x
t = 104.523, df = 9, p-value = 3.408e-15
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 117.2953 122.4847
sample estimates:
mean of x
   119.89
```

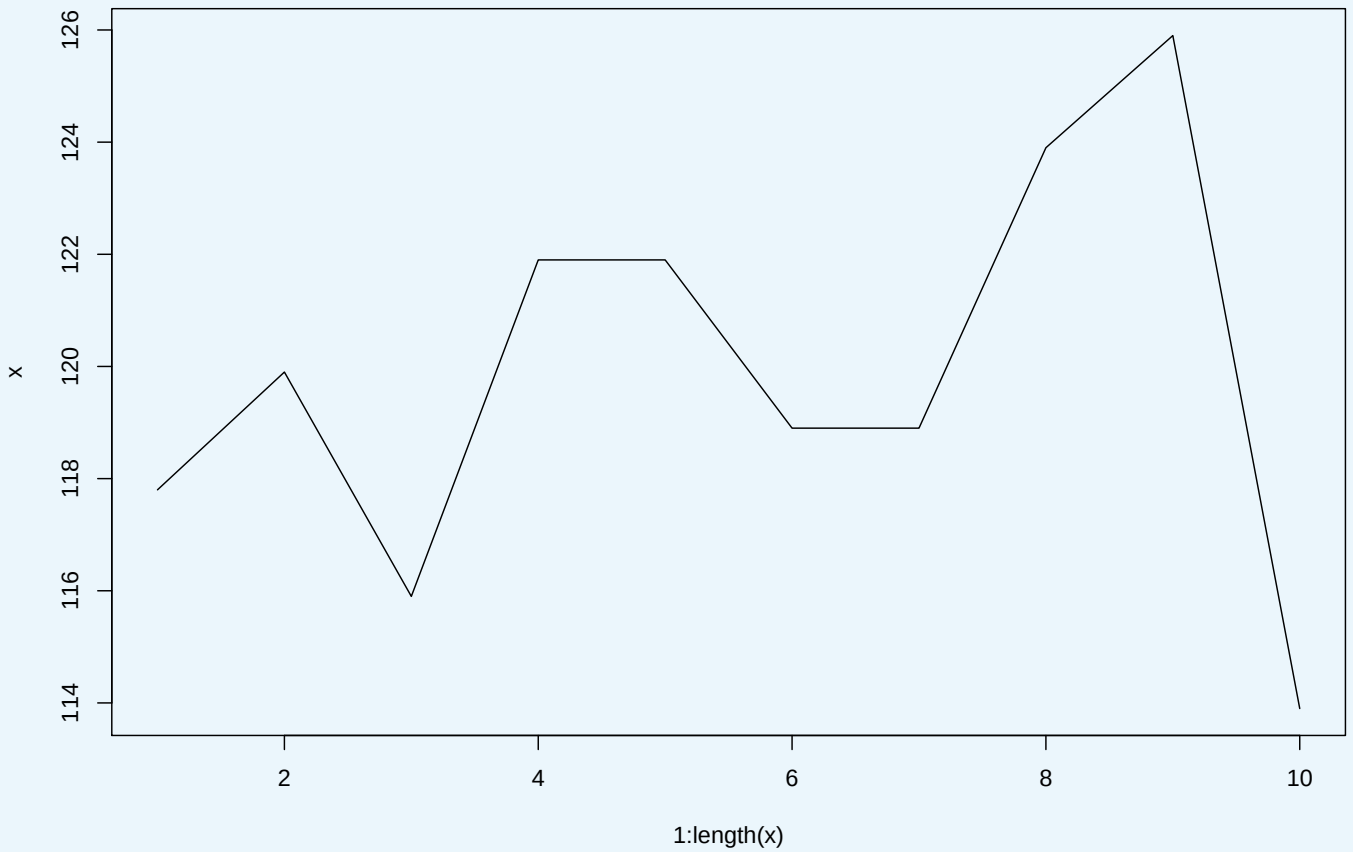
Some statistical graphics

```
> boxplot(x)
```



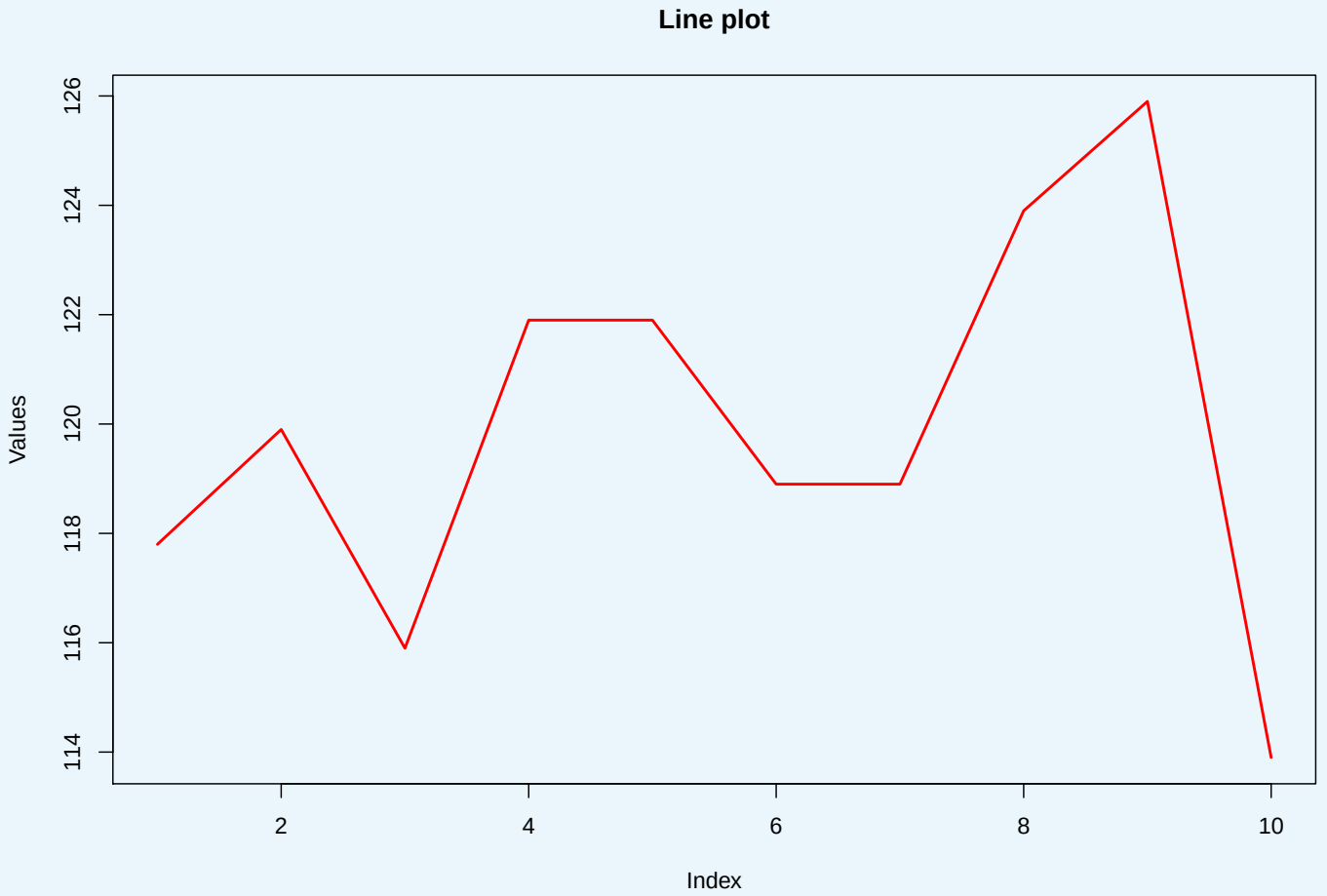
Some statistical graphics

```
> plot(1:length(x), x, type="l")
```



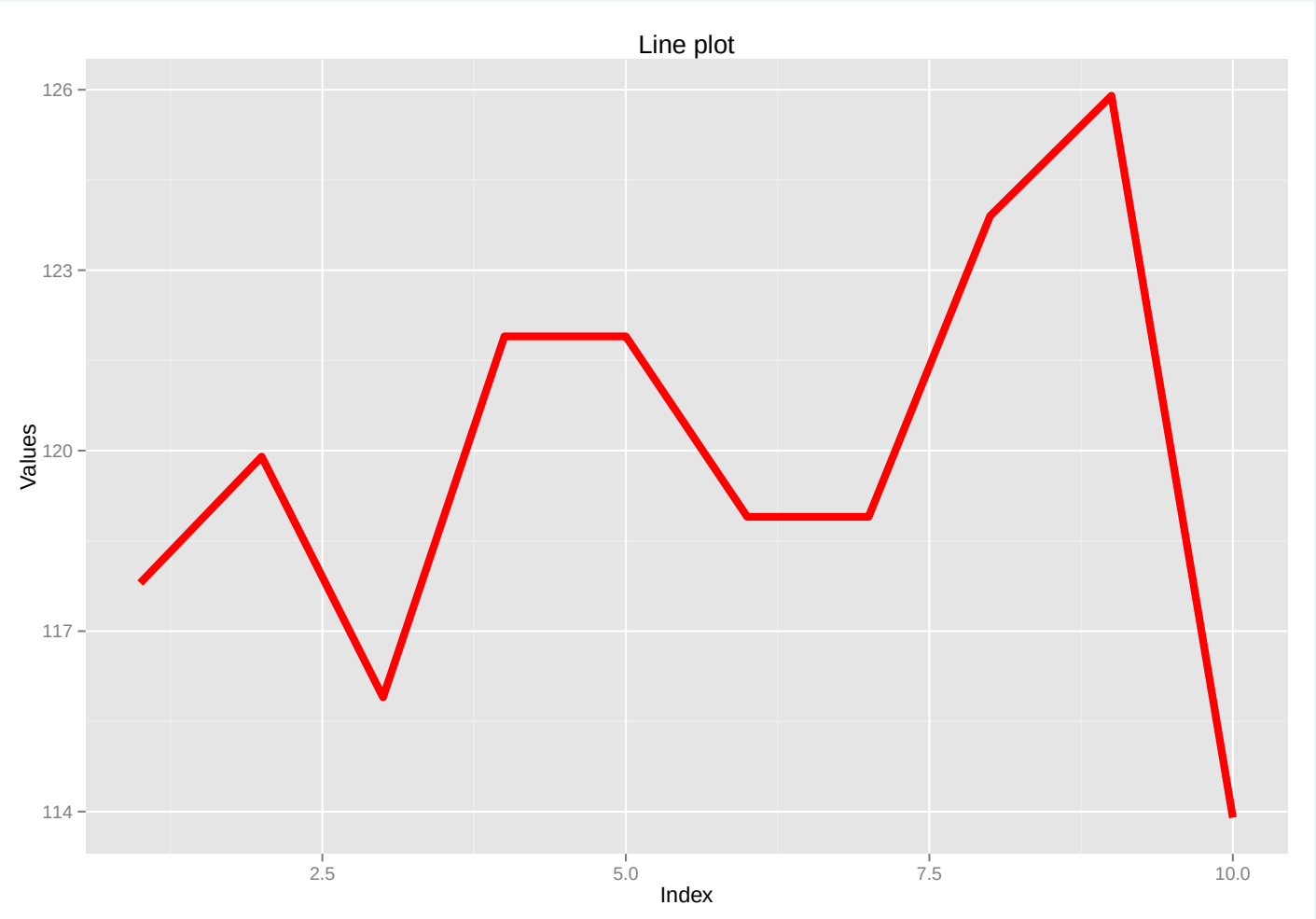
Some statistical graphics

```
> plot(1:length(x), x, type="l", col="red", lwd=2,
+      xlab="Index", ylab="Values", main="Line plot")
```



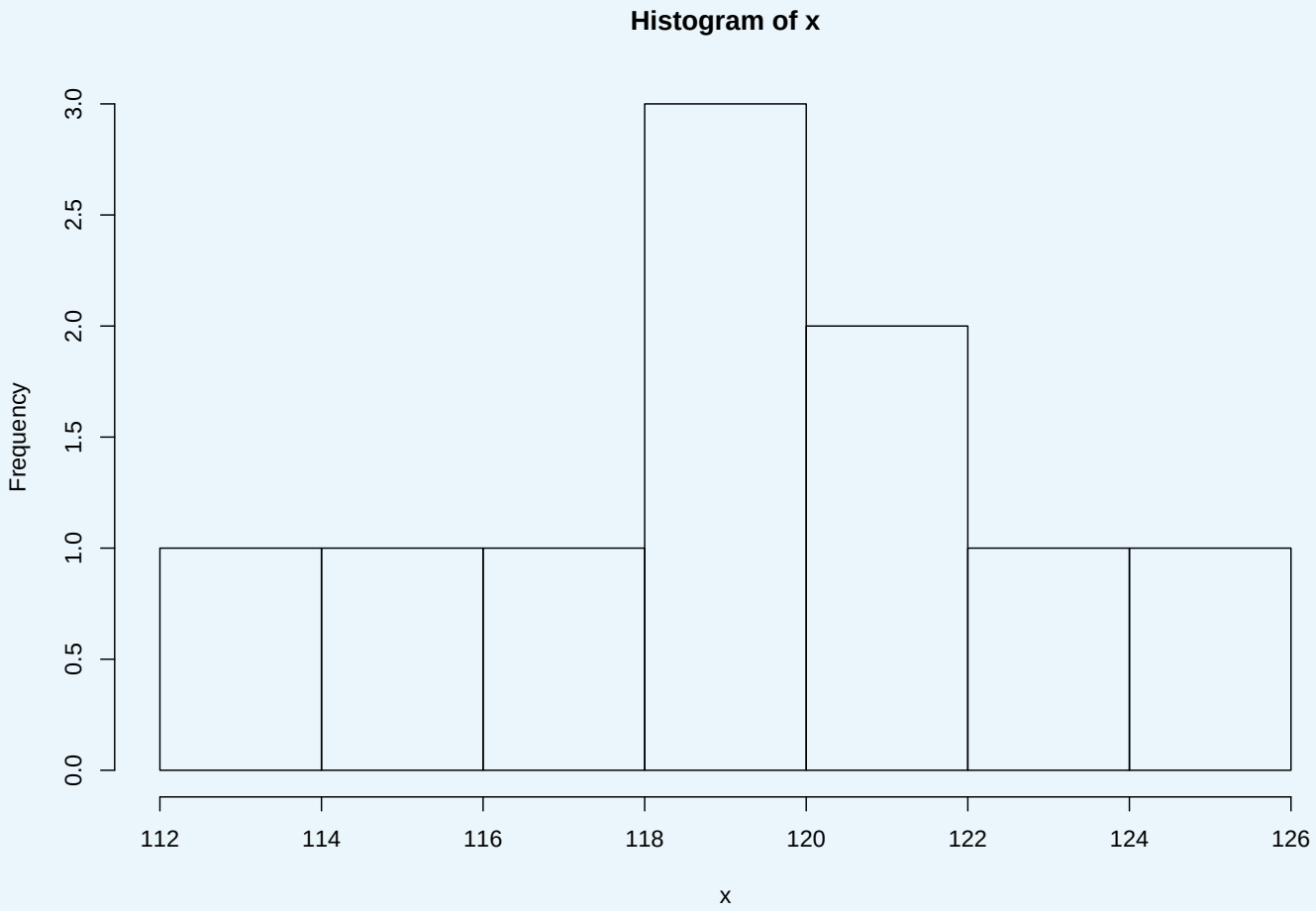
Some statistical graphics

```
> library(ggplot2)
> ggplot(data.frame(x), aes(x=1:10, y=x)) + geom_line(col="red", lwd=2) +
+     labs(x="Index", y="Values", title="Line plot")
```



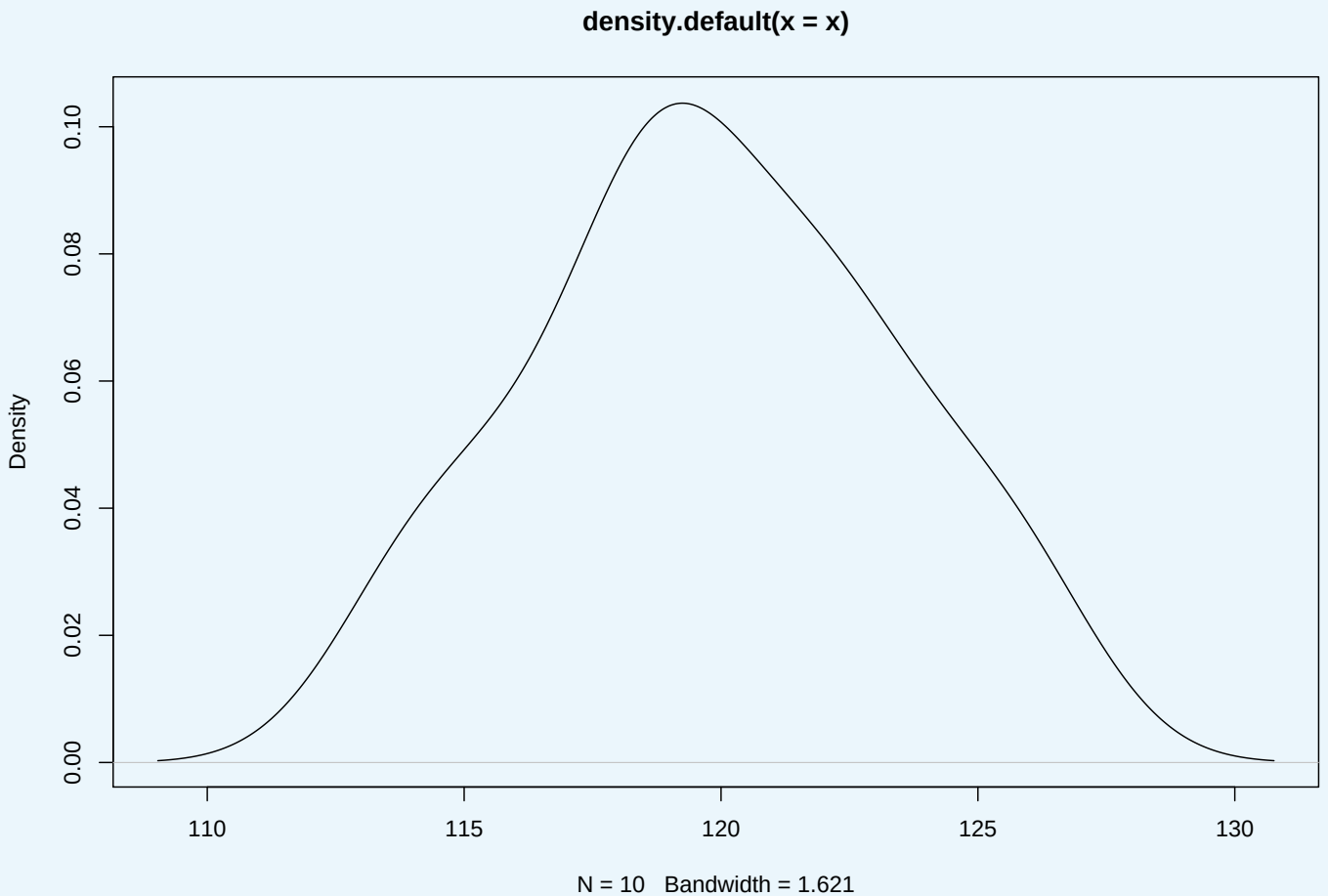
Some statistical graphics

```
> hist(x)
```



Some statistical graphics

```
> plot(density(x))
```



Missing values in R

- Missing data in R are represented as `NA` (stands for *Not Available*).
- No difference is made between string or numerical missing values.
- `NA` can be treated as any other value, e.g. we can create a vector and include missing values in it by writing:

```
> v <- c(10, 20, NA, 30, 40)
```

- `NA` is the one of the few non-numbers that could be included in `v` without generating an error.
- R distinguishes between `NA` and `"NA"` - the former is a missing value while the latter is simply a string.

```
> anum <- c(10, 20, NA, 30, 40)
> astr <- c("Oman", "Qatar", NA, NA, "Egypt", "NA")
> anum
[1] 10 20 NA 30 40
> astr
[1] "Oman"  "Qatar" NA      NA      "Egypt" "NA"
```

Missing values in R

- We cannot determine which variable has a missing value by comparison, i.e. `anum[1] == NA` is not a valid logical expression and will not return `FALSE` as one would expect but will return `NA`.
- Use the function `is.na()`, e.g.

```
> is.na(anum)

[1] FALSE FALSE  TRUE FALSE FALSE

> is.na(astr)

[1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

- To find the indexes of the missing values in a vector one could use the function `which()` in combination with `is.na()`:

```
> which(is.na(anum))

[1] 3

> which(is.na(astr))

[1] 3 4
```

Missing values in R

- To assign a NA value: use the assignment operator or the function is.na():

```
> x <- c(1, 4, 7, 10)
> x[4] <- NA                    # sets the 4th element to NA
> is.na(x) <- 1                 # sets the first element to NA
> x

[1] NA  4  7 NA
```

- Note the difference between NA and NULL
- These codes are not equivalent - while NA represents an element which is "not available", NULL denotes something which never existed and cannot exist at all.

Missing values in R

- Most of the summary functions in R can optionally exclude the missing values from calculations by using the argument is `na.rm` —by default is set to `FALSE`, meaning that the missing values are not removed.
- As a result of this the return value will be `NA`.

```
> x <- c(1, 4, NA, 10)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
    1.0     2.5     4.0     5.0     7.0    10.0       1
> mean(x)
[1] NA
> sd(x)
[1] NA
> mean(x, na.rm=TRUE)
[1] 5
> sd(x, na.rm=TRUE)
[1] 4.582576
```

Missing values in R

- `NaN` is used to denote a value which is "not a number".
- Can arise for example when we try to compute the undeterminate 0/0. To check whether a value is "not a number" the function `is.nan()` is used.

```
> x <- c(1, 0, 10)
> x/x
[1]   1 NaN   1
> is.nan(x/x)
[1] FALSE  TRUE FALSE
```

- Another special symbol is infinity `Inf` which results from computations like 1/0 — use `is.finite()` and `is.infinite()` to determine whether a number is finite or not.

```
> 1/x
[1] 1.0 Inf 0.1
> is.finite(1/x)
[1]  TRUE FALSE  TRUE
> -10/x
[1]  -10 -Inf   -1
```

Differences to other packages

Summary of Part 2

In this session we discussed the following R concepts:

- Data types, modes and classes
- R vectors
- Data frames, matrices, and arrays
- Lists
- Missing data