Basic R functions

Hauke Licht
University of Cologne

April 17, 2024

Introduction

Recap: Types of vectors

Terminology

- **vector**: a sequence of values
- **element**/**value**: a single value in a vector

```
vector <- c(1, 2, 3)
value <- vector[1]
```

Vector Types in R

In R, there are four main types of vectors:

1. `logical` for "boolean" (true/false) values
2. `integer` for whole numbers (e.g., … -99, … -1, 0, 1, 2, …)
3. `double` for real-valued numbers (e.g., 1.3)
4. `character` for representing **text**

Each type has its own characteristics and is used for different purposes.

NA values

> **Note**
>
> NA is the special value used in R to represent missing or undefined values

There are special NA values for each vector type:

1. `logical` → `NA`
2. `integer` → `NA_integer_`
3. `double` → `NA_real_` (don't ask me why it's not `NA_double_` 😅)
4. `character` → `NA_character_`

```
1 typeof(NA)
```

```
[1] "logical"
```

```
1 typeof(as.character(NA))
```

```
[1] "character"
```

```
1 typeof(NA_character_)
```

```
[1] "character"
```

Logical Vectors

- logical vectors are used to represent *boolean* values
- boolean values can be either TRUE or FALSE
- they are commonly used for logical operations and conditional statements.

Example

```
1 x <- c(TRUE, FALSE, TRUE)
2 typeof(x)
3 x
```

1

```
[1] "logical"
[1]  TRUE FALSE  TRUE
```

Logical Vectors

```r
is.logical(TRUE)
```

```
[1] TRUE
```

```r
is.logical(FALSE)
```

```
[1] TRUE
```

```r
is.logical(NA)
```

```
[1] TRUE
```

Integer Vectors

- integer vectors are used to represent whole numbers.
- they are commonly used for counting, indexing, and arithmetic operations.

Example

Important! Just typing numbers gives you "double" vectors (see next slide)

```
1  x <- c(1, 2, 3)
2  typeof(x)
```

```
[1] "double"
```

```
1  x
```

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

Need to convert to integer vector …

```
1  x <- as.integer(x)
2  typeof(x)
```

```
[1] "integer"
```

```
1  x
```

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

… or use L to declare integer explicitly

```
1  x <- c(1L, 2L, 3L)
2  typeof(x)
```

```
[1] "integer"
```

```
1  x
```

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

Functions for creating integer sequences

seq()

`seq()` generates a sequence of numbers

Arguments

- `from`: Starting value of the sequence
- `to`: Ending value of the sequence
- `by`: Step size (default: 1)
- `length.out`: Length of the sequence (alternative to `to`)
- `along.with`: Vector to match the length of the sequence with

Example

```
1  x <- seq(from = 1, to = 10, by = 2)
2  x
```

```
[1] 1 3 5 7 9
```

Functions for creating integer sequences

seq_len()

`seq_len()` generates a sequence of numbers from 1 to a specified length

Arguments

- `length.out`: Length of the sequence

Example

```
1  x <- seq_len(length.out = 5)
2  x
```

```
[1] 1 2 3 4 5
```

Functions for creating integer sequences

seq_along()

`seq_along()` generates a sequence of numbers from 1 to the length of a vector

Arguments

- `along.with`: Vector to match the length of the sequence with

Example:

```
x <- c("a", "b", "c")
y <- seq_along(along.with = x)
y
```

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

Double Vectors

- Double vectors are used to represent decimal numbers.
- They are commonly used for mathematical calculations and statistical analysis.

Example

```r
x <- c(1.5, 2.7, 3.9)
x
```

```
[1] 1.5 2.7 3.9
```

Character Vectors

- Character vectors are used to represent text or strings.
- They are commonly used for storing and manipulating **textual data**.

Example

```
1 x <- "Hello"
2 typeof(x)
```

```
[1] "character"
```

```
1 x
```

```
[1] "Hello"
```

Basic R functions for text wrangling

Overview

1. Functions for *creating* character vectors
2. Functions for *manipulating* (“changing”) character vectors
3. Functions for *analyzing* character vectors

Functions for creating character vectors

c()

`c()` combines values into a vector

Example

```r
x <- c("Hello", "World")
x
```

```
[1] "Hello" "World"
```

Functions for creating character vectors

rep()

`rep()` replicates values in a vector

Arguments

- `x`: Value to be replicated
- `times`: Number of times to replicate the value
- `each`: Number of times to repeat each value

Examples

```r
x <- rep("Hello", times = 3)
x
```

```
[1] "Hello" "Hello" "Hello"
```

```r
rep(c("Hello", "World"), each = 2)
```

```
[1] "Hello" "Hello" "World" "World"
```

```r
rep(c("Hello", "World"), times = 2)
```

```
[1] "Hello" "World" "Hello" "World"
```

Functions for creating character vectors

paste()

`paste()` combines or concatenates several character values into a single character value

Arguments

- `...`: Character vectors to be combined
- `sep`: Separator between the values (default: ” “)
- `collapse`: Separator between the combined values (default: NULL)

Example

```r
x <- c("a", "b", "c")
y <- c("1", "2", "3")
paste(x, y, sep = ":")
```

```
[1] "a:1" "b:2" "c:3"
```

```r
x <- c("a", "b", "c")
paste(x, collapse = ", ")
```

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

Functions for creating character vectors

sprintf()

`sprintf()` formats character values according to a specified format

Arguments

- `fmt`: Format string
- `...`: Values to be formatted

Example

```
1 sprintf(fmt = "Hello, %s! My name is %s", "friends", "Hauke")
```

```
[1] "Hello, friends! My name is Hauke"
```

Note: we use `%s` as a placeholder to insert a character value

Functions for creating character vectors

sprintf()

the most important formatting options (a.k.a “placeholders”) are:

- `%s` for inserting a character value
- `%d` for inserting an integer value
- `%f` for inserting a double value

See https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/sprintf for more options

Functions for creating character vectors

sprintf()

`sprintf()` is great for creating identifiers (IDs):

```
df <- data.frame(
  document_number = c(1, 1, 2, 2),
  paragraph_number = c(1, 2, 1, 2)
)
df$paragraph_id <- sprintf("d%02d_p%02d", df$document_number, df$paragraph_number)
df
```

```
  document_number paragraph_number paragraph_id
1               1                1      d01_p01
2               1                2      d01_p02
3               2                1      d02_p01
4               2                2      d02_p02
```

Note: `%02d` is a placeholder for a two-digit integer with leading zeros

```
sprintf("%02d", 8:12)
```

```
[1] "08" "09" "10" "11" "12"
```

```
sprintf("%03d", c(1, 10, 100))
```

```
[1] "001" "010" "100"
```

Functions for creating character vectors

Exercise

1. read the sample of the ParlSpeech2 *UK Hose of Commons* corpus I have created
2. group the data by *party* and *date* using `dplyr::group_by`
3. use `dplyr::summarize` and `paste` to aggregate ("combine") speeches by party and date
4. use `dplyr::mutate` and `sprintf` to create a new column with a unique identifier for each party–date text unit

```
# step 1
library(readr)
fp <- file.path("data", "datasets", "parlspeech2_gbr_sample.tsv")
df <- read_tsv(fp)
```

Functions for analyzing character vectors

nchar()

`nchar()` counts the number of characters in a character value

Arguments

- `x`: Character vector

Example

```
1  x <- "Hello, World!"
2  nchar(x)
```

```
[1] 13
```

Functions for analyzing character vectors

grepl()

`grepl()` tests if a pattern is present in a character vector

Arguments

- `pattern`: Pattern to be matched
- `x`: Character vector

Example

```
1 x <- "Hello, World!"
2 grepl("Hello", x)
```

```
[1] TRUE
```

```
1 grepl("banana", x)
```

```
[1] FALSE
```

```
1 grepl("hello", x, ignore.case = TRUE)
```

```
[1] TRUE
```

Note: be careful with `NA` values:

```
1 grepl(pattern = "Hello", x = NA)
```

```
[1] FALSE
```

```
1 grepl(pattern = NA, x = "Hello")
```

Functions for analyzing character vectors

grep()

`grep()` returns the indices of the values in a character vector that contain a pattern

Arguments

- `pattern`: Pattern to be matched
- `x`: Character vector
- `value`: Logical value (`TRUE`/`FALSE`) indicating whether to return the values instead of the indices

Example

```r
x <- c("apple", "banana", "cherry")
grep("a", x)
```

```
[1] 1 2
```

```r
grep("a", x, value = TRUE)
```

```
[1] "apple"  "banana"
```

Functions for analyzing character vectors

Exercise (*continued*)

1. Take the data frame of party–date level speech texts created previously
2. Use `dplyr::mutate` and `nchar` to create a new column with the number of characters in each text
3. Use `dplyr::mutate` and `grepl` to create an indicator "mentions_brexit" that is `True` if a party–date text unit contains the term "Brexit" and `False` otherwise
4. Summarize the proportion of party–date text units that contain the term "Brexit" by party and year
5. Analyze:
    1. In what year was the term "Brexit" most prevalent?
    2. Does the answer to 5.1 depend on the party?

Functions for manipulating character vectors

`trimws`

`trimws()` removes leading and trailing whitespace from a character value

Arguments

- `x`: Character vector

Example

```
x <- "  Hello, World!  "
trimws(x)
```

```
[1] "Hello, World!"
```

Functions for manipulating character vectors

`tolower()` and `toupper()`

`tolower()` and `toupper()` convert text to lowercase and uppercase, respectively

Arguments

- `x`: Character vector

Example

```
x <- "Hello, World!"
tolower(x)
```

```
[1] "hello, world!"
```

```
toupper(x)
```

```
[1] "HELLO, WORLD!"
```

Functions for manipulating character vectors

`substr()`

`substr()` extracts a substring from a character value

Arguments

- `x`: Character vector
- `start`: Starting position of the substring
- `stop`: Ending position of the substring

Example

```
1  x <- "Hello, World!"
2  substr(x, start = 1, stop = 5)
```

```
[1] "Hello"
```

Functions for manipulating character vectors

`sub` and `gsub`

- `sub()` replaces the first occurrence of a pattern in a character value
- `gsub()` replaces all occurrences of a pattern in a character value

Arguments

- `pattern`: Pattern to be replaced
- `replacement`: Replacement value
- `x`: Character vector

Example

```r
x <- "Hello, World!"
sub(pattern = "World", replacement = "Universe", x)
```

```
[1] "Hello, Universe!"
```

```r
gsub(pattern = "o", replacement = "🤗", x)
```

```
[1] "Hell🤗, W🤗rld!"
```

Functions for manipulating character vectors

strsplit

`strsplit()` splits a character value into substrings based on a specified delimiter

Arguments

- `x`: Character vector
- `split`: Delimiter to split the character value

Example

```r
x <- "Hello, World!"
strsplit(x, split = ", ")
```

```
[[1]]
[1] "Hello"  "World!"
```

Important: `strsplit()` returns a **list** of character vectors

Functions for manipulating character vectors

Exercise (*continued*)

1. Take the data frame of party–date level speech texts with the "mentions_brexit" created previously
2. locate the character positions where the term "Brexit" occurs
3. *advanced*
    1. for each occurrence, extract the term "Brexit" ± 20 characters left and right of it
    2. what are the 20 terms that most freuquently co-occur in the ±20 character window with the term "Brexit"?

stringr

The `stringr` package

- The `stringr` package provides a set of functions for working with character vectors
- The functions in `stringr` are designed to be more consistent and easier to use than the base R functions

How to

```
1 library(stringr)
```

Note: `stringr` is part of the "tidyverse" and is loaded automatically when you load the `tidyverse` package

`stringr` equivalents to base R functions

- `str_c()` is equivalent to `paste()`
- `str_length()` is equivalent to `nchar()`
- `str_to_lower()` and `str_to_upper()` are equivalent to `tolower()` and `toupper()`
- `str_sub()` is equivalent to `substr()`
- `str_replace()` and `str_replace_all()` are equivalent to `sub()` and `gsub()`
- `str_split()` is equivalent to `strsplit()`
- `str_detect()` is equivalent to `grepl()`
- `str_locate()` is equivalent to `grep()`

`stringr` equivalents to base R functions

Practical advantage: `stringr` functions are defined and named more consistently

- the first argument is always the character vector
- if applicable, the second argument is always the pattern
- if applicable, the third argument is always the replacement value

Examples

```
1 str_replace("Hello, World!", pattern = "World", replacement = "Universe")
```

```
[1] "Hello, Universe!"
```

```
1 # instead of `sub(World, "Universe", "Hello, World!")`
2
3 str_detect("Hello, World!", "Hello")
```

```
[1] TRUE
```

```
1 # instead of `grepl("Hello", "Hello, World!")`
2
3 str_locate("Hello, World!", "World")
```

```
     start end
[1,]     8  12
```

```
1 # instead of `grep("World", "Hello, World!")`
```

Other use stringr functions

- `str_trim()`: removes leading and trailing whitespace
- `str_pad()`: pads a string with spaces
- `str_wrap()`: wraps a string to a specified width
- `str_sort()`: sorts a character vector
- `str_order()`: returns the order of a character vector
- `str_replace_na()`: replaces `NA` values with a specified value
- `str_extract()`: extracts a pattern from a character vector