Text Mining and Sentiment Analysis

Prof. Annamaria Bianchi
A.Y. 2024/2025

Lecture 2

18 February 2025

Outline

What are stings?

Strings in R

Creating strings

String length

Subsetting strings

Combining strings

Convert case of a string

Extracting data from string

Packages: **stringr, tidyr**

Functions: `stringr::str_length(), str_sub(), str_c(), str_to_lower(), str_to_upper(), str_to_title(), tidyr::separate_longer_delim(), separate_wider_delim()`

What are strings?

- A string is a set of characters. String represent textual content and can contain numbers, spaces and special characters.
- Strings are enclosed in quotation marks (single or double) for the data to recognized as a string and not a number or variable name.
- For example, the word "hamburger" and the phrase "I ate 3 hamburgers" are both strings. Even "12345" could be considered a string, if specified correctly.

Strings in R

- In R strings should be of type **character**
- Notice that **strings are not to be considered factors.** R’s default understanding of text strings is to treat them as individual factors like ‘Monday’, ‘Tuesday’ and so on with distinct levels.
- For text mining, we are aggregating strings to distill meaning, so treating the strings as individual factors makes aggregation impossible

Class exercise

1. Create a vector (named nn) containing numbers 1, 2, 3.
2. Check the type of the vector.
3. Convert the vector to character type.

Creating a string

You can create strings with either single quotes or double quotes.

```
> string1 = "This is a string"          #creating a string with double quotes
> string2 = ‘This is also a string’     #creating a string with single quotes
```

We can check the type

```
> class(string1)
[1] "character"
> class(string2)
[1] "character"
```

Creating a string

If you forget to close a quote, you will see + (the continuation character):

```
> string3 = "This is a string without a closing quote
+
+
+
```

If this happens to you, press **Esc**

Creating a string

In case you want to create a string that contains double quotes, you need to create it using single quotes

```
> string4 = 'To put a "quote" inside a string, use single quotes'
> string4
[1] "To put a \"quote\" inside a string, use single quotes"
```

To include a single or double quote in a string you can use \ to «escape» it

```
> string5 = "To put a \"quote\" inside a string, you can use the backslash"
> string5
[1] "To put a \"quote\" inside a string, you can use the backslash"
```

Creating a string

Notice that the printed representation of a string is not the same as the string itself because the printed representation shows the escapes

To see the raw contents of the string, use `str_view()` or the base R function `writeLines()`

```
> str_view(string5)
[1] │ To put a "quote" inside a string, you can use the backslash

> writeLines(string5)
To put a "quote" inside a string, you can use the backslash
```

Creating a string

You can also create a vector of strings

```
> c("one", "two", "three")
[1] "one" "two" "three
```

An empty string is represented by using “”

String manipulation

String manipulation: detect matches, subset strings, lengths, mutate, join, split and order.

Regular expressions: describe a specific set of strings and it is used for string matching, replacing and removing

stringr package

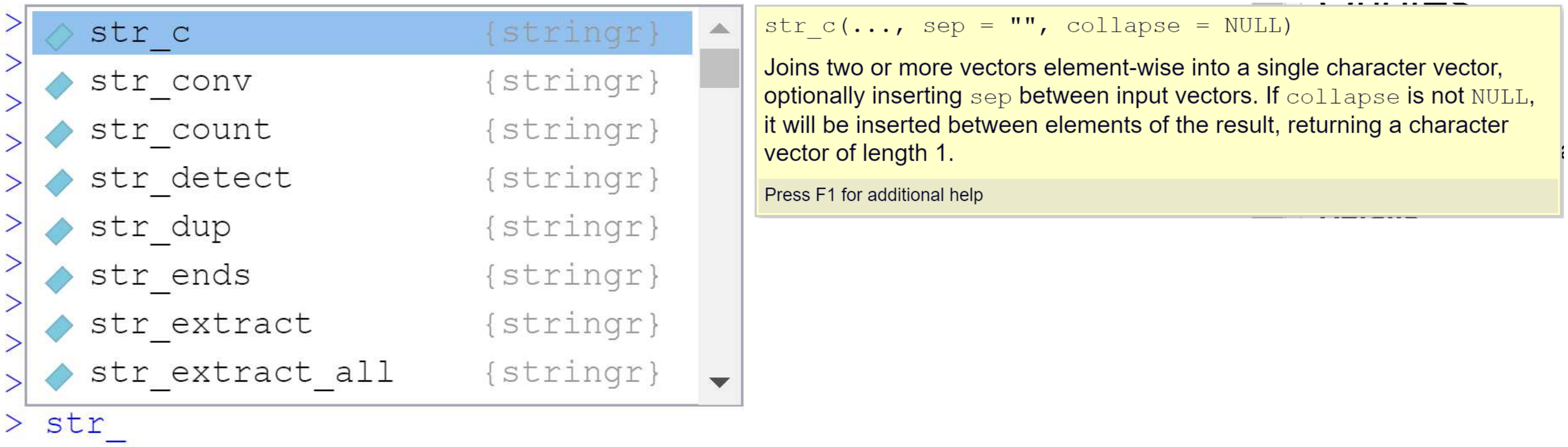
- R has many functions for string manipulation automatically installed within the base software version. But because they have grown organically over time, they can be inconsistent and a little hard to learn.
- In addition, the common libraries extending R’s string functionality are **stringi** and **stringr**. These packages provide simple implementations for dealing with character strings.
- **stringr** provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of **stringi**, which uses the ICU C library to provide fast, correct implementations of common string manipulations.
- **stringr** is part of **tidyverse**, that is a set of packages sharing common data representations and API design. The tidyverse package is designed to make it easy to install and load core packages from the tidyverse in a single command.

stringr package

- Load the core tidyverse packages (ggplot2, dplyr, tidyr, readr, purr, tibble, **stringr**, forcats):

  ```
  library(tidyverse) # Load the core tidyverse packages
  ```

- All functions in stringr start with `str_` and take a **vector of strings** as the first argument (Vectored functions)
- The prefix `str_` is particularly useful in Rstudio, because typing `str_` will trigger autocomplete, allowing you to see all stringr functions



```
str_c            {stringr}
str_conv         {stringr}
str_count        {stringr}
str_detect       {stringr}
str_dup          {stringr}
str_ends         {stringr}
str_extract      {stringr}
str_extract_all  {stringr}
> str_
```

```
str_c(..., sep = "", collapse = NULL)
```

Joins two or more vectors element-wise into a single character vector, optionally inserting `sep` between input vectors. If `collapse` is not `NULL`, it will be inserted between elements of the result, returning a character vector of length 1.

Press F1 for additional help

string package

A number of functions are available to manipulate strings. Today we will see the following:

`str_length()`

`str_sub()`

`str_c()`

`str_to_lower(), str_to_upper(), str_to_title()`

Basic syntax: `str_`*`fname`*`(`*`string`*`,…)`

All ‘stringr’ functions are vectorized.

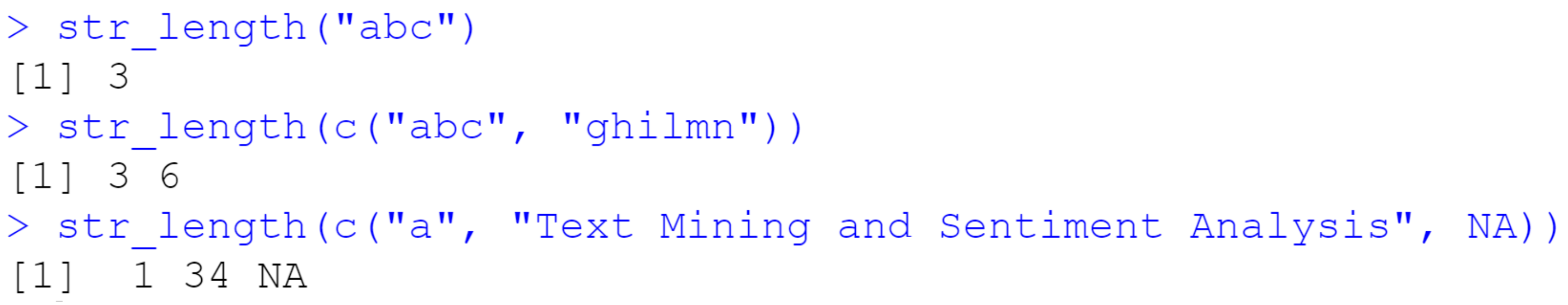
String Length

`str_length()` returns the number of characters in a string:

```
str_length(string)
```

`string` input vector. Either a character vector or something coercible to one.

String Length



```
> str_length("abc")
[1] 3
> str_length(c("abc", "ghilmn"))
[1] 3 6
> str_length(c("a", "Text Mining and Sentiment Analysis", NA))
[1]  1 34 NA
```

Notice that **spaces** are counted as characters. **Missing** strings have missing length

Subsetting strings

`str_sub()` extracts and replaces substrings from a character vector. It takes start and end arguments that give the (inclusive) position of the substring

```
str_sub(string, start = 1, end = -1)
str_sub(string, start = 1, end = -1) = value
```

`string` input character vector

`start, end` two integer vectors: `start` gives the position of the first character (defaults to first), `end` gives the position of the last (defaults to last character). Negative values count backwards from the last character.

`value` replacement string

Subsetting strings

```
> x = c("Apple", "Banana", "Pear")
> x
[1] "Apple" "Banana" "Pear"

> str_sub(x, start = 1, end = 3)
[1] "App" "Ban" "Pea"
```

You can use negative values to count back from the end of the string

```
> str_sub(x, start = -3, end = -1)
[1] "ple" "ana" "ear"
> str_sub(x, start = -3)
[1] "ple" "ana" "ear"
```

Subsetting strings

`str_sub()` won’t fail if the string is too short: it will just return as much as possible

```
> str_sub("a", 1, 5)
[1] "a"
```

You can also use the assignment form to modify strings

```
> str_sub(x, 1, 1) = c("a", "b", "p")
> x
[1] "apple" "banana" "pear"
```

Combining strings

`str_c()` joins two or more vectors element-wise into a single character vector, optionally inserting sep between input vectors. If collapse is not NULL, it will be inserted between elements of the result, returning a character vector of length 1.

```
str_c(..., sep = "", collapse = NULL)
```

Arguments

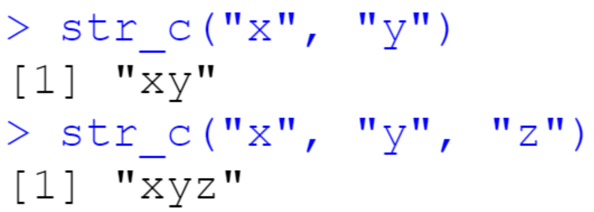
`...`	One or more character vectors.
`sep`	String to insert between input vectors.
`collapse`	Optional string used to combine output into single string.

Output

If collapse = NULL (the default) a character vector with length equal to the longest input. If collapse is a string, a character vector of length 1.

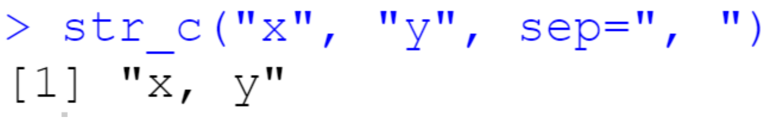
Combining strings

Basic usage



```
> str_c("x", "y")
[1] "xy"
> str_c("x", "y", "z")
[1] "xyz"
```

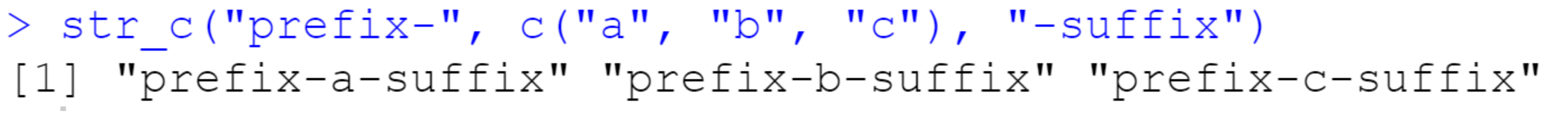
Use the `sep` argument to control how elements are separated:



```
> str_c("x", "y", sep=", ")
[1] "x, y"
```

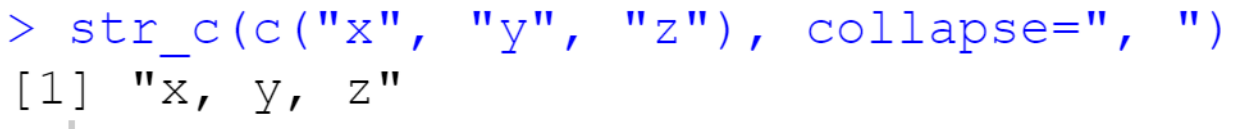
Combining strings

`str_c()` is vectorized and it automatically recycles shorter vectors to the same length of the longest



```
> str_c("prefix-", c("a", "b", "c"), "-suffix")
[1] "prefix-a-suffix" "prefix-b-suffix" "prefix-c-suffix"
```

Use `collapse` to collapse a vector of strings into a single string



```
> str_c(c("x", "y", "z"), collapse=", ")
[1] "x, y, z"
```

Combining strings

Exercise.

1. Create a tibble containing three names
2. Add a variable with greetings (e.g. «Hello Maria!»)

Covert case of a string

To covert case of a string

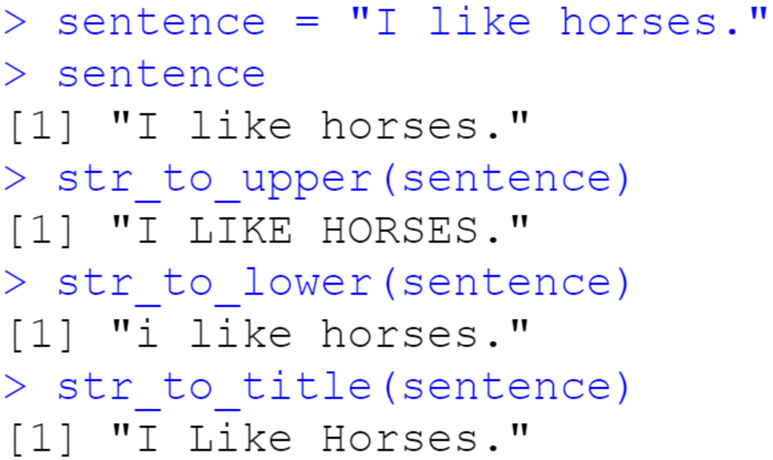
```
str_to_upper(string, locale = "en")

str_to_lower(string, locale = "en")

str_to_title(string, locale = "en")
```

`string`	String to modify
`locale`	Locale to use for translations. Defaults to "en" (English) to ensure consistent default ordering across platforms.

Covert case of a string



```
> sentence = "I like horses."
> sentence
[1] "I like horses."
> str_to_upper(sentence)
[1] "I LIKE HORSES."
> str_to_lower(sentence)
[1] "i like horses."
> str_to_title(sentence)
[1] "I Like Horses."
```

Extracting data from strings

It is rather common to have several variables packed into a single string. Some **tidyr** functions can be used to extract them:

```
separate_longer_delim()
separate_wider_delim()
```

`_longer` function make the input data frame longer by creating new rows, **_**wider function make the input data frame wider by generating new columns

`_delim` refers to the fact that these functions split string on the basis of a delimiter

Split a string into rows

`tidyr::separate_longer_delim()` takes a string and splits it into multiple rows by a delimiter

```
separate_longer_delim(data, cols, delim, ...)
```

Arguments

`data` A data frame

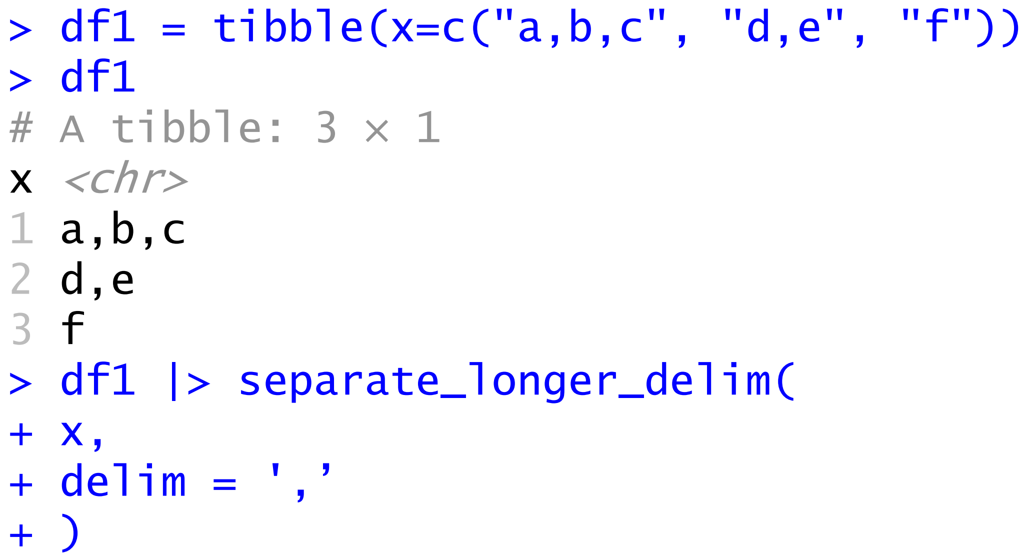
`cols` Columns to separate

`delim` string giving the delimiter between values

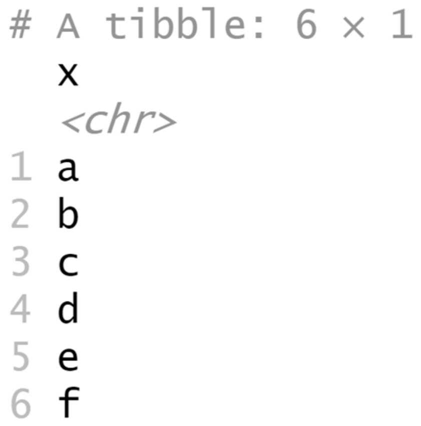
Output

A data frame based on data. It has the same columns, but different rows.

Split a string into rows



```
> df1 = tibble(x=c("a,b,c", "d,e", "f"))
> df1
# A tibble: 3 × 1
x <chr>
1 a,b,c
2 d,e
3 f
> df1 |> separate_longer_delim(
+ x,
+ delim = ',’
+ )
```



```
# A tibble: 6 × 1
  x
  <chr>
1 a
2 b
3 c
4 d
5 e
6 f
```

Split a string into columns

`tidyr::separate_wider_delim()` takes a string and splits it into multiple new columns by delimiter

```
separate_wider_delim(data, cols, delim, names...)
```

Arguments

`data`	A data frame
`cols`	Columns to separate
`delim`	string giving the delimiter between values
`names`	a character vector of output column names. Use NA if there are components that you don't want to appear in the output

Output

A data frame based on data. It has the same rows, but different columns.

Split a string into columns

In the tibble df2, x is made up of a code, an edition number, and a year, separated by '.'.

```
> df2 = tibble(x = c("a10.1.2022", "b10.1.2011", "e15.1.2015"))
> df2
# A tibble: 3 × 1
x <chr>
1 a10.1.2022
2 b10.1.2011
3 e15.1.2015
> df2 |> separate_wider_delim(
+ x,
+ delim = ".",
+ names = c("code", "edition", "year")
+ )
```

Class exercise

1) Create a vector of strings (named x) containing the words: *why, video, cross, extra, deal*
2) Compute the length of elements of x.
3) Compute the mean length of elements of x.
4) Combine all the words in x in a single character vector of length 1. Separate words using a comma.
5) Create an object y that combines the first four words in x in a single character vector of length 1. Separate words using blank space.
6) Extract the first letter of each word in x.

Exercises for you

Exercise 1

1. Create a vector of strings (named xx) containing the elements: a, abc, abcd, abcde, abcdef
2. Use the functions `str_length()` and `str_sub()` to extract the middle character from each string. What will you do for strings that have an even number of characters?

Exercise 2

Write a function that turns the vector `c(“a”, “b”, “c”)` into the string `“a, b, and c”.`

Exercises for you

Exercise 3

1. Create a vector, named *fruits*, containing the words: apple, banana, pear, persimmon, kiwi, mango, orange.
2. Compute the maximum length of elements of *fruits*. Which fruit name has maximum length?
3. Try to answer the previous point using the pipe operator.
4. Create the vector *list_fruits* containing the list of all fruits (character vector of length 1, with elements separated by comma).
5. Create the vector *fcolor* containing the colors of each fruit in the fruits vector. Combine each fruit with the corresponding color.
6. Substitute the fourth element of each fruit name with the symbol -.