Video 23: strings and stringr

Stats 102A

Miles Chen, PhD

Introduction

Most of statistical computing involves working with numeric data. However, many modern applications have considerable amounts of data in the form of text.

There are whole areas of statistics and machine learning devoted to organizing and interpreting text-based data, such as textual data analysis, linguistic analysis, text mining, sentiment analysis, and natural language processing (NLP).

For more information and resources:

- https://cran.r-project.org/web/views/NaturalLanguageProcessing.html
- https://www.tidytextmining.com/

Text-based analyses are beyond the scope of this course.

Resources

We will discuss the most common syntax and functions for string manipulation. For more information and resources:

Books and Articles

- Gaston Sanchez's "Handling Strings with R": https://www.gastonsanchez.com/r4strings/
- Garrett Grolemund and Hadley Wickham's "R for Data Science": http://r4ds.had.co.nz/strings.html
- https://en.wikibooks.org/wiki/R_Programming/Text_Processing
- https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html

Characters in R

Characters in R

Symbols in R that represent text or words are called **characters.**

A **string** is a character variable that contains one or more characters, but we often will use "character" and "string" interchangeably.

Values that are stored as characters have base type `character` and are typically printed with quotation marks.

```r
x <- "Hello World"
x
```

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

```r
typeof(x)
```

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

Creating Character Strings

Characters can be created using single or double quotation marks.

Single quotation marks can be used within double quotation marks and vice versa, but you cannot directly insert single quotes within single quotes or double quotes within double quotes.

```r
"This is the 'R' Language"
```

```
[1] "This is the 'R' Language"
```

```r
'This is the "R" Language'
```

```
[1] "This is the \"R\" Language"
```

```r
"This is an "error""
```

```
## Error: unexpected symbol in ""This is an "error"
```

Note: The double quotation inside a string is a special character, which is why it prints with a backslash `\"`.

Empty Characters

The `character()` function creates a character vector of a specified length. The default value in each element of the vector is the **empty character** `""`.

```
character(5)
```

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

Note: The empty character `""` is **not the same** as `character(0)` which is a character vector of length 0. `c("")` is identical to `character(1)`

The `paste()` Function

The **`paste()`** function is one of the most important functions for creating and building strings.

The `paste()` function inputs one or more R objects, converts them to `character`, and then concatenates (pastes) them to form one or several character strings.

The basic syntax is:

```r
paste(..., sep = " ", collapse = NULL)
```

- The **`...`** argument means the input can be any number of objects.
- The optional **`sep`** argument specifies the separator between characters after pasting. The default is a single whitespace **`" "`**.
- The optional **`collapse`** argument specifies characters to separate the result.

The **`paste0()`** function is equivalent to using `paste(..., sep = "", collapse)`

The paste() Function

```r
paste("I ate some", pi, "and it was delicious.")
```

```
[1] "I ate some 3.14159265358979 and it was delicious."
```

```r
paste("Bears", "Beets", "Battlestar Galactica", sep = ", ")
```

```
[1] "Bears, Beets, Battlestar Galactica"
```

```r
paste("h", c("a", "e", "i"), sep = "") # No collapsing produces vector
```

```
[1] "ha" "he" "hi"
```

```r
paste(c("a", "b", "c"), 1:3, sep = "") # pastes element-wise
```

```
[1] "a1" "b2" "c3"
```

```r
paste("a", 1:3, sep = "", collapse = ", ") # recycles + collapses
```

```
[1] "a1, a2, a3"
```

Note: Partial recycling will not throw a warning.

Print Functions for Characters

There are several functions to output strings:

- `print()` is for generic printing.
- `cat()` is for concatenation.
- `format()` is for (pretty) printing with special formatting.

`print()` and `noquote()`

The `print()` function (technically the `print.default()` method) has an optional logical `quote` argument that specifies whether to print characters with or without quotation marks.

A similar output can be produced using `noquote()`.

```r
print(x, quote = FALSE)
```

```
[1] Hello World
```

```r
noquote(x)
```

```
[1] Hello World
```

Note: While the output appears identical, the commands are not the same. The `noquote()` function outputs a `noquote` class object, which is then inputted into the `print.noquote()` method.

The `cat()` Function

The `cat()` funtion concatenates multiple character vectors into a single vector, adds a specified separator, and outputs the result (without quotations).

```
cat(x, "Hello Universe", sep = ", ")
```

```
Hello World, Hello Universe
```

The printing is slightly different from that of `noquote()`. In particular, the printed output does not have the vector index, and the `cat()` function returns an invisible `NULL` (meaning assigning the printed output to a variable does not work).

The `cat()` Function

One benefit of `cat()` is that the printed output can be saved to an external file using the **`file`** argument:

```r
cat(x, "Hello Universe",
    sep = ", ", file = "hello.txt"
)
```

When `file` is specified, an optional logical argument **append** specifies whether the result should be appended to or overwrite an existing file.

Side Note: There are a few other optional arguments that are useful for longer text strings. Consult the R documentation for more information.

The `format()` Function

The `format()` function formats an R object for “pretty” printing.

Some useful arguments used in `format()`:

- `width` specifies the (minimum) width of strings produced.
- `trim` specifies whether there should be no padding with spaces (`TRUE`).
- `justify` controls how padding takes place for strings. Takes the values `"left"`, `"right"`, `"centre"`, or `"none"`.

For controling the printing of numbers:

- `digits` specifies the number of significant digits to use.
- `nsmall` specifies the minimum number of digits to the right of the decimal point to include.
- `scientific` specifies whether to use scientific notation (`TRUE`) or standard notation (`FALSE`).

The `format()` Function

```r
format(1 / (1:5), digits = 2)
```

```
[1] "1.00" "0.50" "0.33" "0.25" "0.20"
```

```r
format(1 / (1:5), digits = 2, scientific = TRUE)
```

```
[1] "1.0e+00" "5.0e-01" "3.3e-01" "2.5e-01" "2.0e-01"
```

```r
format(c("Hello", "world", "Hello", "Universe"),
  width = 10, justify = "left"
)
```

```
[1] "Hello     " "world     " "Hello     " "Universe  "
```

Basic String Manipulation

Functions for Basic String Manipulation

There are many functions in base R for basic string manipulation.

- `nchar()` Returns number of characters
- `tolower()` Converts to lower case
- `toupper()` Converts to upper case
- `chartr()` Translates characters
- `substr()` Extracts substrings of a character vector
- `strsplit()` Splits strings into substrings

Examples of Basic String Manipulation

```r
y <- c("Hello", "World", "Hello", "Universe")
nchar(y)
```

```
[1] 5 5 5 8
```

```r
tolower(y)
```

```
[1] "hello"    "world"    "hello"    "universe"
```

```r
toupper(y)
```

```
[1] "HELLO"    "WORLD"    "HELLO"    "UNIVERSE"
```

Examples of Basic String Manipulation

```r
chartr("H", "y", y)
```

```
[1] "yello"    "World"    "yello"    "Universe"
```

```r
chartr("elio", "31!0", y)
```

```
[1] "H3110"    "W0r1d"    "H3110"    "Un!v3rs3"
```

Examples of Basic String Manipulation

```r
substr(x, 2, 9)
```

```
[1] "ello Wor"
```

```r
strsplit(x, split = " ")
```

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

```r
strsplit(x, split = "o")
```

```
[[1]]
[1] "Hell" " W"   "rld"
```

stringr

stringr documentation

Examples and documentation can be found in:

- https://www.gastonsanchez.com/r4strings/stringr-basics.html
- https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html

The `stringr` Package

The **`stringr`** package is part of the core tidyverse: it is automatically loaded by typing `library(tidyverse)` or alone with `library(stringr)`.

```r
library(tidyverse)
```

The functions in `stringr` are intended to make R's string functions more consistent, simpler, and easier to use. The primary ways it does this are:

- Function and argument names (and positions) are consistent.
- All functions deal with NAs and zero length characters appropriately.
- The output data structures from each function matches the input data structures of other functions.

Basic `stringr` Functions

The `stringr` package has functions for basic manipulation and for regular expression operations.

The main `stringr` functions for basic manipulation are shown below.

- **`str_c()`** String concatenation, similar to `paste()`
- **`str_length()`** Number of characters, similar to`nchar()`
- **`str_sub()`** Extracts substrings similar to `substr()`
- **`str_dup()`** Duplicates characters
- **`str_trim()`** Removes leading and trailing whitespace
- **`str_pad()`** Pads a string
- **`str_wrap()`** Wraps a string paragraph
- **`str_trim()`** Trims a string

The str_c() Function

The `str_c()` function automatically removes `NULL` and `character(0)` arguments.

```r
paste(x, NULL, character(0), c("Hello Universe"),
      sep = ", "
)
```

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

```r
str_c(x, NULL, character(0), c("Hello Universe"),
      sep = ", "
)
```

```
character(0)
```

Note: The default separator for `paste()` is whitespace (`sep=" "`) and for `str_c()` is the empty string (`sep=""`).

The `str_length()` Function

The `str_length()` function works with factors, whereas `nchar()` does not.

```r
nchar(factor(month.name))
```

```
Error in nchar(factor(month.name)): 'nchar()' requires a character vector
```

```r
str_length(factor(month.name))
```

```
 [1] 7 8 5 5 3 4 4 6 9 7 8 8
```