4. Strings and Text

Text Manipulation with paste, glue, stringr, and Number Formatting
Dr. Paul Schmidt

To install and load all packages used in this chapter, run the following code:

```
for (pkg in c("glue", "scales", "stringr", "tidyverse")) {
  if (!require(pkg, character.only = TRUE)) install.packages(pkg)
}

library(glue)
library(scales)
library(stringr)
library(tidyverse)
```

Introduction

In data analysis, we constantly work with text: assembling file names, cleaning column names, standardizing categories, creating labels for graphics. Formatting numbers for reports and tables is also part of this – percentages, thousands separators, p-values.

R offers various tools for this – from the built-in functions `paste()` and `paste0()`, to the elegant {glue} package, to the powerful manipulation functions from {stringr}, and specialized formatting functions from {scales}.

This chapter shows the most important techniques for typical data cleaning tasks and value formatting for reports.

Example Data

For this chapter, we create a small dataset with typical “dirty” strings, as they commonly occur in practice:

```
survey <- tibble(
  id = 1:8,
  response = c("Yes", " Yes", "yes ", " YES ", "No", "no", "NO ", "maybe"),
  comment = c(
    "All good",
    "  Leading whitespace",
    "Trailing whitespace   ",
    "   Both   ",
    "Too   many   spaces",
    NA,
    "",
    "Contains number: 42"
  ),
  category = c("Cat_A", "Cat_B", "Cat_A", "CAT_C", "cat_a", "Cat-B", "Cat A",
"Cat_C")
)

survey
```

```
# A tibble: 8 × 4
     id response comment                  category
  <int> <chr>    <chr>                    <chr>
```

```
1     1 "Yes"    "All good"               Cat_A
2     2 " Yes"   "  Leading whitespace"   Cat_B
3     3 "yes "   "Trailing whitespace   " Cat_A
4     4 " YES "  "   Both   "             CAT_C
5     5 "No"     "Too   many   spaces"    cat_a
6     6 "no"      <NA>                    Cat-B
7     7 "NO "    ""                       Cat A
8     8 "maybe"  "Contains number: 42"    Cat_C
```

We can see typical problems: inconsistent capitalization, leading/trailing whitespace, different spellings of the same category.

Base R: paste() and paste0()

The functions `paste()` and `paste0()` are built into R and serve to concatenate strings.

Basic Principle

```r
# paste() joins with space (default)
paste("Hello", "World")
```

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

```r
# paste0() joins without separator
paste0("Hello", "World")
```

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

```r
# With variables
name <- "Anna"
age <- 28
paste("Name:", name, "- Age:", age)
```

```
[1] "Name: Anna - Age: 28"
```

The sep Argument

With `sep`, we can specify the separator between elements:

```r
paste("2024", "01", "15", sep = "-")
```

```
[1] "2024-01-15"
```

```r
paste("A", "B", "C", sep = "_")
```

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

```r
paste("One", "Two", "Three", sep = " | ")
```

```
[1] "One | Two | Three"
```

The collapse Argument

When we want to combine a vector into a single string:

```r
cities <- c("Berlin", "Hamburg", "Munich")

# Without collapse: vector with 3 elements
paste("City:", cities)
```

```
[1] "City: Berlin"  "City: Hamburg" "City: Munich"
```

```r
# With collapse: a single string
paste(cities, collapse = ", ")
```

```
[1] "Berlin, Hamburg, Munich"
```

```r
paste(cities, collapse = " and ")
```

```
[1] "Berlin and Hamburg and Munich"
```

Limitation

With more complex strings, `paste()` quickly becomes unwieldy:

```
abbrev <- "Ei"
date <- "2024-01-15"
version <- 2

# Hard to read
paste0("Report_", abbrev, "_", date, "_v", version, ".xlsx")
```

```
[1] "Report_Ei_2024-01-15_v2.xlsx"
```

This is where `glue()` offers a more elegant solution.

💡 Exercise: paste() and paste0()

a) Create the string `"R-Workshop-2024"` from the three parts “R”, “Workshop”, and “2024” using `paste()`.

b) Given the vector `months <- c("Jan", "Feb", "Mar")`. Create the string `"Jan, Feb, Mar"` from it.

ℹ Solution

```
# a) With hyphen as separator
paste("R", "Workshop", "2024", sep = "-")
```

```
[1] "R-Workshop-2024"
```

```
# b) Combine vector with collapse
months <- c("Jan", "Feb", "Mar")
paste(months, collapse = ", ")
```

```
[1] "Jan, Feb, Mar"
```

glue: Elegant String Interpolation

The {glue} package allows embedding variables directly in strings – with curly braces `{}`.

Basic Principle

```
name <- "Anna"
age <- 28

glue("My name is {name} and I am {age} years old.")
```

```
My name is Anna and I am 28 years old.
```

The code is much more readable than the corresponding `paste()` version.

Practical Example: Creating File Names

A common use case is creating file names:

```
abbrev <- "Ei"
date <- Sys.Date()
version <- 2

# Elegant and readable
filename <- glue("Report_{abbrev}_{date}_v{version}.xlsx")
filename
```

```
Report_Ei_2026-02-08_v2.xlsx
```

Expressions in glue

You can also use R expressions directly within the braces:

```
x <- 10
glue("The double of {x} is {x * 2}.")
```

```
The double of 10 is 20.
```

```
glue("Today is {format(Sys.Date(), '%Y-%m-%d')}.")
```

```
Today is 2026-02-08.
```

glue_data() for Tibbles

With `glue_data()`, we can access columns of a tibble row by row:

```
people <- tibble(
  first_name = c("Anna", "Ben", "Clara"),
  last_name = c("Miller", "Smith", "Weber"),
  points = c(85, 92, 78)
)

people %>%
  mutate(description = glue_data(., "{first_name} {last_name}: {points} points"))
```

```
# A tibble: 3 × 4
  first_name last_name points description
  <chr>      <chr>      <dbl> <glue>
1 Anna       Miller        85 Anna Miller: 85 points
```

```
2 Ben        Smith         92 Ben Smith: 92 points
3 Clara      Weber         78 Clara Weber: 78 points
```

Comparison: paste0() vs glue()

```r
# paste0: Variables interrupt the string
paste0("Result_", name, "_", date, "_final.csv")

# glue: Flows smoothly
glue("Result_{name}_{date}_final.csv")
```

Both produce the same result, but `glue()` is much clearer with complex strings.

Exercise: glue()

Given the variables:

```r
project <- "Analysis"
year <- 2024
month <- "March"
```

a) Create the string `"Project: Analysis (March 2024)"` using `glue()`.

b) Create the filename `"Analysis_2024_March_report.pdf"`.

Solution

```r
# a) Description text
glue("Project: {project} ({month} {year})")
```

```
Project: Analysis (March 2024)
```

```r
# b) Filename
glue("{project}_{year}_{month}_report.pdf")
```

```
Analysis_2024_March_report.pdf
```

stringr: Manipulating Strings

The {stringr} package (part of the tidyverse) provides consistent functions for string manipulation. All functions start with `str_`, which makes autocomplete easier.

Removing Whitespace

```
# str_trim: Remove whitespace at start/end
str_trim("  Hello World  ")
```

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

```
str_trim("  Hello World  ", side = "left")   # Only left
```

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

```
str_trim("  Hello World  ", side = "right")  # Only right
```

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

```
# str_squish: Additionally reduce multiple spaces within text
str_squish("  Too   many   spaces  ")
```

```
[1] "Too many spaces"
```

Application to our dataset:

```
survey %>%
  mutate(
    response_clean = str_trim(response),
    comment_clean = str_squish(comment)
  ) %>%
  select(response, response_clean, comment, comment_clean)
```

```
# A tibble: 8 × 4
  response response_clean comment                  comment_clean
  <chr>    <chr>          <chr>                    <chr>
1 "Yes"    Yes            "All good"               "All good"
2 " Yes"   Yes            "  Leading whitespace"   "Leading whitespace"
3 "yes "   yes            "Trailing whitespace   " "Trailing whitespace"
4 " YES "  YES            "   Both   "             "Both"
5 "No"     No             "Too   many   spaces"    "Too many spaces"
6 "no"     no              <NA>                     <NA>
7 "NO "    NO             ""                       ""
8 "maybe"  maybe          "Contains number: 42"    "Contains number: 42"
```

Changing Case

```
text <- "HeLLo WoRLD"

str_to_lower(text)   # all lowercase
```

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

```
str_to_upper(text)   # ALL UPPERCASE
```

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

```
str_to_title(text)   # First Letter Of Each Word Uppercase
```

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

```
str_to_sentence(text) # Only first letter of sentence uppercase
```

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

Application: Standardizing responses:

```
survey %>%
  mutate(response_standard = str_to_lower(str_trim(response))) %>%
  select(response, response_standard)
```

```
# A tibble: 8 × 2
  response response_standard
  <chr>    <chr>
1 "Yes"    yes
2 " Yes"   yes
3 "yes "   yes
4 " YES "  yes
5 "No"     no
6 "no"     no
7 "NO "    no
8 "maybe"  maybe
```

Searching with str_detect()

`str_detect()` checks if a pattern occurs in a string (returns TRUE/FALSE):

```
# Single strings
str_detect("Hello World", "World")
```

```
[1] TRUE
```

```
str_detect("Hello World", "world")  # Case-sensitive!
```

```
[1] FALSE
```

```
# Apply to vector/column
survey %>%
  filter(str_detect(comment, "whitespace"))
```

```
# A tibble: 2 × 4
     id response comment                  category
  <int> <chr>    <chr>                    <chr>
1     2 " Yes"   "  Leading whitespace"   Cat_B
2     3 "yes "   "Trailing whitespace   " Cat_A
```

Replacing with str_replace()

```
# Replace first occurrence
str_replace("Cat_A and Cat_B", "_", "-")
```

```
[1] "Cat-A and Cat_B"
```

```
# Replace all occurrences
str_replace_all("Cat_A and Cat_B", "_", "-")
```

```
[1] "Cat-A and Cat-B"
```

Application: Standardizing categories:

```
survey %>%
  mutate(
    category_clean = category %>%
      str_to_lower() %>%           # All lowercase
      str_replace_all("-", "_") %>% # Hyphens to underscores
      str_replace_all(" ", "_")     # Spaces to underscores
  ) %>%
  select(category, category_clean)
```

```
# A tibble: 8 × 2
  category category_clean
  <chr>    <chr>
1 Cat_A    cat_a
2 Cat_B    cat_b
3 Cat_A    cat_a
4 CAT_C    cat_c
5 cat_a    cat_a
6 Cat-B    cat_b
7 Cat A    cat_a
8 Cat_C    cat_c
```

Extracting with str_extract()

```
# Extract first occurrence
str_extract("Contains number: 42 and 99", "\\d+")
```

```
[1] "42"
```

```
# Extract all occurrences
str_extract_all("Contains number: 42 and 99", "\\d+")
```

```
[[1]]
[1] "42" "99"
```

Substrings with str_sub()

```
text <- "ABCDEFGH"

str_sub(text, 1, 3)    # Characters 1-3
```

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

```
str_sub(text, -3, -1)  # Last 3 characters
```

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

```
str_sub(text, 3)       # From character 3 to end
```

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

Other Useful Functions

```
# Length of a string
str_length("Hello")
```

```
[1] 5
```

```
# Concatenate strings (alternative to paste)
str_c("A", "B", "C", sep = "-")
```

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

```r
# Pad with zeros (e.g., for IDs)
str_pad(1:5, width = 3, pad = "0")
```

```
[1] "001" "002" "003" "004" "005"
```

```r
# Split string
str_split("A,B,C", ",")
```

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

Exercise: stringr

Use the `survey` dataset:

a) Clean the `response` column: Remove whitespace and convert everything to lowercase. Save the result as a new column `response_clean`.

b) Count how many rows in `comment` contain the word “whitespace”.

c) Create a new column `id_formatted` from the `id` column in the format “ID-001”, “ID-002”, etc.

Solution

```r
# a) Clean responses
survey %>%
  mutate(response_clean = str_to_lower(str_trim(response))) %>%
  select(response, response_clean)
```

```
# A tibble: 8 × 2
  response response_clean
  <chr>    <chr>
1 "Yes"    yes
2 " Yes"   yes
3 "yes "   yes
4 " YES "  yes
5 "No"     no
6 "no"     no
7 "NO "    no
8 "maybe"  maybe
```

```r
# b) Count rows with "whitespace"
survey %>%
  filter(str_detect(comment, "whitespace")) %>%
  nrow()
```

```
[1] 2
```

```r
# c) Format IDs
survey %>%
  mutate(id_formatted = glue("ID-{str_pad(id, width = 3, pad = '0')}")) %>%
  select(id, id_formatted)
```

```
# A tibble: 8 × 2
     id id_formatted
  <int> <glue>
1     1 ID-001
2     2 ID-002
3     3 ID-003
4     4 ID-004
5     5 ID-005
6     6 ID-006
7     7 ID-007
8     8 ID-008
```

Formatting Numbers

When creating reports and tables, numbers often need to be formatted attractively: percentages with % signs, thousands separators, rounded decimal places, or correctly formatted p-values. R offers various tools for this.

Base R: round() vs. format()

A common stumbling block is the difference between `round()` and `format()`:

```r
numbers <- c(1.5, 2.0, 3.456, 10.1)

# round(): Rounds mathematically, but removes trailing zeros
round(numbers, 2)
```

```
[1]  1.50  2.00  3.46 10.10
```

```r
# format(): Keeps trailing zeros, but returns strings
format(round(numbers, 2), nsmall = 2)
```

```
[1] " 1.50" " 2.00" " 3.46" "10.10"
```

`round()` returns numbers (1.5 becomes 1.5, not 1.50), while `format()` produces strings with a constant number of decimal places.

scales: Formatting for Reports

The {scales} package offers specialized functions for common formatting tasks:

Percentages

```r
proportions <- c(0.1, 0.255, 0.5, 1)

# Simple percentage formatting
percent(proportions)
```

```
[1] "10%"  "26%"  "50%"  "100%"
```

```r
# With precision
percent(proportions, accuracy = 0.1)
```

```
[1] "10.0%"  "25.5%"  "50.0%"  "100.0%"
```

```r
# European decimal separator
percent(proportions, accuracy = 0.1, decimal.mark = ",")
```

```
[1] "10,0%"  "25,5%"  "50,0%"  "100,0%"
```

Thousands Separators

```r
large_numbers <- c(1234, 56789, 1234567)

# English (comma as thousands separator)
comma(large_numbers)
```

```
[1] "1,234"     "56,789"    "1,234,567"
```

```r
# European (period as thousands separator)
number(large_numbers, big.mark = ".")
```

```
Warning in prettyNum(.Internal(format(x, trim, digits, nsmall, width, 3L, :
'big.mark' und 'decimal.mark' sind beide '.', was verwirrend sein könnte
```

```
[1] "1.234"     "56.789"    "1.234.567"
```

General Number Formatting

```
values <- c(1.2345, 67.891, 0.0052)

# Fixed decimal places
number(values, accuracy = 0.01)
```

```
[1] "1.23"  "67.89" "0.01"
```

```
# With prefix/suffix
number(values, accuracy = 0.01, suffix = " kg")
```

```
[1] "1.23 kg"  "67.89 kg" "0.01 kg"
```

```
number(large_numbers, prefix = "€ ", big.mark = ".")
```

```
Warning in prettyNum(.Internal(format(x, trim, digits, nsmall, width, 3L, :
'big.mark' und 'decimal.mark' sind beide '.', was verwirrend sein könnte
```

```
[1] "€ 1.234"     "€ 56.789"    "€ 1.234.567"
```

P-Values

```
p_values <- c(0.5, 0.05, 0.001, 0.00001)

# Automatic formatting
pvalue(p_values)
```

```
[1] "0.500"  "0.050"  "0.001"  "<0.001"
```

```
# With precision
pvalue(p_values, accuracy = 0.001)
```

```
[1] "0.500"  "0.050"  "0.001"  "<0.001"
```

i Additional Formatting Functions

For complex formatting, base R also offers `sprintf()` with C-style syntax (e.g., `sprintf("%.2f", 3.14159)` for two decimal places). The syntax is powerful but cryptic – for most use cases, the {scales} functions are more readable.

Exercise: Formatting Numbers

Given the following values:

```
revenue <- c(12500, 8900, 156000)
proportions <- c(0.125, 0.089, 0.786)
p <- 0.0234
```

a) Format `revenue` with thousands separators (periods) and the suffix ” €“.

b) Format `proportions` as percentages with one decimal place.

c) Format the p-value `p` using `pvalue()`.

Solution

```
# a) Format revenue
number(revenue, big.mark = ".", suffix = " €")
```

```
Warning in prettyNum(.Internal(format(x, trim, digits, nsmall, width, 3L, :
'big.mark' und 'decimal.mark' sind beide '.', was verwirrend sein könnte
```

```
[1] "12.500 €"  "8.900 €"   "156.000 €"
```

```
# b) Proportions as percent
percent(proportions, accuracy = 0.1)
```

```
[1] "12.5%" "8.9%"  "78.6%"
```

```
# c) p-value
pvalue(p)
```

```
[1] "0.023"
```

Outlook: Smart Rounding with BioMathR

A common problem with rounding: How many decimal places are sensible? The `round_smart()` function from the {BioMathR} package solves this elegantly. It rounds so that results have as few digits as possible, but as many as necessary:

```
# Installation from GitHub
# remotes::install_github("SchmidtPaul/BioMathR")

library(BioMathR)

# Different numbers, automatically sensibly rounded
round_smart(c(1.0001234, 0.0012345, 123.456))
# Result: 1.0001, 0.001, 123.5

# Apply to entire columns
data %>%
 mutate(across(where(is.numeric), round_smart))
```

The special feature: `round_smart()` never changes the part before the decimal point and allows a maximum number of decimal places. Details at github.com/SchmidtPaul/BioMathR.

Outlook: Regular Expressions

Regular Expressions (Regex) are a powerful language for pattern description in strings. We already used `\\d+` above to extract numbers.

A Mini Example

```r
texts <- c(
  "Order No. 12345",
  "Customer: Max Mustermann",
  "Amount: 99.50 EUR",
  "Date: 15.01.2024"
)

# Extract all numbers
str_extract_all(texts, "\\d+")
```

```
[[1]]
[1] "12345"

[[2]]
character(0)

[[3]]
[1] "99" "50"

[[4]]
[1] "15"   "01"   "2024"
```

```r
# Only numbers with decimal point
str_extract(texts, "\\d+\\.\\d+")
```

```
[1] NA      NA      "99.50" "15.01"
```

```r
# Email-like pattern (simplified)
email_text <- "Contact: info@example.com or support@test.de"
str_extract_all(email_text, "[a-z]+@[a-z]+\\.[a-z]+")
```

```
[[1]]
[1] "info@example.com" "support@test.de"
```

Important Regex Building Blocks

Pattern	Meaning
`\\d`	A digit (0-9)
`\\w`	A “word character” (letter, digit, underscore)
`\\s`	A whitespace (space, tab, newline)
`.`	Any character
`+`	One or more of the previous
`*`	Zero or more of the previous
`?`	Zero or one of the previous
`[abc]`	One of the characters a, b, or c

Pattern	Meaning
`^`	Start of string
`$`	End of string

i Learning Regex

Regular expressions have a steep learning curve but are extremely powerful. Good resources:

- regex101.com – Interactive regex tester
- R for Data Science: Strings – Chapter on strings and regex
- `?regex` in R for the documentation

Outlook: epoxy

The {epoxy} package extends the idea of {glue} for dynamic documents in Quarto and RMarkdown. It enables elegant inline formatting of numbers and text directly in prose.

```r
# Installation
install.packages("epoxy")

# In Quarto: Automatically format numbers
# ```{epoxy}
# The analysis includes {nrow(data)} observations with an
# average of {mean(data$value):.2f}.
# ```
```

For recurring reports where numbers in prose need to be updated, {epoxy} is very practical. See epoxy documentation.

Summary

In this chapter, we learned the most important tools for working with strings in R.

i Key Takeaways

Comparison of Concatenation Methods:

Function	Package	Strength
`paste()` / `paste0()`	base R	Always available, sep/collapse
`glue()`	glue	Readability with many variables
`str_c()`	stringr	Consistent with stringr ecosystem

Key stringr Functions for Data Cleaning:

Function	Purpose
`str_trim()`	Remove whitespace at edges
`str_squish()`	1. reduce multiple spaces
`str_to_lower()`	Convert to lowercase
`str_detect()`	Search for pattern (TRUE/FALSE)
`str_replace_all()`	Replace pattern
`str_extract()`	Extract pattern
`str_pad()`	Pad with characters

Formatting Numbers:

Function	Package	Purpose
`percent()`	scales	Percentages (10%)
`comma()` / `number()`	scales	Thousands separators, decimals
`pvalue()`	scales	p-values
`round_smart()`	BioMathR	Smart rounding (as few as possible, as many as necessary)

Typical Cleaning Workflow:

```r
data %>%
  mutate(
    column_clean = column %>%
      str_trim() %>%           # Remove whitespace
      str_to_lower() %>%       # Lowercase
      str_replace_all(" ", "_") # Replace spaces
  )
```

Further Resources:

- stringr Documentation
- glue Documentation
- scales Documentation
- BioMathR on GitHub
- R for Data Science: Strings
- epoxy for Dynamic Documents

Bibliography