4. Strings und Text

Textmanipulation mit paste, glue, stringr und Zahlenformatierung

Dr. Paul Schmidt

Um alle in diesem Kapitel verwendeten Pakete zu installieren und zu laden, führt man folgenden Code aus:

```
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)
```

Einleitung

In der Datenanalyse arbeiten wir ständig mit Text: Dateinamen zusammensetzen, Spaltennamen bereinigen, Kategorien vereinheitlichen, Labels für Grafiken erstellen. Auch die Formatierung von Zahlen für Berichte und Tabellen gehört dazu – Prozente, Tausendertrennzeichen, p-Werte.

R bietet verschiedene Werkzeuge dafür – von den eingebauten Funktionen `paste()` und `paste0()` über das elegante {glue}-Paket, die mächtigen Manipulationsfunktionen aus {stringr} bis hin zu spezialisierten Formatierungsfunktionen aus {scales}.

Dieses Kapitel zeigt die wichtigsten Techniken für typische Data-Cleaning-Aufgaben und die Formatierung von Werten für Berichte.

Beispieldaten

Für dieses Kapitel erstellen wir einen kleinen Datensatz mit typischen “dreckigen” Strings, wie sie in der Praxis häufig vorkommen:

```
umfrage <- tibble(
  id = 1:8,
  antwort = c("Ja", " Ja", "ja ", " JA ", "Nein", "nein", "NEIN ", "vielleicht"),
  kommentar = c(
    "Alles gut",
    "  Leerzeichen am Anfang",
    "Leerzeichen am Ende   ",
    "   Beides   ",
    "Zu   viele   Leerzeichen",
    NA,
    "",
    "Enthält Zahl: 42"
  ),
  kategorie = c("Kat_A", "Kat_B", "Kat_A", "KAT_C", "kat_a", "Kat-B", "Kat A",
"Kat_C")
)

umfrage
```

```
# A tibble: 8 × 4
     id antwort      kommentar                 kategorie
  <int> <chr>        <chr>                     <chr>
1     1 "Ja"         "Alles gut"               Kat_A
2     2 " Ja"        "  Leerzeichen am Anfang" Kat_B
3     3 "ja "        "Leerzeichen am Ende   "  Kat_A
4     4 " JA "       "   Beides   "            KAT_C
5     5 "Nein"       "Zu   viele   Leerzeichen" kat_a
6     6 "nein"        <NA>                     Kat-B
7     7 "NEIN "      ""                        Kat A
8     8 "vielleicht" "Enthält Zahl: 42"        Kat_C
```

Man sieht typische Probleme: inkonsistente Groß-/Kleinschreibung, führende/nachfolgende Leerzeichen, unterschiedliche Schreibweisen derselben Kategorie.

Base R: paste() und paste0()

Die Funktionen `paste()` und `paste0()` sind in R eingebaut und dienen dazu, Strings zusammenzufügen.

Grundprinzip

```
# paste() fügt mit Leerzeichen zusammen (Standard)
paste("Hallo", "Welt")
```

```
[1] "Hallo Welt"
```

```
# paste0() fügt ohne Trennzeichen zusammen
paste0("Hallo", "Welt")
```

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

```
# Mit Variablen
name <- "Anna"
alter <- 28
paste("Name:", name, "- Alter:", alter)
```

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

Das sep-Argument

Mit `sep` können wir das Trennzeichen zwischen den Elementen festlegen:

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

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

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

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

```
paste("Eins", "Zwei", "Drei", sep = " | ")
```

```
[1] "Eins | Zwei | Drei"
```

Das collapse-Argument

Wenn wir einen Vektor zu einem einzigen String zusammenfügen wollen:

```
staedte <- c("Berlin", "Hamburg", "München")

# Ohne collapse: Vektor mit 3 Elementen
paste("Stadt:", staedte)
```

```
[1] "Stadt: Berlin"  "Stadt: Hamburg" "Stadt: München"
```

```
# Mit collapse: Ein einziger String
paste(staedte, collapse = ", ")
```

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

```
paste(staedte, collapse = " und ")
```

```
[1] "Berlin und Hamburg und München"
```

Limitierung

Bei komplexeren Strings wird `paste()` schnell unübersichtlich:

```
kuerzel <- "Ei"
datum <- "2024-01-15"
version <- 2

# Schwer lesbar
paste0("Tabellenband_", kuerzel, "_", datum, "_v", version, ".xlsx")
```

```
[1] "Tabellenband_Ei_2024-01-15_v2.xlsx"
```

Hier bietet `glue()` eine elegantere Lösung.

Übung: paste() und paste0()

a) Erstelle mit `paste()` den String `"R-Workshop-2024"` aus den drei Teilen “R”, “Workshop” und “2024”.

b) Gegeben ist der Vektor `monate <- c("Jan", "Feb", "Mär")`. Erstelle daraus den String `"Jan, Feb, Mär"`.

Lösungsvorschlag

```
# a) Mit Bindestrich als Trennzeichen
paste("R", "Workshop", "2024", sep = "-")
```

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

```
# b) Vektor mit collapse zusammenfügen
monate <- c("Jan", "Feb", "Mär")
paste(monate, collapse = ", ")
```

```
[1] "Jan, Feb, Mär"
```

glue: Elegante String-Interpolation

Das {glue}-Paket ermöglicht es, Variablen direkt in Strings einzubetten – mit geschweiften Klammern `{}`.

Grundprinzip

```
name <- "Anna"
alter <- 28

glue("Mein Name ist {name} und ich bin {alter} Jahre alt.")
```

```
Mein Name ist Anna und ich bin 28 Jahre alt.
```

Der Code ist viel lesbarer als die entsprechende `paste()`-Version.

Praxisbeispiel: Dateinamen erzeugen

Ein häufiger Anwendungsfall ist das Erstellen von Dateinamen:

```
kuerzel <- "Ei"
datum <- Sys.Date()
version <- 2

# Elegant und lesbar
dateiname <- glue("Tabellenband_{kuerzel}_{datum}_v{version}.xlsx")
dateiname
```

```
Tabellenband_Ei_2026-02-08_v2.xlsx
```

Ausdrücke in glue

Man kann auch R-Ausdrücke direkt in den Klammern verwenden:

```
x <- 10
glue("Das Doppelte von {x} ist {x * 2}.")
```

```
Das Doppelte von 10 ist 20.
```

```
glue("Heute ist {format(Sys.Date(), '%d.%m.%Y')}.")
```

```
Heute ist 08.02.2026.
```

glue_data() für Tibbles

Mit `glue_data()` können wir zeilenweise auf Spalten eines Tibbles zugreifen:

```
personen <- tibble(
  vorname = c("Anna", "Ben", "Clara"),
  nachname = c("Müller", "Schmidt", "Weber"),
  punkte = c(85, 92, 78)
)

personen %>%
  mutate(beschreibung = glue_data(., "{vorname} {nachname}: {punkte} Punkte"))
```

```
# A tibble: 3 × 4
  vorname nachname punkte beschreibung
  <chr>   <chr>     <dbl> <glue>
1 Anna    Müller       85 Anna Müller: 85 Punkte
```

```
2 Ben     Schmidt      92 Ben Schmidt: 92 Punkte
3 Clara   Weber        78 Clara Weber: 78 Punkte
```

Vergleich: paste0() vs glue()

```r
# paste0: Variablen unterbrechen den String
paste0("Ergebnis_", name, "_", datum, "_final.csv")

# glue: Flüssig lesbar
glue("Ergebnis_{name}_{datum}_final.csv")
```

Beide produzieren dasselbe Ergebnis, aber `glue()` ist bei komplexeren Strings deutlich übersichtlicher.

Übung: glue()

Gegeben sind die Variablen:

```r
projekt <- "Analyse"
jahr <- 2024
monat <- "März"
```

a) Erstelle mit `glue()` den String `"Projekt: Analyse (März 2024)"`.

b) Erstelle den Dateinamen `"Analyse_2024_März_report.pdf"`.

Lösungsvorschlag

```r
# a) Beschreibungstext
glue("Projekt: {projekt} ({monat} {jahr})")
```

```
Projekt: Analyse (März 2024)
```

```r
# b) Dateiname
glue("{projekt}_{jahr}_{monat}_report.pdf")
```

```
Analyse_2024_März_report.pdf
```

stringr: Strings manipulieren

Das {stringr}-Paket (Teil des tidyverse) bietet konsistente Funktionen zur String-Manipulation. Alle Funktionen beginnen mit `str_`, was die Autovervollständigung erleichtert.

Leerzeichen entfernen

```
# str_trim: Leerzeichen am Anfang/Ende entfernen
str_trim("  Hallo Welt  ")
```

```
[1] "Hallo Welt"
```

```
str_trim("  Hallo Welt  ", side = "left")   # Nur links
```

```
[1] "Hallo Welt  "
```

```
str_trim("  Hallo Welt  ", side = "right")  # Nur rechts
```

```
[1] "  Hallo Welt"
```

```
# str_squish: Zusätzlich mehrfache Leerzeichen im Text reduzieren
str_squish("  Zu   viele   Leerzeichen  ")
```

```
[1] "Zu viele Leerzeichen"
```

Anwendung auf unseren Datensatz:

```
umfrage %>%
  mutate(
    antwort_clean = str_trim(antwort),
    kommentar_clean = str_squish(kommentar)
  ) %>%
  select(antwort, antwort_clean, kommentar, kommentar_clean)
```

```
# A tibble: 8 × 4
  antwort      antwort_clean kommentar                 kommentar_clean
  <chr>        <chr>         <chr>                     <chr>
1 "Ja"         Ja            "Alles gut"               "Alles gut"
2 " Ja"        Ja            "  Leerzeichen am Anfang" "Leerzeichen am Anfang"
3 "ja "        ja            "Leerzeichen am Ende   "  "Leerzeichen am Ende"
4 " JA "       JA            "   Beides   "            "Beides"
5 "Nein"       Nein          "Zu   viele   Leerzeichen" "Zu viele Leerzeichen"
6 "nein"       nein           <NA>                      <NA>
7 "NEIN "      NEIN          ""                        ""
8 "vielleicht" vielleicht    "Enthält Zahl: 42"        "Enthält Zahl: 42"
```

Groß- und Kleinschreibung

```
text <- "HaLLo WeLT"

str_to_lower(text)   # alles klein
```

```
[1] "hallo welt"
```

```
str_to_upper(text)   # ALLES GROSS
```

```
[1] "HALLO WELT"
```

```
str_to_title(text)   # Erster Buchstabe Jedes Wortes Groß
```

```
[1] "Hallo Welt"
```

```
str_to_sentence(text) # Nur erster Buchstabe des Satzes groß
```

```
[1] "Hallo welt"
```

Anwendung: Antworten vereinheitlichen:

```
umfrage %>%
  mutate(antwort_standard = str_to_lower(str_trim(antwort))) %>%
  select(antwort, antwort_standard)
```

```
# A tibble: 8 × 2
  antwort      antwort_standard
  <chr>        <chr>
1 "Ja"         ja
2 " Ja"        ja
3 "ja "        ja
4 " JA "       ja
5 "Nein"       nein
6 "nein"       nein
7 "NEIN "      nein
8 "vielleicht" vielleicht
```

Suchen mit str_detect()

`str_detect()` prüft, ob ein Muster im String vorkommt (gibt TRUE/FALSE zurück):

```
# Einzelne Strings
str_detect("Hallo Welt", "Welt")
```

```
[1] TRUE
```

```
str_detect("Hallo Welt", "welt")  # Case-sensitive!
```

```
[1] FALSE
```

```
# Auf Vektor/Spalte anwenden
umfrage %>%
  filter(str_detect(kommentar, "Leerzeichen"))
```

```
# A tibble: 3 × 4
     id antwort kommentar                   kategorie
  <int> <chr>   <chr>                       <chr>
1     2 " Ja"   "  Leerzeichen am Anfang"   Kat_B
2     3 "ja "   "Leerzeichen am Ende   "    Kat_A
3     5 "Nein"  "Zu   viele   Leerzeichen"  kat_a
```

Ersetzen mit str_replace()

```
# Erstes Vorkommen ersetzen
str_replace("Kat_A und Kat_B", "_", "-")
```

```
[1] "Kat-A und Kat_B"
```

```
# Alle Vorkommen ersetzen
str_replace_all("Kat_A und Kat_B", "_", "-")
```

```
[1] "Kat-A und Kat-B"
```

Anwendung: Kategorien vereinheitlichen:

```
umfrage %>%
  mutate(
    kategorie_clean = kategorie %>%
      str_to_lower() %>%           # Alles klein
      str_replace_all("-", "_") %>% # Bindestriche zu Unterstrichen
      str_replace_all(" ", "_")     # Leerzeichen zu Unterstrichen
  ) %>%
  select(kategorie, kategorie_clean)
```

```
# A tibble: 8 × 2
  kategorie kategorie_clean
  <chr>     <chr>
1 Kat_A     kat_a
2 Kat_B     kat_b
3 Kat_A     kat_a
4 KAT_C     kat_c
5 kat_a     kat_a
6 Kat-B     kat_b
7 Kat A     kat_a
8 Kat_C     kat_c
```

Extrahieren mit str_extract()

```
# Erstes Vorkommen extrahieren
str_extract("Enthält Zahl: 42 und 99", "\\d+")
```

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

```
# Alle Vorkommen extrahieren
str_extract_all("Enthält Zahl: 42 und 99", "\\d+")
```

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

Teilstrings mit str_sub()

```
text <- "ABCDEFGH"

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

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

```
str_sub(text, -3, -1)   # Letzte 3 Zeichen
```

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

```
str_sub(text, 3)        # Ab Zeichen 3 bis Ende
```

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

Weitere nützliche Funktionen

```
# Länge eines Strings
str_length("Hallo")
```

```
[1] 5
```

```
# Strings zusammenfügen (Alternative zu paste)
str_c("A", "B", "C", sep = "-")
```

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

```r
# Mit Nullen auffüllen (z.B. für IDs)
str_pad(1:5, width = 3, pad = "0")
```

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

```r
# String aufteilen
str_split("A,B,C", ",")
```

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

Übung: stringr

Verwende den `umfrage`-Datensatz:

a) Bereinige die Spalte `antwort`: Entferne Leerzeichen und wandle alles in Kleinbuchstaben um. Speichere das Ergebnis als neue Spalte `antwort_clean`.

b) Zähle, wie viele Zeilen in `kommentar` das Wort “Leerzeichen” enthalten.

c) Erstelle aus der `id`-Spalte eine neue Spalte `id_formatted` im Format “ID-001”, “ID-002”, etc.

Lösungsvorschlag

```r
# a) Antworten bereinigen
umfrage %>%
  mutate(antwort_clean = str_to_lower(str_trim(antwort))) %>%
  select(antwort, antwort_clean)
```

```
# A tibble: 8 × 2
  antwort      antwort_clean
  <chr>        <chr>
1 "Ja"         ja
2 " Ja"        ja
3 "ja "        ja
4 " JA "       ja
5 "Nein"       nein
6 "nein"       nein
7 "NEIN "      nein
8 "vielleicht" vielleicht
```

```r
# b) Zeilen mit "Leerzeichen" zählen
umfrage %>%
  filter(str_detect(kommentar, "Leerzeichen")) %>%
  nrow()
```

```
[1] 3
```

```r
# c) IDs formatieren
umfrage %>%
  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
```

Zahlen formatieren

Beim Erstellen von Berichten und Tabellen müssen Zahlen oft ansprechend formatiert werden: Prozente mit %-Zeichen, Tausendertrennzeichen, gerundete Dezimalstellen oder korrekt formatierte p-Werte. R bietet dafür verschiedene Werkzeuge.

Base R: round() vs. format()

Ein häufiger Stolperstein ist der Unterschied zwischen `round()` und `format()`:

```
zahlen <- c(1.5, 2.0, 3.456, 10.1)

# round(): Rundet mathematisch, entfernt aber trailing zeros
round(zahlen, 2)
```

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

```
# format(): Behält trailing zeros, gibt aber Strings zurück
format(round(zahlen, 2), nsmall = 2)
```

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

`round()` gibt Zahlen zurück (1.5 wird zu 1.5, nicht 1.50), während `format()` Strings mit konstanter Dezimalstellenzahl erzeugt.

scales: Formatierung für Berichte

Das {scales}-Paket bietet spezialisierte Funktionen für häufige Formatierungsaufgaben:

Prozente

```
anteile <- c(0.1, 0.255, 0.5, 1)

# Einfache Prozentformatierung
percent(anteile)
```

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

```
# Mit Genauigkeit
percent(anteile, accuracy = 0.1)
```

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

```
# Deutsche Dezimaltrennung
percent(anteile, accuracy = 0.1, decimal.mark = ",")
```

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

Tausendertrennzeichen

```
grosse_zahlen <- c(1234, 56789, 1234567)

# Englisch (Komma als Tausendertrenner)
comma(grosse_zahlen)
```

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

```
# Deutsch (Punkt als Tausendertrenner)
number(grosse_zahlen, 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"
```

Allgemeine Zahlenformatierung

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

# Feste Dezimalstellen
number(werte, accuracy = 0.01)
```

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

```
# Mit Präfix/Suffix
number(werte, accuracy = 0.01, suffix = " kg")
```

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

```
number(grosse_zahlen, 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-Werte

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

# Automatische Formatierung
pvalue(p_werte)
```

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

```
# Mit Genauigkeit
pvalue(p_werte, accuracy = 0.001)
```

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

> **Weitere Formatierungsfunktionen**
>
> Für komplexe Formatierungen bietet base R auch `sprintf()` mit C-Style Syntax (z.B. `sprintf("%.2f", 3.14159)` für zwei Dezimalstellen). Die Syntax ist mächtig, aber kryptisch – für die meisten Anwendungsfälle sind die {scales}-Funktionen lesbarer.

Übung: Zahlen formatieren

Gegeben sind folgende Werte:

```
umsatz <- c(12500, 8900, 156000)
anteile <- c(0.125, 0.089, 0.786)
p <- 0.0234
```

a) Formatiere `umsatz` mit Tausenderpunkten und dem Suffix " €".

b) Formatiere `anteile` als Prozente mit einer Dezimalstelle.

c) Formatiere den p-Wert `p` mit `pvalue()`.

Lösungsvorschlag

```
# a) Umsatz formatieren
number(umsatz, 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) Anteile als Prozent
percent(anteile, accuracy = 0.1)
```

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

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

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

Ausblick: Intelligentes Runden mit BioMathR

Ein häufiges Problem beim Runden: Wie viele Dezimalstellen sind sinnvoll? Die Funktion `round_smart()` aus dem {BioMathR}-Paket löst das elegant. Sie rundet so, dass die Ergebnisse so wenige Stellen wie möglich haben, aber so viele wie nötig:

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

library(BioMathR)

# Verschiedene Zahlen, automatisch sinnvoll gerundet
round_smart(c(1.0001234, 0.0012345, 123.456))
# Ergebnis: 1.0001, 0.001, 123.5

# Auf ganze Spalten anwenden
daten %>%
 mutate(across(where(is.numeric), round_smart))
```

Das Besondere: `round_smart()` verändert nie den Teil vor dem Dezimaltrennzeichen und erlaubt eine maximale Anzahl an Nachkommastellen. Details unter github.com/SchmidtPaul/BioMathR.

Ausblick: Regular Expressions

Regular Expressions (Regex) sind eine mächtige Sprache zur Musterbeschreibung in Strings. Wir haben oben bereits `\\d+` verwendet, um Zahlen zu extrahieren.

Ein Mini-Beispiel

```
texte <- c(
  "Bestellung Nr. 12345",
  "Kunde: Max Mustermann",
  "Betrag: 99.50 EUR",
  "Datum: 15.01.2024"
)

# Alle Zahlen extrahieren
str_extract_all(texte, "\\d+")
```

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

[[2]]
character(0)

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

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

```
# Nur Zahlen mit Dezimalpunkt
str_extract(texte, "\\d+\\.\\d+")
```

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

```
# E-Mail-ähnliches Muster (vereinfacht)
email_text <- "Kontakt: info@example.com oder support@test.de"
str_extract_all(email_text, "[a-z]+@[a-z]+\\.[a-z]+")
```

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

Wichtige Regex-Bausteine

Muster	Bedeutung
`\\d`	Eine Ziffer (0-9)
`\\w`	Ein “Wort-Zeichen” (Buchstabe, Ziffer, Unterstrich)
`\\s`	Ein Whitespace (Leerzeichen, Tab, Newline)
`.`	Ein beliebiges Zeichen
`+`	Ein oder mehrere des vorherigen
`*`	Null oder mehrere des vorherigen
`?`	Null oder eines des vorherigen
`[abc]`	Eines der Zeichen a, b oder c

Muster	Bedeutung
`^`	Anfang des Strings
`$`	Ende des Strings

> **Regex lernen**
>
> Regular Expressions haben eine steile Lernkurve, sind aber extrem mächtig. Gute Ressourcen:
>
> - regex101.com – Interaktiver Regex-Tester
> - R for Data Science: Strings – Kapitel zu Strings und Regex
> - `?regex` in R für die Dokumentation

Ausblick: epoxy

Das {epoxy}-Paket erweitert die Idee von {glue} für dynamische Dokumente in Quarto und RMarkdown. Es ermöglicht elegante Inline-Formatierung von Zahlen und Text direkt im Fließtext.

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

# In Quarto: Zahlen automatisch formatieren
# ```{epoxy}
# Die Analyse umfasst {nrow(daten)} Beobachtungen mit einem
# Durchschnitt von {mean(daten$wert):.2f}.
# ```
```

Für wiederkehrende Reports, in denen Zahlen im Fließtext aktualisiert werden müssen, ist {epoxy} sehr praktisch. Siehe epoxy Dokumentation.

Zusammenfassung

In diesem Kapitel haben wir die wichtigsten Werkzeuge für die Arbeit mit Strings in R kennengelernt.

ℹ Wichtige Erkenntnisse

Vergleich der Methoden zum Zusammenfügen:

Funktion	Paket	Stärke
`paste()` / `paste0()`	base R	Immer verfügbar, sep/collapse
`glue()`	glue	Lesbarkeit bei vielen Variablen
`str_c()`	stringr	Konsistent mit stringr-Ökosystem

Die wichtigsten stringr-Funktionen für Data Cleaning:

Funktion	Zweck
`str_trim()`	Leerzeichen am Rand entfernen
`str_squish()`	1. mehrfache Leerzeichen reduzieren
`str_to_lower()`	Alles kleinschreiben
`str_detect()`	Muster suchen (TRUE/FALSE)
`str_replace_all()`	Muster ersetzen
`str_extract()`	Muster extrahieren
`str_pad()`	Mit Zeichen auffüllen

Zahlen formatieren:

Funktion	Paket	Zweck
`percent()`	scales	Prozente (10%)
`comma()` / `number()`	scales	Tausendertrenner, Dezimalstellen
`pvalue()`	scales	p-Werte
`round_smart()`	BioMathR	Intelligentes Runden (so wenig wie möglich, so viel wie nötig)

Typischer Cleaning-Workflow:

```
daten %>%
  mutate(
    spalte_clean = spalte %>%
      str_trim() %>%            # Leerzeichen entfernen
      str_to_lower() %>%        # Kleinschreibung
      str_replace_all(" ", "_") # Leerzeichen ersetzen
  )
```

Weiterführende Ressourcen:

- stringr Dokumentation

- glue Dokumentation
- scales Dokumentation
- BioMathR auf GitHub
- R for Data Science: Strings
- epoxy für dynamische Dokumente

Bibliography