10. Parametrisierte Reports

Ein Template, viele Varianten

Dr. Paul Schmidt

Bisher haben wir unseren Report nur für Adelie-Pinguine erstellt. Aber was, wenn wir denselben Report für alle drei Pinguinarten möchten? Statt drei separate Dokumente zu pflegen, können wir **Parameter** verwenden: Ein Template, das beim Rendern mit verschiedenen Werten gefüllt wird.

Das Konzept

Ein parametrisierter Report ist wie ein Formular mit Lücken:

1. Das Template definiert Parameter mit Standardwerten
2. Beim Rendern können andere Werte übergeben werden
3. Der Report passt sich automatisch an

Parameter definieren

Parameter werden im YAML-Header unter `params:` definiert:

```yaml
---
title: "Pinguin-Report"
format: docx
params:
  species: "Adelie"
---
```

Hier ist `species` der Parametername und `"Adelie"` der Standardwert.

Parameter im Code verwenden

Im R-Code greift man mit `params$parametername` auf Parameter zu:

```r
# Daten basierend auf Parameter filtern
selected_penguins <- penguins %>%
  filter(species == params$species) %>%
  drop_na()
```

Der Fließtext kann ebenfalls Parameter enthalten:

```
Dieser Report analysiert **Adelie**-Pinguine.
```

Ein vollständiges Beispiel

Hier ist unser Pinguin-Report als parametrisierte Version:

```yaml
---
title: "Pinguin-Report"
subtitle: "Adelie-Pinguine"
author: "Forschungsteam"
date: today
format: docx
```

```
params:
  species: "Adelie"
execute:
  echo: false
  warning: false
  message: false
---
```

```
# Einleitung

Dieser Report analysiert **Adelie**-Pinguine aus dem
Palmer Penguins Datensatz. Der Datensatz umfasst 146
Individuen dieser Art.

# Deskriptive Statistik

```{r}
#| label: tbl-params-summary
#| tbl-cap: "Zusammenfassung der Messwerte"
selected_penguins %>%
  summarise(
    N = n(),
    `Schnabellänge (mm)` = round(mean(bill_length_mm), 1),
    `Körpermasse (g)` = round(mean(body_mass_g), 0)
  ) %>%
  flextable() %>%
  autofit()
```

Visualisierung

```{r}
#| label: fig-params-scatter
#| fig-cap: "Schnabelmaße der ausgewählten Pinguinart"
#| fig-width: 5
#| fig-height: 4
ggplot(selected_penguins, aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_point(alpha = 0.6) +
  labs(
    x = "Schnabellänge (mm)",
    y = "Schnabeltiefe (mm)",
    title = glue::glue("{params$species}-Pinguine")
  ) +
  theme_minimal()
```
```

Reports mit verschiedenen Parametern rendern

In RStudio

1. Klicke auf den Pfeil neben dem Render-Button
2. Wähle “Render with Parameters…”
3. Ein Dialog öffnet sich, in dem man die Werte ändern kann

Via Kommandozeile

```
quarto render report.qmd -P species:Gentoo
```

Mehrere Parameter:

```
quarto render report.qmd -P species:Chinstrap -P year:2008
```

Programmatisch in R

```
# Einen Report rendern
quarto::quarto_render(
  input = "report.qmd",
  execute_params = list(species = "Gentoo"),
  output_file = "report_gentoo.docx"
)
```

Alle Varianten auf einmal rendern

Mit einer Schleife kann man alle Versionen automatisch erzeugen:

```
library(purrr)

# Alle Pinguinarten
species_list <- c("Adelie", "Chinstrap", "Gentoo")

# Für jede Art einen Report rendern
walk(species_list, function(sp) {
  quarto::quarto_render(
    input = "report.qmd",
    execute_params = list(species = sp),
    output_file = glue::glue("report_{tolower(sp)}.docx")
  )
})
```

Das erzeugt drei Dateien: `report_adelie.docx`, `report_chinstrap.docx`, `report_gentoo.docx`.

Mehrere Parameter

Man kann beliebig viele Parameter definieren:

```
params:
  species: "Adelie"
  island: "Biscoe"
  min_year: 2007
  include_plots: true
```

Und im Code:

```
filtered_data <- penguins %>%
  filter(
    species == params$species,
    island == params$island,
    year >= params$min_year
  )

# Bedingte Ausführung
if (params$include_plots) {
  # Plot-Code hier
}
```

Parametertypen

Parameter können verschiedene Typen haben:

```
params:
  species: "Adelie"           # Text
  sample_size: 100            # Zahl
```

```yaml
  include_plots: true          # Boolean
  islands:                     # Liste
    - Biscoe
    - Dream
```

Dynamische Titel

Der Titel kann Parameter enthalten:

```yaml
---
title: "Adelie\-Pinguin\-Analyse"
params:
  species: "Adelie"
---
```

Oder einfacher im Subtitle:

```yaml
---
title: "Pinguin-Analyse"
subtitle: "Adelie"
params:
  species: "Adelie"
---
```

Praktische Anwendungsfälle

Anwendung	Parameter
Reports pro Region	`region`, `year`
Kundenberichte	`client_name`, `client_id`
Testversionen	`include_draft_watermark: true`
Sprachversionen	`language: "de"`
Datenquellen	`data_file: "data_2024.csv"`

Tipps

Standardwerte sinnvoll setzen

Der Standardwert sollte ein typischer, funktionierender Wert sein — so kann man das Template einfach testen.

Parameterwerte validieren

Am Anfang des Dokuments prüfen, ob die Parameter gültig sind:

```r
# Prüfen ob Spezies existiert
valid_species <- c("Adelie", "Chinstrap", "Gentoo")

if (!params$species %in% valid_species) {
  stop(glue::glue(
    "Ungültige Spezies: {params$species}. ",
    "Erlaubt sind: {paste(valid_species, collapse = ', ')}"
  ))
}
```

Ausgabedateinamen

Bei der Massenproduktion sinnvolle Dateinamen verwenden:

```
output_file = glue::glue(
  "report_{params$species}_{Sys.Date()}.docx"
)
```

> Übung: Parametrisierten Report erstellen
>
> 1. Nimm den bisherigen Pinguin-Report
> 2. Füge einen `species`-Parameter mit Standardwert “Adelie” hinzu
> 3. Ersetze alle Stellen, die auf Adelie verweisen, durch `params$species`
> 4. Rendere den Report mit verschiedenen Spezies (Adelie, Chinstrap, Gentoo)
> 5. Bonus: Schreibe eine Schleife, die alle drei Reports automatisch erzeugt

Weiterführende Ressourcen

- Quarto Parameters — Offizielle Dokumentation
- Parameterized Reporting with Quarto — Rendering-Optionen

Was kommt als Nächstes

Im letzten Kapitel werfen wir einen Blick über den Word-Tellerrand: PDF-Export mit Typst, Präsentationen mit Reveal.js, und HTML-Dokumente — alles mit demselben Quarto-Wissen.

Bibliography