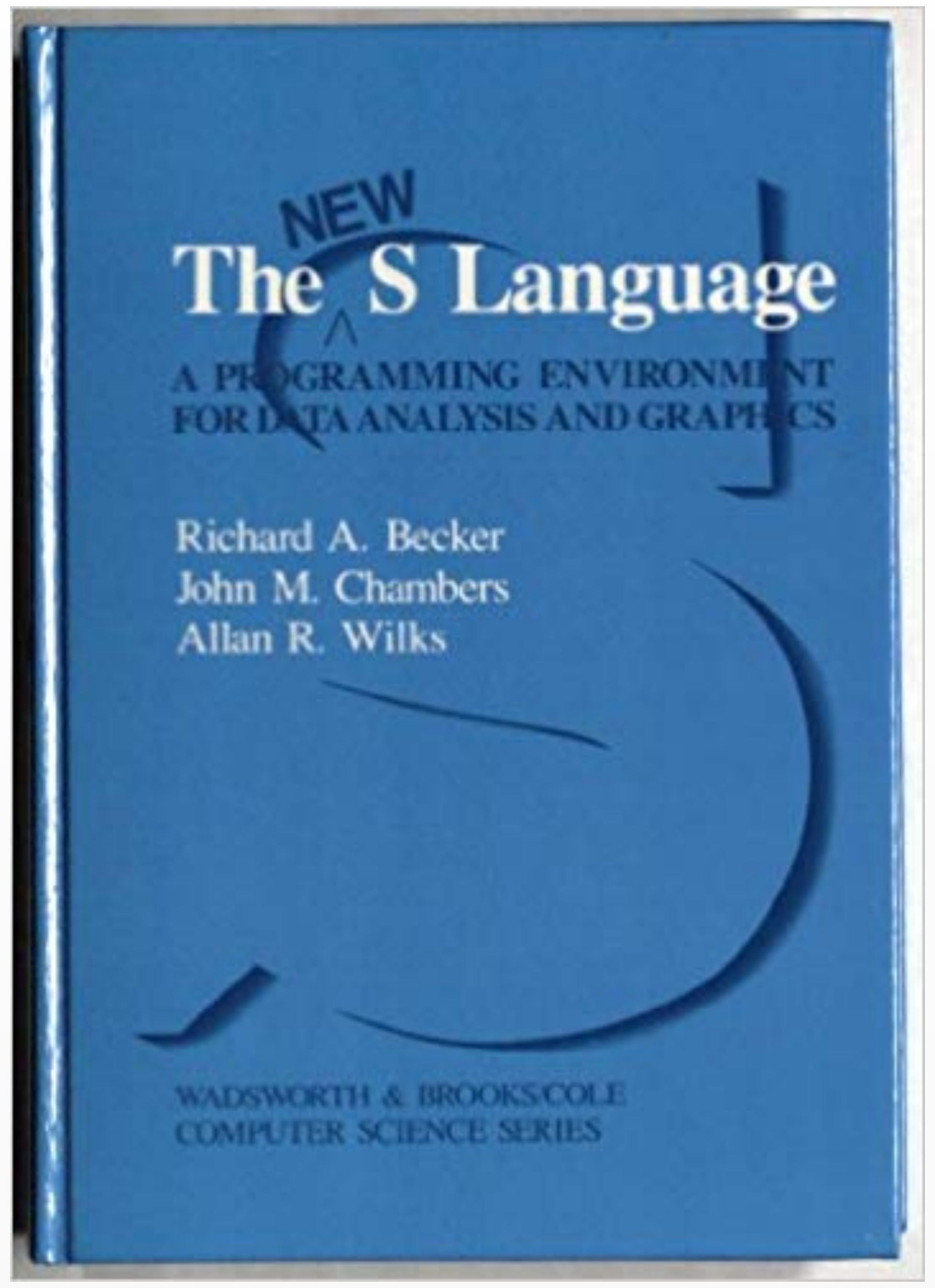
Interactivity and programming in the tidyverse

Data-masking in R

- Idea of blending data with the workspace
- Helps "turning ideas into software" (John Chambers) but hinders code reuse
- Progress in tooling and teaching

tidy eval made easy??

Data-masking in R

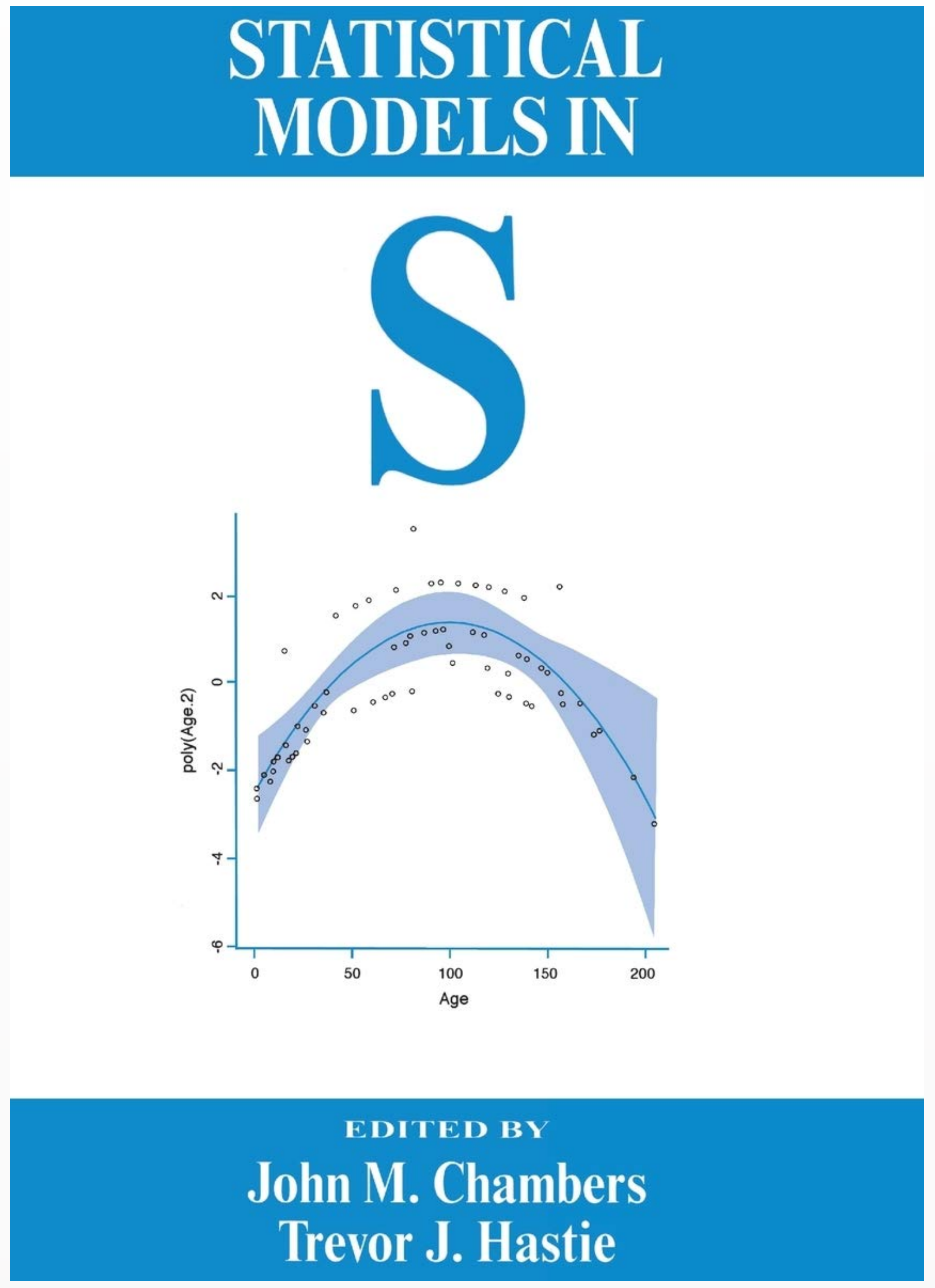


1988 — The New S Language (Bell labs)

```
attach(starwars)

mean(height, na.rm = TRUE)
#> [1] 174.358
```

Data-masking in R



1993 — Statistical Models in S

```
lm(
  birth_year ~ mass + height,
  starwars
)
```

Data-masking in R

1997 — frametools (Peter Dalgaard, R core)

```r
aq <- airquality[1:10,]
subset.frame(aq, Ozone > 20)
select.frame(aq, Ozone:Temp)
modify.frame(aq, ratio = Ozone / Temp)
```

Data-masking in R

1997 — frametools (Peter Dalgaard, R core)

```
select.frame(aq, Ozone:Temp)
```

First apparition of *selections*

R 0.62

```r
subset.frame(aq, Ozone > 20)
select.frame(aq, Ozone:Temp)
```

→

```r
subset(aq, Ozone > 20, select = Ozone:Temp)
```

```r
modify.frame(aq, ratio = Ozone / Temp)
```

→

```r
transform(aq, ratio = Ozone / Temp)
```

Data-masking in R

Few developments after inclusion of frametools

2001 — Luke Tierney

```r
bmi <- with(
  starwars,
  mass / (height / 100)^2
)
```

2007 — Peter Dalgaard

```r
starwars <- within(
  starwars,
  bmi <- mass / (height / 100)^2
)
```

Data-masking in R

2006 — data.table

Most new developments in *package space*

`dt[i, j]`

- Data-masking in `i`
- Selections in `j`

```
starwars[
  mass > 150,
  name:mass
]
```

Data-masking in R

2014 — dplyr

Most new developments in *package space*

```
airquality %>%
  filter(Ozone > 20) %>%
  select(Ozone:Temp) %>%
  mutate(ratio = Ozone / Temp)
```

Trouble in data-masking town

`?subset` `?transform`

“ This is a convenience function intended for use interactively [...]

The non-standard evaluation [...] can have unanticipated consequences. „

😱

Trouble in data-masking town

Ambiguity between data-variables
and environment-variables (workspace)

1. Unexpected masking by data-variables
2. Data-variables can't get through arguments

The tidyverse offers solutions for both issues

1. Unexpected masking

```r
n <- 100
```

```r
data.frame(x = 1) %>%
  mutate(y = x / n) %>%
  pull(y)
#> [1] 0.01
```

1. Unexpected masking

```r
n <- 100

data.frame(x = 1) %>%
  mutate(y = x / n) %>%
  pull(y)
#> [1] 0.01
```

Data frame is a *moving part*

```r
data.frame(x = 1, n = 2) %>%
  mutate(y = x / n) %>%
  pull(y)
#> [1] 0.5
```

1. Unexpected masking

Solution:
Be *explicit* in production code

```r
n <- 100
data <- data.frame(x = 1, n = 2)

data %>%
  mutate(y = .data$x / .env$n)
```

- Use the **.env** pronoun to refer to the *workspace*
- Use the **.data** pronoun to refer to the *data frame*

2. Data-variables through arguments

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by(by) %>%
    summarise(avg = mean(var))
}
```

```r
iris %>% mean_by(Species, Sepal.Width)
#> Error: Column `by` is unknown
```

2. Data-variables through arguments

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by(by) %>%
    summarise(avg = mean(var))
}
```

- env-variable **by**
- data-variable **Species**

```r
iris %>% mean_by(Species, Sepal.Width)
#> Error: Column `by` is unknown
```

2. Data-variables through arguments

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by({{ by }}) %>%
    summarise(avg = mean({{ var }}))
}
```

Tunnel the data-variable through the env-variable with the **{{ }}** operator

```r
iris %>% my_function(Species, Sepal.Width)
#>   Species      avg
#>   <fct>      <dbl>
#> 1 setosa      3.43
#> 2 versicolor  2.77
#> 3 virginica   2.97
```

2. Data-variables through arguments

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by({{ by }}) %>%
    summarise(avg = mean({{ var }}))
}
```

Tunnel the data-variable through the env-variable with the **{{ }}** operator

Hard-coded result name?

```r
iris %>% my_function(Species, Sepal.Width)
#>   Species      avg
#>   <fct>      <dbl>
#> 1 setosa      3.43
#> 2 versicolor  2.77
#> 3 virginica   2.97
```

2. Data-variables through arguments

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by({{ by }}) %>%
    summarise("{{ var }}" := mean({{ var }}))
}
```

Hard-coded result name?

Tunnel data-variable inside strings!

Variant of glue syntax

```r
iris %>% my_function(Species, Sepal.Width)
#>   Species    Sepal.Width
#>   <fct>            <dbl>
#> 1 setosa            3.43
#> 2 versicolor        2.77
#> 3 virginica         2.97
```

2. Data-variables through arguments

Tunnelling causes data-masking to propagate

```
iris %>% my_function(Species, Sepal.Width)
iris %>% my_function(.data$Species, .data$Sepal.Width)
```

Can we wrap tidyverse pipelines
without data-masking contagion?

2. Hard to reuse code in functions

```r
iris %>%
  group_by(.data$Species) %>%
  summarise(avg = mean(.data$Sepal.Width))
```

2. Hard to reuse code in functions

```
data %>%
  group_by(.data[[by]]) %>%
  summarise(avg = mean(.data[[var]]))
```

Subset `.data`
with `[[`

2. Hard to reuse code in functions

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by(.data[[by]]) %>%
    summarise(avg = mean(.data[[var]]))
}
```

Subset `.data` with `[[`

```r
iris %>% my_function("Species", "Sepal.Width")
#>   Species      avg
#>   <fct>      <dbl>
#> 1 setosa      3.43
#> 2 versicolor  2.77
#> 3 virginica   2.97
```

2. Hard to reuse code in functions

```r
mean_by <- function(data, by, var) {
  data %>%
    group_by(.data[[by]]) %>%
    summarise("{var}" := mean(.data[[var]], na.rm = TRUE))
}
```

Use single **{**
to *glue*
the string

```r
iris %>% my_function("Species", "Sepal.Width")
#>   Species    Sepal.Width
#>   <fct>            <dbl>
#> 1 setosa            3.43
#> 2 versicolor        2.77
#> 3 virginica         2.97
```

~~Trouble~~ in data-masking town

1. Unexpected masking by data-variables
    - Use **`.data`** and **`.env`** to disambiguate

2. Data-variables can't get through arguments
    - Tunnel data-variables with **`{{ }}`**
    - Subset **`.data`** with **`[[`**

What about selections?

Selections are a separate sublanguage

- Data-variables represent **locations**
- Ambiguity much less an issue

```
starwars %>% select(name:mass)
starwars %>% select(c(name, mass))
```

$\Longleftrightarrow$

```
starwars %>% select(1:3)
starwars %>% select(c(1, 3))
```

What about selections?

```r
name <- c("mass", "height")
starwars %>% select(name)
```

$\longrightarrow$ Data-variable

Use **`all_of()`** to disambiguate

```r
starwars %>% select(all_of(name))
```

$\longrightarrow$ Env-variable

```r
averages <- function(data, vars) {
  data %>%
    select(all_of(vars)) %>%
    map_dbl(mean, na.rm = TRUE)
}
```

Take *character vectors* with **`all_of()`**

```r
x <- c("Sepal.Length", "Petal.Length")
iris %>% averages(x)
#> Sepal.Length  Sepal.Width Petal.Length  Petal.Width
#>     5.843333     3.057333     3.758000     1.199333
```

```r
averages <- function(data, vars) {
  data %>%
    select({{ vars }}) %>%
    map_dbl(mean, na.rm = TRUE)
}
```

Tunnel *selections* with **{{ }}**

```r
iris %>% averages(starts_with("Sepal"))
#> Sepal.Length  Sepal.Width
#>     5.843333     3.057333
```

1. Use `.data` / `.env` or `all_of()` to disambiguate
2. Tunnel data-variables and selections with `{{   }}`