Data wrangling

Oliver Kirchkamp

Universität Jena

Data wrangling

Working with data...

- Data cleaning
- Data wrangling
- Descriptives, tests, estimates, graphs

In this part of the course...

- Work with text, regular expressions.
- Dates
- Work with HTML, XML, XPath.
- Work with unusual datasets.
- Application/exercise.

Working with Text: (1) Regular Expressions

- Goal:
  - Find structure (in text)
  - Replace with a different structure.
- Examples:
  - Find email, zip-code,...
- Applications:
  - Shell commands: `sed`, `grep`,...
  - R: `sub`, `gsub`, `grep`, `grepl`,...
  - Python: `re.search`, `re.sub`,...
  - ⋮

Regular expressions...

- They are daunting.
- They can be very helpful!

Regular expressions in R

- Find things:
  `grep`, `grepl`
  For approximate finds:
  `stringdist::afind`
- Replace things:
  `sub`, `gsub`,
  `stringr::str_replace`, `stringr::str_replace_all`, `stringr::str_extract_all`
- Splitting things up:
  `strsplit`

Examples

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown","Susan Smith")
grep("Susan",text)
```

```
[1] 3 5
```

```
grep("Susan",text,value=TRUE)
```

```
[1] "Susan Jones" "Susan Smith"
```

```
grepl("Susan",text)
```

```
[1] FALSE FALSE  TRUE FALSE  TRUE
```

```
sub("Susan","Sally",text)
```

```
[1] "Jane Smith"  "John Taylor" "Sally Jones" "Jack Brown"  "Sally Smith"
```

Elements of regular expressions

- History:
  - Stephen Cole Kleene (1951). “Representation of Events in Nerve Nets and Finite Automata”.
  - Ken Thompson (1968). “Programming Techniques: Regular expression search algorithm.”

  ↓ Today:
- Most characters (`01234...ABCDE...`) just match themselves.
- Sets of characters are denoted by `[...]`. Characters within the brackets can be...
  - listed individually `[,;:\.]`
  - ranges `[a-z0-9]`
  - complemented `[^a-z]`
  - any character: `.`
- Alternatives: `|`
  - `Susan|Sally`
- Groups: `(...)`
  - `(The|That) (dog|cat)`
- Repetition: `* + ?`
  - `[0-9]+`

Elements of regular expressions

Most characters (01234...ABCDE...) just match themselves, except...

`.`	any character	
`^`	start of the string	`^Susan`
`$`	end of the string	`Susan$`

Characters that match repetitions of preceding RE:

`*`	0 or more repetitions of preceding RE	`.*`	`A*`	`[0-9]*`	`(Susan)*`
`+`	1 or more repetitions of preceding RE	`.+`	`A+`	`[0-9]+`	`(Susan)+`
`?`	0 or 1 repetitions of preceding RE	`.?`	`A?`	`[0-9]?`	`(Susan)?`
`{m}`	exactly m matches of preceding RE	`A{3}`	`AAA`		
`{m,n}`	m to n matches of preceding RE	`A{3,5}`	`AAAA?A?`		

Repetitions are greedy. The can be followed by `?` to make them lazy:

```
sub(".*A","-","xyzABCAgg")

[1] "-gg"

sub(".*?A","-","xyzABCAgg")

[1] "-BCAgg"
```

Examples

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
grep("s.n",text)

[1] 3

sub("s.n","-",text)

[1] "Jane Smith"  "John Taylor" "Su- Jones"   "Jack Brown"

sub("^..s","-",text)

[1] "Jane Smith"  "John Taylor" "-an Jones"   "Jack Brown"

sub("t.*$","-",text)

[1] "Jane Smi-"   "John Taylor" "Susan Jones" "Jack Brown"

sub("(?i)t.*$","-",text) ## (?i) ignores case

[1] "Jane Smi-"   "John -"      "Susan Jones" "Jack Brown"
```

Repetition

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
sub("[a-z]*","-",text)
```

```
[1] "-Jane Smith"  "-John Taylor" "-Susan Jones" "-Jack Brown"
```

```
sub("[a-z]+","-",text)
```

```
[1] "J- Smith"  "J- Taylor" "S- Jones"  "J- Brown"
```

```
gsub("[a-z]*","-",text)
```

```
[1] "-J- -S-" "-J- -T-" "-S- -J-" "-J- -B-"
```

```
gsub("[a-z]?","-",text)
```

```
[1] "-J--- -S----"  "-J--- -T-----" "-S---- -J----" "-J--- -B----"
```

Sets

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
gsub("[a-z]","-",text)    ## [a-z] is a "set"

[1] "J--- S----"  "J--- T-----" "S---- J----" "J--- B----"

gsub("[^a-z]","-",text)

[1] "-ane--mith"  "-ohn--aylor" "-usan--ones" "-ack--rown"
```

Alternatives and groups

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
gsub("Susan|John","-",text)  ## .../... is an alternative

[1] "Jane Smith" "- Taylor"   "- Jones"    "Jack Brown"

gsub("(Susan|John)"," [\\1] ",text)  ## (...) is a group

[1] "Jane Smith"      " [John]  Taylor" " [Susan]  Jones" "Jack Brown"
```

Groups

```
text <- c("Jane T. Smith","John W. Taylor","Susan S. Jones","Jack Brown, Jr.","Susan Smith")
sub("Susan (.*)$","\\1, S.",text)

[1] "Jane T. Smith"   "John W. Taylor"  "S. Jones, S."    "Jack Brown, Jr."
[5] "Smith, S."

gsub("(.)(.*)\\1","-\\2-",text)

[1] "Jane T. Smith"   "John W. Taylor"  "-usan -. Jones"  "-ack Brown, -r."
[5] "-usan -mith"

sub("(.*) (.*)$","\\2, \\1",text)   ## greedy \1

[1] "Smith, Jane T."   "Taylor, John W."  "Jones, Susan S."  "Jr., Jack Brown,"
[5] "Smith, Susan"

sub("(.*?) (.*)$","\\2, \\1",text)  ## lazy \1

[1] "T. Smith, Jane"   "W. Taylor, John"  "S. Jones, Susan"  "Brown, Jr., Jack"
[5] "Smith, Susan"
```

Example

```
text <- "Susan Jones, susan.jones@some.domain.com, https://www.susan-jones.net/,
         01234 Somecity,..."
sub("^.*?([a-z\\.]+@[a-z\\.]+).*$","\\1",text)

[1] "susan.jones@some.domain.com"

sub("^.*(https?://[a-z\\.-]+/?).*$","\\1",text)

[1] "https://www.susan-jones.net/"

sub("^.*[^0-9]([0-9]{4,5})[^0-9a-zA-Z]+([a-zA-Z]+).*$","\\2, \\1",text)

[1] "Somecity, 01234"
```

The stringr library

```
text <- c("Jane Smith","John Taylor","Susan Jones","Jack Brown")
stringr::str_replace_all(text,c("Jane"="Joan","John"="Jack","Jack"="Tim"))
```

```
[1] "Joan Smith"  "Tim Taylor"  "Susan Jones" "Tim Brown"
```

```
stringr::str_count(text,c("J|a|o"))
```

```
[1] 2 4 3 3
```

```
stringr::str_extract_all(text,c("J|a|o"))
```

```
[[1]]
[1] "J" "a"

[[2]]
[1] "J" "o" "a" "o"

[[3]]
[1] "a" "J" "o"

[[4]]
[1] "J" "a" "o"
```

Splitting strings

```
text <- c("Mary Jane Smith","John Taylor","Susan Jones","Jack-Brown")
strsplit(text," ")

[[1]]
[1] "Mary"  "Jane"  "Smith"

[[2]]
[1] "John"   "Taylor"

[[3]]
[1] "Susan" "Jones"

[[4]]
[1] "Jack-Brown"

strsplit("Jack-Brown","[ -]")

[[1]]
[1] "Jack"  "Brown"
```

Approximate matches

```
##          123456789 123456789 123456789 12
text <- c("Today, Anne and John do the work",
          "Tomorrow, Susan and Joan are singing")
```

```
stringdist::afind(text,
                  c("Anna","John","Liz"),
                  value=FALSE,method="jw")

$location
     [,1] [,2] [,3]
[1,]    7   17    1
[2,]   15   21   30

$distance
          [,1]      [,2]      [,3]
[1,] 0.1666667 0.0000000 1.0000000
[2,] 0.2777778 0.1666667 0.4444444
```

```
stringdist::afind(text,
                  c("Anna","John","Liz"),
                  value=FALSE,method="soundex")

$location
     [,1] [,2] [,3]
[1,]    8   17    1
[2,]   14   21    1

$distance
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    0    1
```

Date and Time

```
as.POSIXct("01-02-03")
```

```
[1] "1-02-03 LMT"
```

```
as.POSIXct("01-02-03") |> class()
```

```
[1] "POSIXct" "POSIXt"
```

```
as.POSIXct("01-02-03") |> unclass()
```

```
[1] -62132748808
attr(,"tzone")
[1] ""
```

```
as.POSIXct("01-02-03") |> format("%d.%m.%Y")
```

```
[1] "03.02.1"
```

```
as.POSIXct("01-02-03",format="%d-%m-%y",tz="UCT")
```

```
[1] "2003-02-01 UTC"
```

```
date.pct <- as.POSIXct("01-02-03"   ,
                       format="%d-%m-%y",tz="UCT")
format(date.pct,"%Y")
```

```
[1] "2003"
```

```
format(date.pct,"%A, %d %B %Y")
```

```
[1] "Saturday, 01 February 2003"
```

```
format(date.pct,"%s seconds since the epoch")
```

```
[1] "1044054000 seconds since the epoch"
```

```
unclass(date.pct)
```

```
[1] 1044057600
attr(,"tzone")
[1] "UCT"
```

Date and Time – Limitations

- POSIX time is stored (at least) as a 32 bit signed integer.
  (modern desktop computers use 64 bits, but embedded systems (routers, cars...) may not.)

```
as.POSIXct(0)

[1] "1970-01-01 01:00:00 CET"

as.POSIXct(-2^31)

[1] "1901-12-13 21:45:52 CET"

as.POSIXct(2^31)

[1] "2038-01-19 04:14:08 CET"
```

- POSIX time “ignores” leap seconds.

Date and Time

Flexible formats:

```
date <- "01-02-03"
date2 <- "02:03:04"
as.POSIXct(c(date,date2),format="%d-%m-%y",tz="UCT

[1] "2003-02-01 UTC" NA

##
library(lubridate)
parse_date_time(c(date,date2),orders="dmy",
                locale = "de_DE.UTF-8")

[1] "2003-02-01 UTC" "2004-03-02 UTC"

dmy(c(date,date2))

[1] "2003-02-01" "2004-03-02"
```

Arithmetic with dates:

```
dmy(date2)-dmy(date)

Time difference of 395 days

mean(c(dmy(date) ,dmy(date2)))

[1] "2003-08-17"

mean(c(dmy(date) ,dmy(date2))) + dweeks(2)

[1] "2003-08-31 12:00:00 UTC"

dmy(date) + dweeks(3)/2

[1] "2003-02-11 12:00:00 UTC"
```

Python

```
import pandas as pd
text = pd.Series(["Mary Jane Smith","John
Taylor","Susan Jones","Jack-Brown"])
text.str.count("Susan")

0    0
1    0
2    1
3    0
dtype: int64

text.str.contains("Susan")

0    False
1    False
2     True
3    False
dtype: bool
```

```
text.str.replace("Susan","Sally")

0    Mary Jane Smith
1         John Taylor
2        Sally Jones
3         Jack-Brown
dtype: object

text.str.split(" ")

0    [Mary, Jane, Smith]
1         [John, Taylor]
2         [Susan, Jones]
3           [Jack-Brown]
dtype: object
```

Working with HTML

- HTML = HyperText Markup Language (Tim Berners-Lee, 1980)
- XML = Extensible Markup Language (1998)
- other markup languages, like LaTeX, markdown, ...

HTML

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></ul>
    </div>
  </body>
</html>
```

- HTML is (for the browser and the server) the internal representation of a page.
- A browser renders the HTML according to the requirements of the user interface (screen of different sizes, printed page, braille terminal, screen reader...).

Elements `<html>`, `<head>`, `<body`, `<div>`, `<h2>`, `<ul>`, `<li>`...

Elements and tags

	Start-tag	End-tag
The entire page	`<html>`	`</html>`
The body of the page	`<body>`	`</body>`
Heading	`<h2>`	`</h2>`
Block of text	`<div class=...>`	`</div>`
Unordered list	`<ul>`	`</ul>`
List item	`<li>`	`</li>`
Anchor	`<a href=...>`	`</a>`

Attributes `class='intro'`, `href=...`,

Elements and attributes tell us more about the type of the information.
Example:

```
<img src='https://www.some.dom/P123.jpeg' alt='A picture of a dog'>
```

Reading HTML in R

- If we want to read many pages into R, we need a command.
  $\rightarrow$ `httr::GET` reads pages and stores them somehow.
- We also want to translate these pages into
  - text.
  - a tree.

  `httr::content` does the translation.
  - `httr::content(...,as='text')`
  - `httr::content(...,as='parsed')`

  The `xml2` library provides commands to navigate the tree.

Reading HTML in R

```
library(httr)
library(xml2)
thisPage <- httr::GET("https://www.some.domain.org/")
httr::content(thisPage,as="text")
xmlTree <- httr::content(thisPage,as="parsed")
```

```
xmlTree

{xml_document}
<html>
[1] <head>\n  <title>My webpage</title>\n</head>
[2] <body>\n  <div class="heading">\n    <h1>This is a Title</h1>\n  </div>\n ...

xml_children(xml_children(xmlTree)[2])

{xml_nodeset (3)}
[1] <div class="heading">\n  <h1>This is a Title</h1>\n</div>
[2] <div class="abstract">\n      Lorem ipsum dolor sit amet, consectetur adi ...
[3] <div class="intro"><h2>Introduction</h2>\n      Ut enim ad minim <a href= ...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></
    </div>
  </body>
</html>
```

XPath (1999)

- HTML pages can have a complicated structure.
- The tree-representation can a complicated structure, too.

We may need a tool to find things in these trees.

→ XPath is a notation to describe search requests for XML trees.

XPath

XPath is a notation to select nodes in an XML or HTML document.

- `xml_find_all(document,'//a')`
  finds all <a...> ... nodes (all hyperlinks) in that document.

```
xml_find_all(xmlTree,"//a")

{xml_nodeset (1)}
[1] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>
```

```
node <- xml_find_all(xmlTree,"//a")
xml_children(node)

{xml_nodeset (1)}
[1] <i>veniam</i>

xml_parent(node)

{xml_nodeset (1)}
[1] <div class="intro"><h2>Introduction</h2>\n      Ut enim ad minim <a href= ...
```

XPath

```
xml_children(xml_parent(node))

{xml_nodeset (3)}
[1] <h2>Introduction</h2>
[2] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>
[3] <ul>\n  <li class="myItems">Item one.</li>\n  <li class="otherItems">Item ...

xml_text(node)

[1] "veniam"

xml_attrs(node)

[[1]]
                    href
"https://www.veniam.org/"

xml_find_all(xmlTree,"//li[@class='myItems']")

{xml_nodeset (1)}
[1] <li class="myItems">Item one.</li>
```

Search for text on a page

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
  <head>
    <title>My webpage</title>
  </head>
  <body>
    <div class="heading">
      <h1>This is a Title</h1>
    </div>
    <div class="abstract">
      Lorem ipsum dolor sit amet, consectetur adipisici elit,...
    </div>
    <div class="intro"><h2>Introduction</h2>
      Ut enim ad minim <a href="https://www.veniam.org/"><i>veniam</i></a>,...
      <ul><li class="myItems">Item one.</li><li class="otherItems">Item two.</li></ul>
    </div>
  </body>
</html>
```

Search for text on a page

```
"https://www.some.domain.org/" |>
   httr::GET() |>
   httr::content(as="parsed") -> xmlTree
```

```
xmlTree |> xml_find_all("//*[contains(text(),'veniam')]")

{xml_nodeset (1)}
[1] <i>veniam</i>

xmlTree |> xml_find_all("//*[contains(text(),'veniam')]") |> xml_parent()

{xml_nodeset (1)}
[1] <a href="https://www.veniam.org/">\n  <i>veniam</i>\n</a>

xmlTree |> xml_find_all("//*[contains(text(),'veniam')]") |> xml_parent() |> xml_attrs()

[[1]]
                     href
"https://www.veniam.org/"
```

XPath

- `xml_find_all(document,'//table')`
  finds all `<table...> ... </table>` nodes (all tables) in that document.
- `xml_find_all(document,'//table[@summary="Notes"]')`
  finds all `<table ... summary="Notes"> ... </table>` nodes
  (all tables with a summary attribute which has the value ”Notes”) in that document.
- `xml_find_all(document,'//p[@class="Relevant"')`
  finds all `<p ... class="Relevant"> ... </p>` nodes
  (all paragraphs with a class attribute which has the value ”Relevant”) in that document.
- `xml_find_all(document,"//*[contains(text(),'veniam')]")`
  finds all nodes that contain the text `veniam`.

XPath

- `xml_children(node)`
  returns all children of a node (e.g. if the node is a table (`<table>...</table>`), then the children are often the rows of that table. If the node is a list (`<ul><li>...</li><li>...</li></ul>`), then the children are the items of that list...).
- `xml_parent(node)`
  returns the parent of that node (e.g. if a node is a row of table, then the parent could be the entire table...)
- `xml_text(node)`
  A textual representation of the node.
- `xml_attrs(node)`
  All attributes of a node.

Tables in HTML

```
library(rvest)
"https://www.some.domain.org/" |>
   httr::GET() |>
   httr::content() |>
   rvest::html_table()
```

...provides a list with all tables on that page.

Working with data, repeating ourselves

Data is a list

```
lapply(data,function(x) {
    ...
    do_something_with_x()
    ...
    data.frame(a=...,b=...,c=...)
}) |>
    bind_rows()

sapply(data,function(x) {...})
```

Data is a grouped dataframe

```
data |>
group_by(variable) |>
mutate(...) |>      ## → df with same # of rows
summarise(...) |>   ## → df with 1 row
reframe(...) |>     ## → df with new # of rows
ungroup()
```

Working with data, repeating ourselves

- The data is a list.
  Perform operation for each item in the list:

```
lapply(data,function(x) {...})  ## → list
bind_rows()                     ## list → dataframe
sapply(data,function(x) {...})  ## → matrix
```

- The data is a dataframe.
  Perform operation for groups of data:

```
data |>
group_by(variable) |>
mutate(...) |>      ## → df with same # of rows
summarise(...) |>   ## → df with 1 row
reframe(...) |>     ## → df with new # of rows
ungroup()
```

Grouping

```
group_by(variable)
rowwise()           ## each row is one group
##
ungroup()
```

Perform operation for several variables in a data frame.

```
mutate(across(...variables...,function() {}))
```

Dataframes with unusual elements

Using dataframes makes it easier to keep things together (in one row) that belong together.

$$\begin{bmatrix} a_1 & b_1 & \cdots & x_1 \\ a_2 & b_2 & \cdots & x_2 \\ a_3 & b_3 & \cdots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & b_n & \cdots & x_n \end{bmatrix} \quad (1)$$

Each row is one record, e.g. one person, one name, one birth-date,...

But what if a record (a row) contains information about a variable number of things (languages spoken, books read, past employment, degrees...) Columns in dataframes can contain as elements lists or even other dataframes:

$$\begin{bmatrix} a_1 & b_1 & \cdots & \{x_{11}, x_{12}\} \\ a_2 & b_2 & \cdots & \{x_2\} \\ a_3 & b_3 & \cdots & \{x_{31}, x_{32}, x_{33}\} \\ \vdots & \vdots & \ddots & \vdots \\ a_n & b_n & \cdots & \{x_{n1}, \ldots, x_{nk}\} \end{bmatrix} \quad (2)$$

To tell R that a `list()` is meant to be a single item:

```
mutate(a=f(b),x=I(list(...)))
data.frame(a=f(b),x=I(list(...))
```

An application

Assume, we want to study data from the German Constitutional Court
(as in Engel (2022), “Lucky you: Your case is heard by a seasoned panel—Panel effects in the German Constitutional Court”, etc.)

- Fetch court cases
- Extract for each case
  - date
  - judges
  - body
  - topic (econ, justice, political, person)
  - whether case was accepted

Libraries to load

```
library(xml2)
library(dplyr)
library(parallel)
options(mc.cores=detectCores()/2)
library(httr)
library(ggplot2)
library(xtable)
```

Robots policy

Are we welcome?

```
## get robots policy:
HOST <- "https://www.bundesverfassungsgericht.de/"
robots <- httr::GET(paste0(HOST,"robots.txt"))
cat(httr::content(robots))
```

```
User-agent: *
Disallow: /SiteGlobals/
Disallow: /DE/Service/
Disallow: /EN/Service/
Allow: /SiteGlobals/Functions/JavaScript/
Allow: /SiteGlobals/StyleBundles/
Allow: /SiteGlobals/Frontend/
Crawl-delay: 10
```

Inspecting HTML

- Firefox: Right-click on element and select “Inspect Element”.
- Chrome: Right-click on element and select “Inspect”.

Parse overviews

The BVerfG presents an overview of 10 decisions on one page. Each entry for one decision looks like

```
<ol id="searchResult" class="links">
   <li>1. ... <a href="SharedDocs/Entscheidungen...">
                    <span class="aktenzeichen">...</span>
                    <div>Beschluss vom ...</div>
               </a>
               <div class="kurztext">...</div>
   </li>
   <li>2. ...</li>
   ...
</ol>
```

The address of these pages follows a structure:

```
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D1
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D2
https://www.bundesverfassungsgericht.de/SiteGlobals/...?gtp=5403124_list%3D3
```

Read summaries

```
## get search results for all judgements:
page <- 1
allTOC <- list()
while(TRUE) {
    thisPage <- httr::GET(paste0("https://www.bundesverfassungsgericht.de/SiteGlobals/Forms/",
                                 "Suche/Entscheidungensuche_Formular.html?gtp=5403124_list%3D",page))
    if(thisPage |> httr::content(as="parsed") |> xml_find_all("//ol[@id='searchResult']") |>
                                                        xml_children() |> length() < 1)
        break; ## stop if there are no more search results
    cat(page,",") ## some feedback to watch
    allTOC[[page]] <- thisPage
    page <- page + 1
    Sys.sleep(10)
}
save(file="data/TOCresponse.Rdata",allTOC)
```

Parse summaries

```
## parse these search results:
lapply(allTOC,function(p) {
    thisPage <- httr::content(p,as="parsed") ## translate search result into xml tree
    links <- xml_find_all(thisPage,
                          "//ol[@id='searchResult']/li/a[@href[contains(.,'Entscheidungen')]]")
    lapply(links,function(n) {
        data.frame(
            az = xml_text(xml_find_first(n,"./span[@class='aktenzeichen']")),
            date = xml_text(xml_find_first(n,"./div")),
            href = xml_attr(n,"href"),
            summary = xml_text(xml_find_first(xml_parent(n),"./div[@class='kurztext']/p"))
        )}) |>
        bind_rows()
}) |> bind_rows() -> allTOC.df
```

Control: do we have multiple observations

```
str(allTOC.df)

'data.frame': 9071 obs. of  4 variables:
 $ az     : chr  "2 BvQ 52/23" "2 BvQ 51/23" "1 BvR 601/23" "2 BvR 406/23" ...
 $ date   : chr  "Beschluss vom 3. Mai 2023" "Beschluss vom 29. April 2023" "Beschluss vom 24. April
 $ href   : chr  "SharedDocs/Entscheidungen/DE/2023/05/qk20230503_2bvq005223.html" "SharedDocs/Entsch
 $ summary: chr  "Eilantrag betreffend Teilungsversteigerungsverfahren nach Maßgabe einer Folgenabwäg

allTOC.df |>
    group_by(az,date) |>
    summarise(n=n()) |>
    ungroup() |>
    with(table(n))

n
   1    2    4
8983   42    1
```

Obtain individual judgements

```
##
wPages <- list()
for (i in 1:nrow(allTOC.df)) {
    url <- paste0(HOST,allPages.df[i,"href"])
    wPages[[i]] <- GET(url)
    cat(i,",")
    Sys.sleep(10)
}
save(file="data/wPages.Rdata",wPages,allTOC.df)
```

Tasks

- Judgement $\rightarrow$ date
- Judgement $\rightarrow$ href
- Judgement $\rightarrow$ body (chamber, senate,...)
- Judgement $\rightarrow$ topic (econ, justice, political, person)
- Judgement $\rightarrow$ judges (2 ways)

Judgement to body

```
...
<div class="absatz">
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr1">
       <span>
          hat die 2. Kammer des Zweiten Senats des Bundesverfassungsgerichts durch
       </span>
     </p>
   </div>
</div>
...
```

We will look for a `<div class="rr1">` to identify the body.

Judgement to body. A little helper function

```
str2num <- function(x) {
    x |> stringr::str_replace_all(c(`(?i)first|erst|premier`="1",
                                    `(?i)second|zweite`="2",
                                    `(?i)third|dritte`="3",
                                    `(?i)fourth|vierte`="4")) |>
        gsub("[^0-9]","",x=_) |>
        as.numeric()
}
##
str2num("Le premier Senat")
```

```
[1] 1
```

```
str2num("The Second Chamber")
```

```
[1] 2
```

```
str2num("Die dritte Kammer")
```

```
[1] 3
```

Judgement to body

```
ptree2body <- function (ptree) {
    ptree |> xml_find_all("//*[@class='rr1']/span") |> xml_text() -> ctable
    if(length(ctable)==0)
        ptree |> xml_find_all("//*[@class='rr1']") |> xml_text() -> ctable
    ctable |> stringr::str_count("Kammer|Chamber|Senat") -> cmatchnum
    which(max(cmatchnum)==cmatchnum)[1] -> cmatch
    ifelse(is.na(cmatch),NA,ctable[cmatch]) |>
    stringr::str_replace_all("[\n  ]+"," ") |>
    stringr::str_replace(paste0("^.*? (([1-9]\\.|First|Second|Third) *(Kammer|Chamber).*",
                                "(Senat[se]?|Panel)|[a-zA-Z]+ [sS][eé]nat[se]?|Beschwerdekammer|",
                                "Complaints Chamber|Plenum|Plenary).*"),"\\1") -> bvbody
    bvbody |> stringr::str_replace("^((.*) (Kammer|Chamber))?.*$","\\2") |> str2num() -> chamber
    bvbody |> stringr::str_replace("^.*?(([^ ]+) (Senat|Panel|[sS]énat)).*$","\\2") |>
        str2num() -> senat
    "" -> bvbody2
    if(grepl("Plenum|Plenary",bvbody)) "Plenum" -> bvbody2
    if(grepl("Beschwerdekammer|Complaints Chamber",bvbody)) "Beschwerdekammer" -> bvbody2
    if(!is.na(senat)) paste0(senat,".S") -> bvbody2
    if(!is.na(chamber)) paste0(senat,"/",chamber) -> bvbody2
    bvbody2
}
```

Judgement to body

```
wpage=wPages[[4103]]
ptree <- content(wpage,as="parsed")
ptree2body(ptree)
```

```
[1] "2.S"
```

Judgement to judges. In the introduction, just after the body:

```
...
<div class="absatz">
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr2">
       <span>
          den Richter ...
       </span>
     </p>
   </div>
   <div class="rechts">...</div>
   <div class="links">
     <p class="rr2">
       <span>
          und die Richterinnen ...
       </span>
     </p>
```

We will search for `<div class="rr2">`
In XPath notation: `"//p[@class='rr2']//span"`
Since sometimes the span is missing, we also look for `"//p[@class='rr2']"`

- Problem: Names of judges are decorated with other things (»den Richter...«)
- Solution: We will try to remove these things as good as we can for most cases.
- This will allow us to identify names of judges.
- We will use these names to process the remaining cases.

Judgement → judges (1)

```
ptree2judges1 <- function(ptree) {
    jtable1 <- xml_text(xml_find_all(ptree,"//p[@class='rr2']//span"))
    if(length(jtable1)==0)
        lapply(xml_text(xml_find_all(ptree,"//p[@class='rr2']")),strsplit,"\n| und |,") |>
            unlist() -> jtable1
    ##
    paste0("^(nd | *| *und |and |durch |unter Mitwirkung )|([Dd]er |[Dd]ie |[Dd]en |des )?",
           "(Richter|Judge|Justice|(Vi[zc]e[ -]?)?[pP]r[äeé]sident(e)?)(in|en|innen)?s? ?|",
           "$| ?[()]|(und|and)$") |> gsub("",jtable1) |> as.list() |>
    lapply(X=_,strsplit,",") |> unlist() -> jtable2  ## multiple judges in one cell
    paste0(", *|^[  ]*| *$|.*hat die Kammer|.*Senat.*|.*Bundesverfassungsgericht|",
           ".*Mitwirkung|.*in Verbindung|.*Bekanntmachung.*|.*beschlossen.*") |>
    gsub("",jtable2) |> setdiff(c("und","and",""))
}
##
ptree2judges1(content(wPages[[4103]],as="parsed"))
```

```
[1] "Voßkuhle"    "Di Fabio"    "Mellinghoff" "Lübbe-Wolff" "Gerhardt"
[6] "Landau"      "Huber"       "Hermanns"
```

Judgement to judges (2). At the end of the page:

```
...
<table class="st" summary="Unterschriften der Richter" width="80%" align="center">
  <tbody ...>
    <tr>
      <td class="st" colspan="4" width="33%">
         <span>Müller</span>
      <td>
      <td class="st" colspan="4" width="33%">
         <span>Langenfeld</span>
      <td>
      <td class="st" colspan="4" width="33%">
         <span>Fetzer</span>
      <td>
    </tr>
   </tbody>
<table>
```

Judgement to judges (2)

Why collect the same information twice?

→ No version is 100% reliable, so we better compare.

Judgement → judges (2)

```
ptree2judges2 <- function(ptree) {
    jtable2 <- xml_find_all(ptree,"//table[@summary='Unterschriften der Richter']//td//span")
    if(length(jtable2)==0)
        jtable2 <- xml_find_all(ptree,"//table[@summary='Unterschriften der Richter']//td")
    ##
    jtable2 |> xml_text() |> lapply(strsplit," und |:|,") |> unlist() |>
        gsub("[, :]*$","",x=_) |>
        gsub("[_ \n]"," ",x=_) |>
        gsub("^[ ]*","",x=_) |>
        gsub(paste0("Dr. |Die |Der |(Richter|Vizepräsident)(in)? |(ist|deshalb).*gehindert|",
                    "Amt ausgeschieden|deshalb an der Unterschrift|gehindert."),"",x=_) |>
        gsub("- *","-",x=_) |>
        setdiff(c("","  ","Judges","unter","Mitwirkung"))
}
##
ptree2judges2(content(wPages[[4103]],as="parsed"))
```

```
[1] "Voßkuhle"    "Di Fabio"    "Mellinghoff" "Lübbe-Wolff" "Gerhardt"
[6] "Landau"      "Huber"       "Hermanns"
```

Map articles to topics

In the text of the judgement:

> *...Art. 2 Abs. 1 GG gewährleistet in Verbindung mit Art. 1 Abs. 1 GG das allgemeine Persönlichkeitsrecht. Dieses Recht...*

Map articles to topics

```
read.csv(text="art,topic
2.1|3.1|9.3|12.1|14.[12],econ
2.2|10.|11.|13.|16.2|19.4|101.|103.[123],justice
5.[123]|8.|9.[12]|28.2|33.[2345]|38.,political
4.[12]|6.|7.|16a.|140.,person") |>
    mutate(art = paste0("^(",gsub("\\.","\\\\.",art),").*$"),
           pat=setNames(topic,art)) |> pull(pat) -> art2topic
```

Now we have a vector with unusual names:

```
art2topic
```

```
                      ^(2\\.1|3\\.1|9\\.3|12\\.1|14\\.[12]).*$
                                                        "econ"
^(2\\.2|10\\.|11\\.|13\\.|16\\.2|19\\.4|101\\.|103\\.[123]).*$
                                                     "justice"
        ^(5\\.[123]|8\\.|9\\.[12]|28\\.2|33\\.[2345]|38\\.).*$
                                                   "political"
                       ^(4\\.[12]|6\\.|7\\.|16a\\.|140\\.).*$
                                                      "person"
```

Judgement → topic

```
ptree2topic <- function(ptree) {
    ptree  |>
    xml_find_all("//*[contains(text(),' GG')]")   |>
        lapply(function(n) stringr::str_extract_all(xml_text(n),
               "Art\\.[  ]*[0-9]+[  ,]*(Abs.[  ][0-9]+)?.{0,10}GG")) |>
    unlist() |>
    stringr::str_replace_all("^Art\\.[  ]*([0-9]+)[  ,]*(Abs.[  ]([0-9]+))?.*$","\\1.\\3") |>
    stringr::str_replace_all(art2topic) -> dirtyMatches
    dirtyMatches[dirtyMatches %in% art2topic] |>
        table() |> sort(decreasing=TRUE) |> names() |> head(1) -> topic
    if(is.null(topic)) topic<-NA
    topic
}
##
sapply(4104:4110,function(p) ptree2topic(content(wPages[[p]],as="parsed")))
```

```
[1] "justice" "econ"    NA        NA        "econ"    "justice" NA
```

Parse all individual judgements

(`mclapply` is similar to `lapply`, except that it is parallel, hence, faster.)

```
mclapply(wPages,function(wpage) {
    href <- sub(HOST,"",wpage[["url"]])
    ptree <- content(wpage,as="parsed")
    bvbody2 <- ptree2body(ptree)
    topic <- ptree2topic(ptree)
    judges1 <- ptree2judges1(ptree)
    judges2 <- ptree2judges2(ptree)
    data.frame(href=href,bvbody=bvbody2,topic=topic,j1=I(list(judges1)),j2=I(list(judges2)))
}) |> bind_rows() -> judgesRaw
```

Judges might need some cleaning...

A first look at the judges

```
## find names of judges
judgesRaw |> select(j1,j2) |> unlist() |>  table() |>
   sort(decreasing=TRUE) -> jTable
##
head(jTable) |> t() |> xtable()
```

	Papier	Kirchhof	Kessal-Wulf	Huber	Mellinghoff	Hassemer
1	2910	2679	2532	2411	2278	2243

```
tail(jTable,5) |> t() |> xtable()
```

	vom 11. August 1993BGBl I S. 1473 am 13. Mai 2009	W	Wintrich	Zeidler	Zeidler als Vorsitzer
1	1	1	1	1	

Find judges – (Generally: find miss-spelled strings)

So far:

- We have identified some judges (`realJudges`).

```
(names(jTable)[jTable>10] -> realJudges) |> paste(collapse=", ")
```

[1] "Papier, Kirchhof, Kessal-Wulf, Huber, Mellinghoff, Hassemer, Broß, Osterloh, König, Gaier, Steiner, Paulus, Müller, Di Fabio, Hermanns, Maidowski, Hohmann-Dennhardt, Landau, Hömig, Voßkuhle, Lübbe-Wolff, Masing, Baer, Schluckebier, Britz, Bryde, Eichberger, Jaeger, Gerhardt, Hoffmann-Riem, Limbach, Sommer, Langenfeld, Haas, Jentsch, Christ, Kühling, Ott, Grimm, Radtke, Harbarth, Wallrabenstein, Härtel, Winter, Seidl, Graßhof, Kruis, Seibert, Henschel, Söllner, Herzog, Wolff, Dieterich, Böckenförde, Klein, Offenloch, Fetzer, s durch, Mahrenholz, mit § 93a BVerfGG in der"
(We still have do do some cleaning)

- There are some "unknowns" which contain the correct string ("Voßkuhle als Vorsitzender")
  We identify them in `knownJ`.
- There are some spelling mistakes ("Vosskuhle" instead of "Voßkuhle")
  We get some help from `stringdist::afind` to match these, but we will do this "manually".

Match miss-spelled judges (1)

```
realJudges[!grepl("\\.|- Erster|BVerfGG|s durch",realJudges)]  |>
    sub("^Wolff","H.A.Wolff",x=_) -> realJudges
unrealJ <- setdiff(names(jTable),realJudges)
sapply(realJudges,function(j) grep(paste0("^(.*[^A-Z])?",j,"([^a-z].*)?$"),unrealJ)) |>
    unlist() |>  unique() |> sort() -> knownJ
unrealJ[-knownJ] -> unknown ## ← these contain missspelled names
str(unknown)
```

```
 chr [1:129] "Wolff" "s durch" " mit § 93a BVerfGG in der" "Benda" ...
```

```
stringdist::afind(unknown,realJudges,value=FALSE,method="jw") -> jDist
str(jDist)
```

```
List of 2
 $ location: int [1:129, 1:58] 1 1 10 1 1 1 4 4 1 1 ...
 $ distance: num [1:129, 1:58] 1 0.556 0.333 1 0.333 ...
```

Match miss-spelled judges (2)

```
lapply(1:length(unknown),
       function(i) {
           best=min(jDist$distance[i,]);
           ji=which(best==jDist$distance[i,])[1];
           data.frame(best=best,ji=ji,judge=realJudges[ji],
                      match=substr(unknown[i],jDist$location,40))}) |>
    bind_rows() |> filter(best < .2) |>
    summarise(pat=paste0(paste0(match,collapse="|"),",",judge[1]),.by="judge") |>
    xtable()
```

	judge	pat
1	Baer	mit § 93a BVerfGG in der|Haager| mit § 93 a BVerfGG in|Berger|BVerfGG|d
2	Voßkuhle	Vosskuhle,Voßkuhle
3	Hassemer	Hesse|Hasseme|Hasssemer,Hassemer
4	Kessal-Wulf	Kessal-Wulff,Kessal-Wulf
5	Hömig	Hmig,Hömig
6	König	Hörnig,König

Correct miss-spelled judges

```
## correct spelling of some judges:
read.csv(text="pattern,replace
Bro$|Brox,Broß
Gerhard$,Gerhardt
Grasshof,Graßhof
Hasseme$|Hasssemer,Hassemer
Hentschel,Henschel
Herrmanns,Hermanns
Hofmann-Riem,Hoffmann-Riem
Hohmann -Dennhardt|Hohmann-Dennhard$,Hohmann-Dennhardt
Hmig,Hömig
Kessal-Wulff,Kessal-Wulf
Khling,Kühling
Lübbe-Woff|Lübbe -Wolff|Lübbe- Wolff|Lübbe-Wollf,Lübbe-Wolff
Stein$|Steine ,Steiner
Vosskuhle,Voßkuhle
Wallrabensein,Wallrabenstein
^Wolff,H.A.Wolff") |>
    with(data=_,setNames(replace,pattern)) ->
    judgeTrans
```

Bulk replace...

```
str(judgeTrans)

 Named chr [1:16] "Broß" "Gerhardt" "Graßhof" "Hassemer" "Henschel" ...
 - attr(*, "names")= chr [1:16] "Bro$|Brox" "Gerhard$" "Grasshof" "Hasseme$|Hasssemer" ...

## spell-correct realJudges:
stringr::str_replace_all(realJudges,judgeTrans) -> realJudges2
## clean all judges:
cleanJudges <- function(j) {
    stringr::str_replace_all(unlist(j),judgeTrans) |>
    sapply(function(text) realJudges2[sapply(realJudges2,grepl,text )]) |>
        unlist() |> unique() |> sort()
}
##
cleanJudges(c("Der Vorsitzende Bro","Lübbe-Woff und Vosskuhle"))

[1] "Broß"        "Lübbe-Wolff" "Voßkuhle"
```

Bulk replace...

```
judgesRaw |> rowwise() |>
    mutate(across(c(j1,j2),function(j) I(list(cleanJudges(j))))) |>
    ungroup() -> jjClean
```

Check the cleaning

```
jjClean |>
    rowwise() |>
    mutate(j=list(unique(unlist(j1,j2)))) |>
    tidyr::unnest(cols=c(j)) |> with(table(j))

j
            Baer      Böckenförde            Britz             Broß
             806               36              770              988
           Bryde            Christ         Di Fabio         Dieterich
             740              462              874               37
      Eichberger           Fetzer            Gaier          Gerhardt
             734               19              922              694
         Graßhof            Grimm       H.A.Wolff             Haas
             123              396               37              525
        Harbarth           Härtel         Hassemer         Henschel
             333              220             1125               75
        Hermanns           Herzog    Hoffmann-Riem Hohmann-Dennhardt
             873               46              677              862
           Hömig           Huber            Jaeger           Jentsch
             860             1206              722              517
     Kessal-Wulf        Kirchhof            Klein            König
```

Compare our two sources for judges

```
jjClean |> rowwise() |> mutate(n1 = length(j1), n2=length(j2)) -> jjNN
with(jjNN,table(n1,n2)) |> xtable()
```

	0	1	2	3	4	5	6	7	8	15	16
0	162	0	0	10	0	0	0	0	18	0	0
1	0	7	0	0	0	0	0	0	0	0	0
2	0	0	4	1	0	0	0	0	0	0	0
3	19	0	2	7180	0	0	0	0	0	0	0
4	0	0	0	0	8	0	0	0	0	0	0
5	0	0	0	0	0	3	0	0	0	0	0
6	0	0	0	0	0	0	42	1	0	0	0
7	8	0	0	0	0	0	0	285	1	0	0
8	4	1	0	0	0	0	0	0	1301	0	0
15	0	0	0	0	0	0	0	0	0	2	0
16	0	0	0	0	0	0	0	0	0	0	1

Problems:

```
(probs <- with(jjNN,which(n1>0 & n2>0 & n1 != n2)))
```

```
[1] 4771 5810 7668 8754 8755 8823
```

```
## look at cases where number of judges j1 and j2 differ:
for (i in 1:length(probs)) {
    cat("\n--------------------",i,"-",probs[i],"--------------------\n")
    cat("1 clean:",paste(sort(unlist(jjClean[probs[i],"j1"])),collapse=","))
    cat("\n")
    cat("1 raw:  ",paste(sort(unlist(judgesRaw[probs[i],"j1"])),collapse=","))
    cat("--\n")
    cat("2 clean:",paste(sort(unlist(jjClean[probs[i],"j2"])),collapse=","))
    cat("\n")
    cat("2 raw:  ",paste(sort(unlist(judgesRaw[probs[i],"j2"])),collapse=","))
    cat("\n")
}
```

```
-------------------- 1 - 4771 --------------------
1 clean: Gaier,Kirchhof
1 raw:   -Dennhardt,Hohmann Gaier,Kirchhof--
2 clean: Gaier,Hohmann-Dennhardt,Kirchhof
2 raw:   Gaier,Hohmann-Dennhardt,Kirchhof

-------------------- 2 - 5810 --------------------
1 clean: Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Steiner
1 raw:    - Erster,Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Steiner--
2 clean: Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Papier,Steiner
2 raw:   Bryde,Gaier,Haas,Hoffmann-Riem,Hohmann-Dennhardt,Hömig,Papier,Steiner

-------------------- 3 - 7668 --------------------
1 clean: Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner
1 raw:   Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner--
2 clean: Haas,Hohmann-Dennhardt,Hömig,Jaeger,Kühling,Papier,Steiner
2 raw:   Haas,Hohmann-Dennhard,Hömig,Jaeger,Kühling,Papier,Steiner

-------------------- 4 - 8754 --------------------
1 clean: Grimm,Hömig,Seidl
1 raw:   Grimm,Hömig,Seidl--
2 clean: Grimm,Hömig
2 raw:   Grimm,Hömig
```

Outcomes of cases

Find patterns in the summaries of cases:

```
PAT <- list()
PAT[["reject"]] <- paste0("unsuccessful|abgelehnt|ablehnung|unzulässig|erfolglos|unbegrün|",
                          "nichtannahme|verfristet|verworfen|zurückgewiesen")
PAT[["accept"]] <- "^successful|erfolgreich| annahme"
##
PAT[["reject2"]] <- paste0("bestätigung| darf |erledigt|einstellung| genüg|",
                           "missbrauchsgebühr|verwerfung|versagung|verfassungsgemäß|",
                            "verfassungsmaßig| vereinbar|zulässig")
PAT[["accept2"]] <- paste0("anspruch|auslagenerstattung|aussetzung|ausgeschlossen|",
           "begründ|angenommen|angeordnet|anordnung|bedenken|bedürfnis|beanstand|",
           "berichtig|berücksichtig|bewillig| muss | bedarf|erfolgreich|fristgerecht|",
           "fehlende| hätte |hinreichen|klarstell|kompetenzwidrig|prozesskosten|",
           "rechtsschutzinteresse| nichtig|verstößt|pflicht|substantiier| verletz|trennung|",
           "unvereinbar|untersag|verfassungswidrig|verstoß|verstößt|verletzung|vorrang")
PAT[["not"]] <- "nicht|kein|trotz|unzureichen|entfällt|wegfall"
```

Outcomes of cases

```
## always take the longer list:
jjClean |> rowwise() |>
  mutate(judge = ifelse(length(j1)>length(j2),I(list(j1)),I(list(j2)))) |>
  ungroup() |>
  left_join(allTOC.df) |>
  mutate(date1 =  lubridate::dmy(date,locale = "de_DE.UTF-8",tz="CET")) |>
  filter(!is.na(date1) & date1 > as.POSIXct("1951-01-01")) |>
  mutate(reject = grepl(PAT[["reject"]],summary,ignore.case=TRUE),
         accept = !reject & grepl(PAT[["accept"]],summary,ignore.case=TRUE),
         not = grepl(PAT[["not"]],summary,ignore.case=TRUE),
         reject2 = xor(not,grepl(PAT[["reject2"]],summary,ignore.case=TRUE)),
         accept2 = xor(not,grepl(PAT[["accept2"]],summary,ignore.case=TRUE)),
         sumx = 10*(accept-reject) + (accept2-reject2),
         sumdec = ifelse(sumx==0,NA,sumx>0)
         ) -> jjSucc
jjSucc |> with(table(sumx))
```

```
sumx
 -11  -10   -9   -1    0    1    9   10   11
 744 1626  674 1444 2505 1399   64  123  444
```

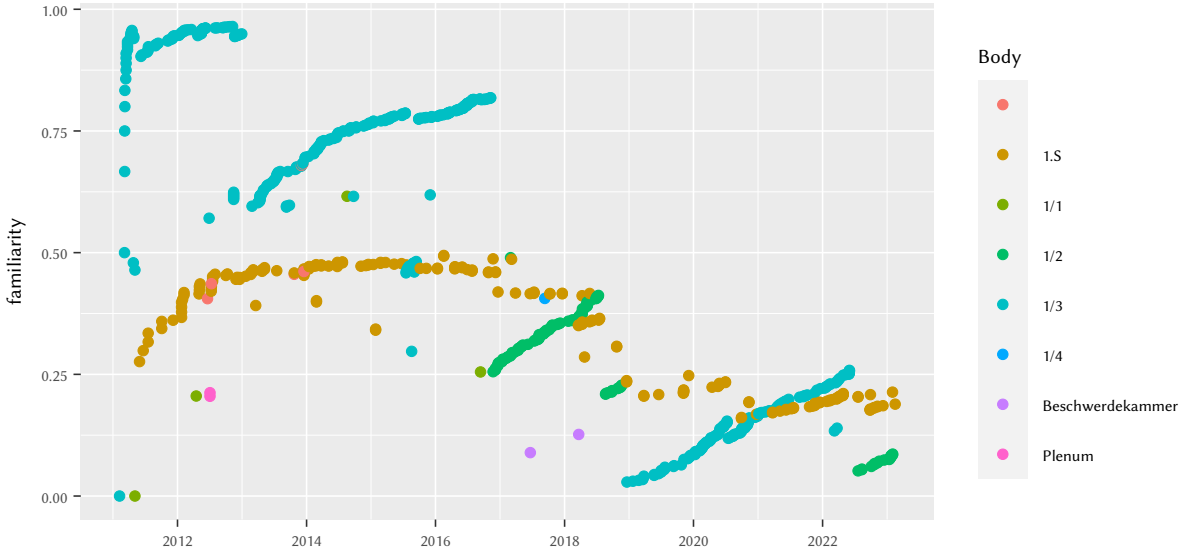
```
jjSucc |> filter(grepl("Baer",judge)) |>
    rowwise() |>
    reframe(data.frame(date=date1,body=bvbody,j=setdiff(judge,"Baer"))) |>
    ggplot(aes(x=date,y=factor(j,levels=rev(sort(unique(j)))),color=factor(body))) +
    geom_point() + labs(x=NULL,y=NULL,color="Body")
```

Familiarity of judges

Define “familiarity” as in Engel (2022), “Lucky you: Your case is heard by a seasoned panel—Panel effects in the German Constitutional Court”.

```
jjSucc |>
    rowwise() |>
    reframe(href,bvbody,date1,other=I(list(judge)),judge=data.frame(judge)) |>
    group_by(judge) |>
    arrange(date1) |>
    mutate(ownExp=1:n()) |>
    rowwise() |>
    reframe(href,bvbody,date1,judge,ownExp,data.frame(other)) |>
    filter(other != judge) |>
    group_by(judge,other) |>
    mutate(otherExp=0:(n()-1)/ownExp) |>
    group_by(href,bvbody,date1,judge) |>
        summarise(familiarity=mean(otherExp),.groups="drop") -> bvFamiliarity
```

```
bvFamiliarity |> filter(judge=="Baer" & date1>as.POSIXct("1950-01-01")) |>
    ggplot(aes(x=date1,y=familiarity,color=bvbody)) + geom_point() + labs(x=NULL,color="Body")
```



Parties

```
"https://de.wikipedia.org/wiki/Liste_der_Richter_des_Bundesverfassungsgerichts" |>
   httr::GET() |>
   httr::content() |>
   rvest::html_table() |> first() -> jWiki

stringdist::afind(x=unlist(jWiki[,1]),
                  pattern=sub("H.A.Wolff","Heinrich Amadeus Wolff",realJudges2),
                  value=FALSE,
                  method="jw") -> jwDist

lapply(seq(length(realJudges2)),function(j) {
    distances <- jwDist$distance[,j]
    m <- min(distances)
    pos=which(m==distances)[1]
    data.frame(judge=realJudges2[j],dist=m,jWiki[pos,])}) |>
    bind_rows() -> judgeDF
```

Parties

```
judgeDF |> arrange(-dist) |> select(judge,Name..Lebensdaten.,Vorschlag,dist)  |>
    head(12) |> xtable()
```

	judge	Name..Lebensdaten.	Vorschlag	dist
1	Papier	Hans-Jürgen Papier (* 1943)	CDU/CSU	0.00
2	Kirchhof	Ferdinand Kirchhof (* 1950)	CDU/CSU	0.00
3	Kessal-Wulf	Sibylle Kessal-Wulf (* 1958)	CDU/CSU	0.00
4	Huber	Peter M. Huber (* 1959)	CDU/CSU	0.00
5	Mellinghoff	Rudolf Mellinghoff (* 1954)	CDU/CSU	0.00
6	Hassemer	Winfried Hassemer (1940–2014)	SPD	0.00
7	Broß	Siegfried Broß (* 1946)	CDU/CSU	0.00
8	Osterloh	Lerke Osterloh (* 1944)	SPD	0.00
9	König	Doris König (* 1957)	SPD	0.00
10	Gaier	Reinhard Gaier (* 1954)	SPD	0.00
11	Steiner	Udo Steiner (* 1939)	CDU/CSU	0.00
12	Paulus	Andreas Paulus (* 1968)	FDP	0.00

Parties

```
judgeDF |> with(table(Vorschlag)) |>
    xtable()
```

	Vorschlag
CDU/CSU	24
FDP	5
Grüne	4
SPD	25

```
judgesRaw |>
    with(table(topic,useNA="always")) |>
    xtable()
```

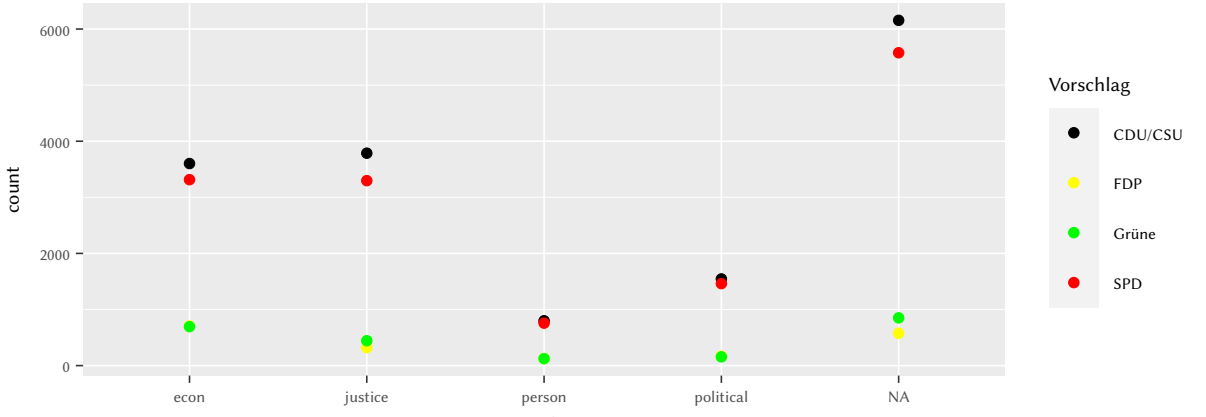
	topic
econ	2200
justice	2336
person	451
political	744
NA.	3329

Parties and Topics

```
jjSucc |>
    rowwise() |>
    reframe(href,accept,topic,data.frame(judge)) |>
    left_join( judgeDF |> select(judge,Vorschlag)) |>
    mutate(topic=factor(topic),Vorschlag=factor(Vorschlag)) -> judgeVS
```

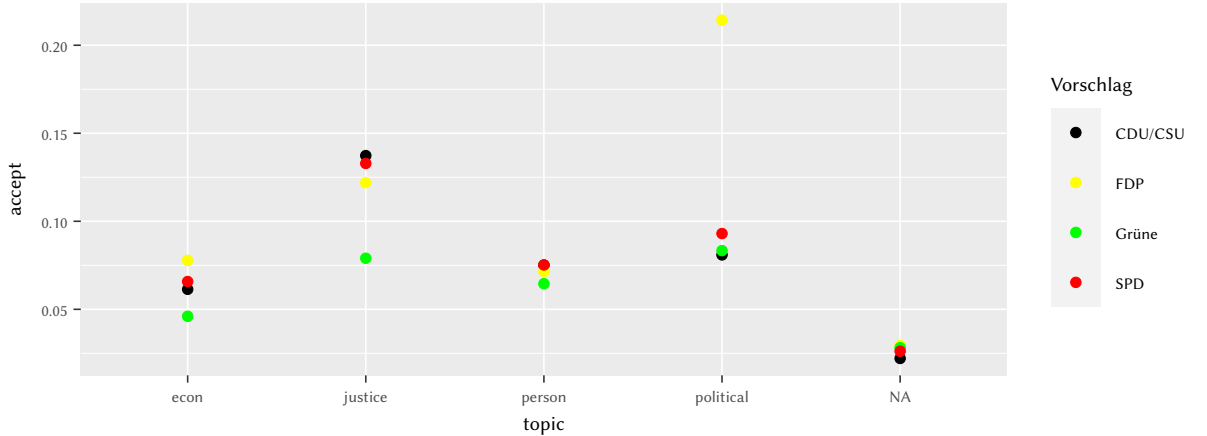
Parties and Topics

```
partyCol <- scale_color_manual(values=c("CDU/CSU"="black","FDP"="yellow","Grüne"="green","SPD"="red")
judgeVS |>
  group_by(Vorschlag,topic) |>
  summarise(count=n()) |>
  ggplot(aes(x=topic,y=count,color=Vorschlag)) + geom_point() + partyCol
```



Parties and Topics

```
judgeVS |>
    group_by(Vorschlag,topic) |>
    summarise(accept=mean(accept)) |>
    ggplot(aes(x=topic,y=accept,color=Vorschlag)) + geom_point() + partyCol
```



Familiarity and acceptance

```
jjSucc |>
    filter(sumx != 0) |>
    mutate(accept = sumx > 1) |>
    select(href,accept) |>
    left_join(bvFamiliarity) -> jjAcceptFam

jjAcceptFam |>
    filter(!is.na(familiarity)) |>
    group_by(href,bvbody,date1) |>
    summarise(familiarity=mean(familiarity),
              accept=mean(accept),
              nJudge=n(),
              .groups="drop") -> jjAcceptFamShort
```

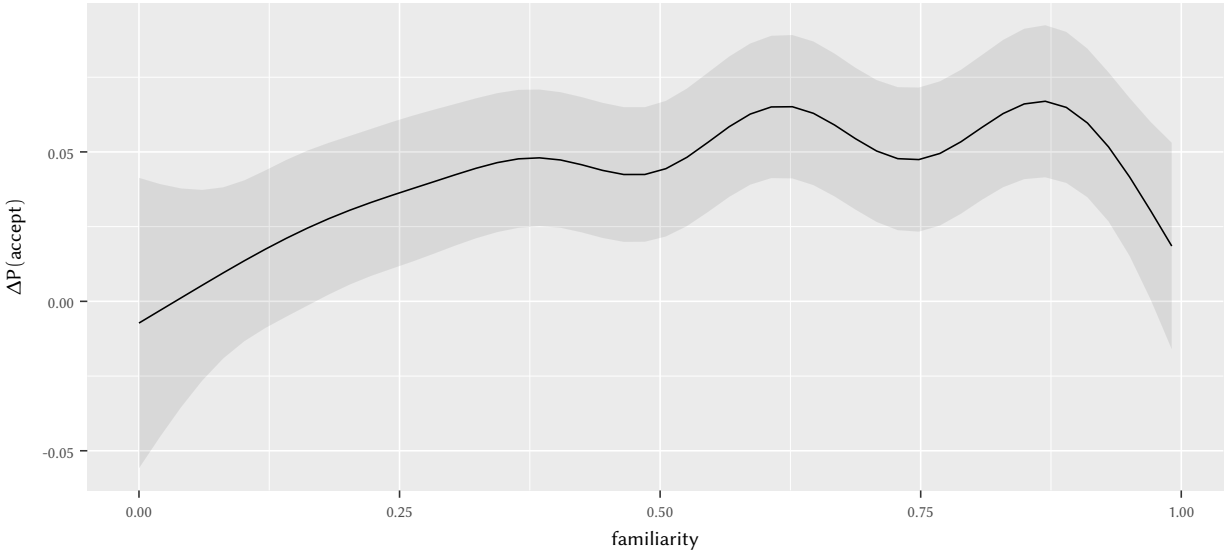
Familiarity and acceptance

```
summary(glm(accept ~ familiarity + nJudge + date1 ,family=binomial,data=jjAcceptFamShort))


Call:
glm(formula = accept ~ familiarity + nJudge + date1, family = binomial,
    data = jjAcceptFamShort)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.846e+00  3.827e-01 -15.277  < 2e-16 ***
familiarity  2.842e+00  3.642e-01   7.803 6.04e-15 ***
nJudge       6.383e-02  2.422e-02   2.635  0.00842 **
date1        1.399e-09  1.782e-10   7.851 4.14e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4030.9  on 6461  degrees of freedom
Residual deviance: 3934.4  on 6458  degrees of freedom
AIC: 3942.4
```

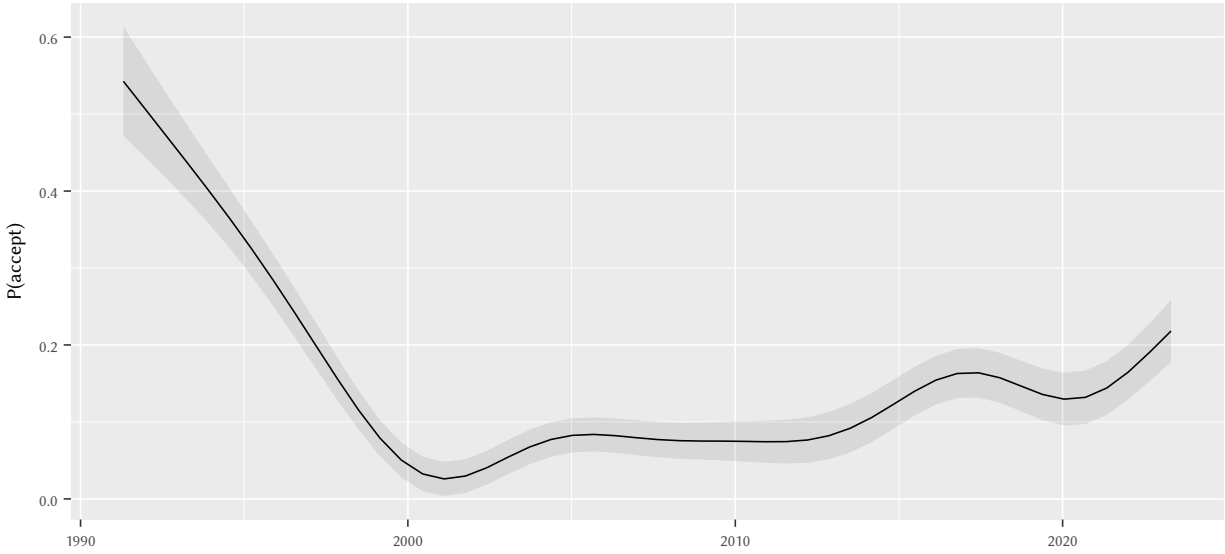
Familiarity...

```
library(mgcv)
library(tidymv)

jjAcceptFam |> mutate(date=as.numeric(date1)) |>
    gam(accept ~ s(familiarity) + s(date) + factor(judge),data=_) -> est
```

```
predict_gam(est,values=list(judge="Papier",date=as.numeric(as.POSIXct("2000-01-01")))) |>
    ggplot(aes(familiarity,fit)) + geom_smooth_ci() + labs(y="$\\Delta P($accept$)$")
```



```
predict_gam(est,values=list(judge="Papier",familiarity=.5)) |>
    ggplot(aes(as.POSIXct(date),fit)) + geom_smooth_ci() + labs(x=NULL,y="P(accept)")
```



```
jjAcceptFam |> mutate(date=as.numeric(date1),bvf=factor(bvbody)) |>
    gam(accept ~ s(familiarity,by=bvf) + s(date),data=_) -> estBody
```

```
predict_gam(estBody,values=list(date=as.numeric(as.POSIXct("2000-01-01")))) |>
ggplot(aes(x=familiarity,y=fit)) + geom_smooth_ci() + facet_wrap(~bvf) +
    coord_cartesian(ylim=c(0,1)) + labs(y="P(accept)")
```