

Forschungspraktikum CSS

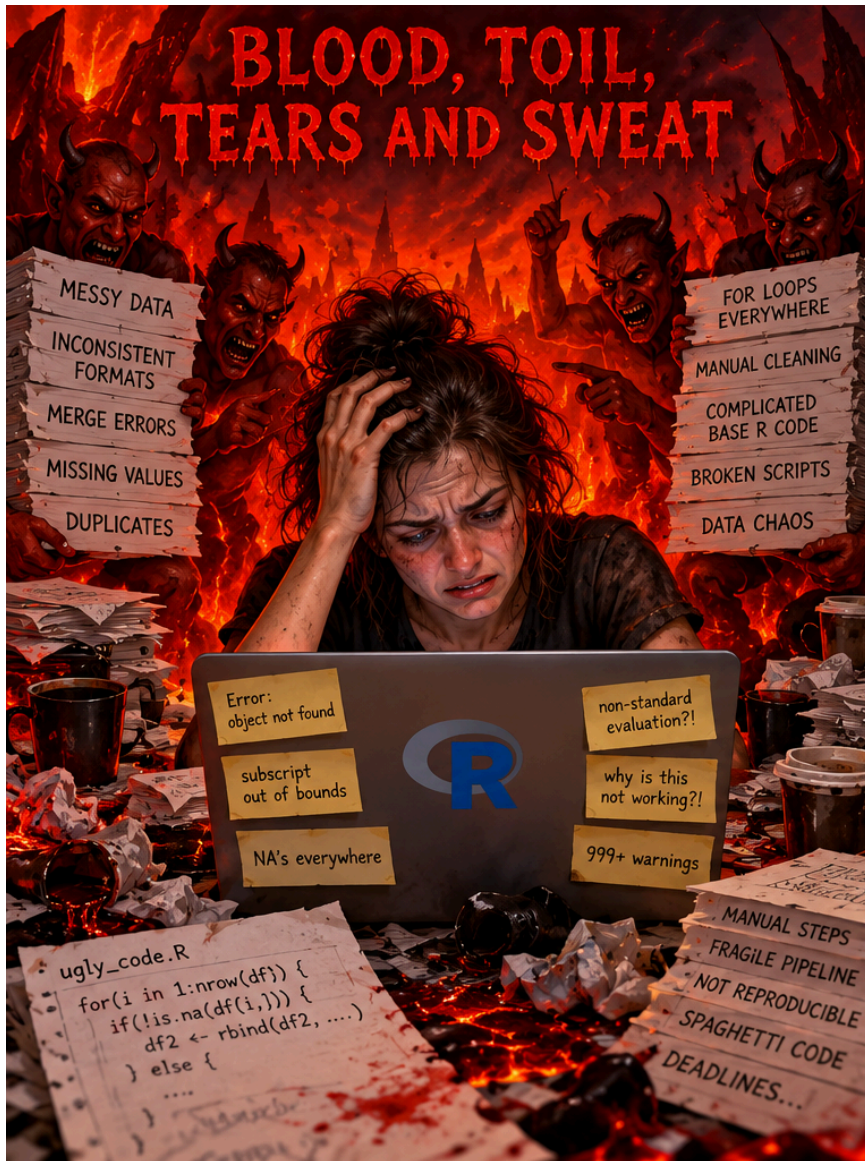
Session 01 · Researching with Style

Dr. Sascha Grehl

2026-10-22

Welcome to the Forschungspraktikum

- **Aim:** To produce a research paper over the next few months
- **Three simple ingredients:**
 - A research idea
 - Data
 - A little work that could be either ...



What makes an analysis manageable?

You already know how to work in R. Today we focus on how to make that work ...

- **Readable:** someone else can follow your code and decisions;
- **Consistent:** the same rule is applied throughout;
- **Repeatable:** a fresh run produces the intended results;
- **Changeable:** new data do not require rewriting everything.

Our guide: the tidyverse style guide.

Three meanings of “tidy”

Level	Review question
Code style	Can others—and my future self—read and understand my code?
Tidy data	Do rows and columns match the unit of observation? Are variable names easy to understand?
Workflow	Can we trace outputs back to inputs and decisions?

The **tidyverse** provides tools. A coherent project still needs deliberate choices.

1 · Keeping projects tidy

Keep a clear project structure

```
1  css_project/  
2  |— css_project.Rproj  
3  |— README.md  
4  |— data/  
5  |   |— raw/  
6  |   └─ derived/  
7  |— output/  
8  |   |— images/  
9  |   └─ pdf/  
10 |— 01_import.R  
11 |— 11_prepare.R  
12 └─ 21_analysis.qmd
```

Note: When working with Git, consider excluding local data and generated output using `.gitignore`.

File names describe jobs

One useful numbering scheme:

- **0x_**: acquire and scrape data, for example
`01_scrape_website1.R`
- **1x_**: prepare data, for example
`11_unsupervised_text_analysis.R`,
`13_generate_final_data.R`
- **2x_**: analyse and report, for example
`21_complete_analysis.qmd`,
`22_final_report.qmd`

Paths and file names

- Avoid personal desktop paths and repeated `setwd()` calls.
- Open the `.Rproj` file to start RStudio in the project folder.
- Use `file.path()` to construct paths with the appropriate directory separator.

```
1 survey_raw <- readr::read_csv(file.path("data", "raw", "mydata.csv"))
```

Please use lowercase file names consistently. My Linux system is case-sensitive, so I prefer *snake_case*.

Give each script an internal structure

1. **Setup:** packages, paths, helper functions.
2. **Parameters:** thresholds, variable selections, analysis decisions.
3. **Import and validation:** load inputs and check assumptions.
4. **Preparation:** create clearly defined analysis data (final data preparation).
5. **Analysis and export:** compute and save results.

Tip: When a decision changes, update the relevant step and **rerun the complete workflow**.

Internal structure: Setup

```
1 rm(list = ls()) # clear the workspace
2 needs::needs(dplyr, readr, ggplot2) # load the required packages
3
4 path_output <- "output"
5 path_data <- "data"
6 path_functions <- "functions"
7
8 # Load the required helper functions.
9 source(file.path(path_functions, "functions_scraping.R"))
```

What is the difference between `needs()` and `needs::needs()`?

Internal structure: Parameters

```
1 # General settings -----
2 min_age <- 18
3
4 # Data settings -----
5 model    <- "rf"
6 version  <- 8
7
8 # Variables -----
9 var_Y <- "news_hours"
10 vars_X_baseline <- c("age", "city")
11 vars_X_control  <- c("edu_level", "has_children")
12 formula_m1 <- reformulate( # news_hours ~ age + city
13   termlabels = vars_X_baseline,
14   response   = var_Y
15 )
16 # Plots -----
17 ggplot_option <- "viridis"
```

Internal structure: Import

```
1 # Fixed file name
2 survey_raw <- readr::read_csv(file.path(path_data, "survey_raw.csv"))
3
4 # Dynamic file name: sentiment_text_rf_8.csv
5 sentiment_text <- read.csv(file = file.path(path_data, paste0(
6   "sentiment_text_",
7   model,
8   "_",
9   version,
10  ".csv"
11  )))
```

Internal structure: Preparation

```
1 survey_clean <- survey_raw |>
2   mutate(
3     ...
4   ) |>
5   left_join(...) |>
6   filter(...)
```

Keep data preparation in this section rather than making ad hoc changes later. Some preliminary exploration may be necessary to decide which transformations to apply.

Internal structure: Analysis

```
1 survey_clean |>  
2   lm(formula = formula_m1, data = _)
```

The fresh-session test

1. Save the files.
2. Restart R without restoring an old workspace.
3. Run the complete script or render the report from its inputs.
4. Inspect outputs and warnings.

Disable automatic `.RData` restoration and workspace saving in RStudio.

2 · Readable code and reuse rules

Using Pipes and spacing

Hard to review:

```
1 d<-mutate(filter(survey_raw,age>=18), h=news_minutes/60)
```

Easier to review:

```
1 minutes_per_hour <- 60
2
3 survey_adults <- survey_raw |>
4   filter(age >= min_age) |>
5   mutate(news_hours = news_minutes / minutes_per_hour)
```

Use:

- Pipes (`|>` or `%>%`)
- Spacing
- Descriptive names in `snake_case`

Separate sources and derived values

Repeatedly running this code changes the meaning of the data:

```
1 survey <- survey |>  
2   mutate(x = x * 100)
```

Preserve the source or rename the variable to make the transformation repeatable.

```
1 survey <- survey |>  
2   mutate(x_percent = x * 100)
```

or

```
1 survey_clean <- survey |>  
2   mutate(x = x * 100)
```

DRY

Don't Repeat Yourself: avoid maintaining the same rule independently in several places.

- A repeated threshold becomes a named parameter.
- A repeated transformation becomes a function.
- The same task across several inputs becomes an iteration.

LLMs are your friends (slaves?)

Example request:

Refactor the following code into a function: [code]. Expected input: [description].
Expected output: [description].

- Review the code and use it only if you understand every line.
- Explain the code to me line by line.
- You remain responsible for the code.

Functions and problematic values

```
1 is_media_consumption_low_bad <- function(minutes) {  
2   minutes %in% 0:60  
3 }  
4  
5 is_media_consumption_low_bad(c(30, 90, 10.5, NA, -4, ""))
```

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

- Do missing values remain missing?
- What happens with empty strings, fractional values or invalid values?

Functions and problematic values II

```
1 is_media_consumption_low <- function(minutes) {  
2   if (!is.numeric(minutes)) {stop("Input must be numeric.") }  
3   ifelse(  
4     is.na(minutes) | !is.finite(minutes) | minutes < 0,  
5     NA,  
6     minutes <= 60  
7   )  
8 }  
9  
10 is_media_consumption_low(c(30, 90, 10.5, NA, -4)) # <- "" no longer possible
```

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

LLMs are your friends II

Example request:

Refactor this code into a function: [code]. Input: a numeric vector of minutes. Output: hours of the same length, preserving missing values. Do not use global objects. Include checks for ordinary, missing and empty inputs. Explain the code to me line by line.

3 · Quality and collaboration

Comments explain decisions

Little added value:

```
1 # Filter by age.  
2 filter(survey_raw, age >= 18)
```

Useful context:

```
1 # Target population: adults; eligibility defined before analysis.  
2 filter(survey_raw, age >= min_age)
```

Document **why, units, sources and exceptions**.

Variable names

For binary variables (0/1), good variable names should make it immediately clear what is being measured and what a value of 1 represents. Ideally, the meaning of 0 should also be obvious.

Bad

```
1 status
2 treatment
3 outcome
4 bla2
```

Good

```
1 is_married
2 has_children
3 did_vote
4 received_treatment
5 passed_exam
```

Transformed variables

For a technical transformation that does not introduce a new substantive concept, add a descriptive suffix to the variable name.

```
1 survey_raw |>
2   mutate(
3     age_squared = age^2,
4     age_z = as.numeric(scale(age)),
5     news_minutes_div60 = news_minutes / 60 # Prefer news_hours for this
6   )
```

Automate formatting and code review

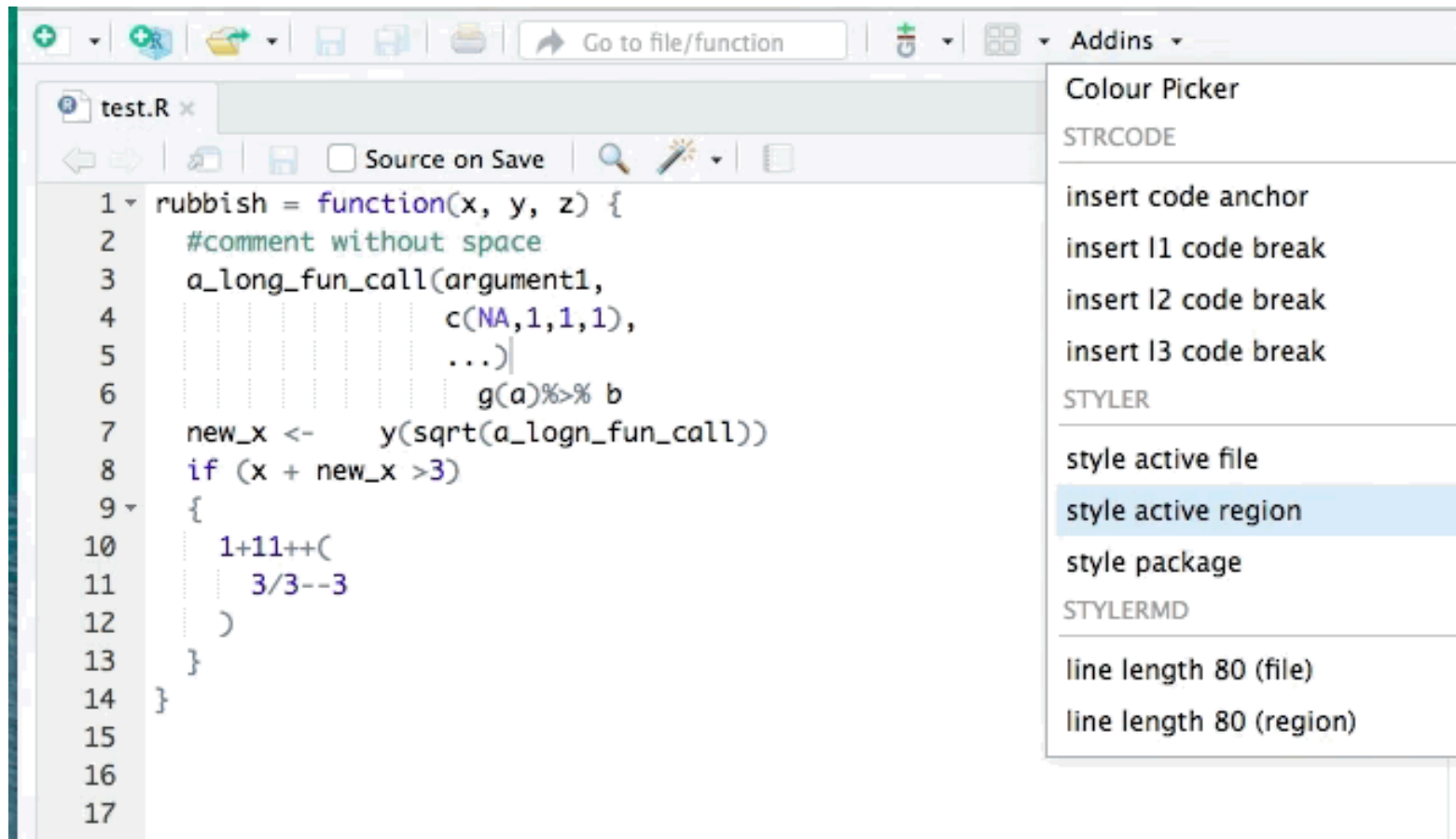
Run deliberately on an existing project file:

```
1 styler::style_file("11_prepare.R")
2 lintr::lint("11_prepare.R")
```

- **styler** rewrites formatting.
- **lintr** flags issues using configurable rules.

Tidyverse Style Guide

Styler



The styler add-in reformats R code in RStudio.

Same code different results

R 3.5.3

```
1 set.seed(123)
2 sample(1:10, 5)
```

```
[1] 3 8 4 7 6
```

R 3.6.0 and later

```
1 set.seed(123)
2 sample(1:10, 5)
```

```
[1] 3 10 2 8 6
```

- **Reason:** The `sample()` function (or a function it depends on) was updated between R versions, changing its behavior.
- **Consequence:** Different environments may produce different results, compromising the reproducibility of your research.

Preserve information about the environment

To ensure reproducibility, it is good practice to save the output of `sessionInfo()` after data collection and before starting your analysis. This documents the R version, operating system, and package versions used in your research.

```
1 timestamp <- format(Sys.time(), "%Y-%m-%d_%H-%M-%S")
2
3 capture.output(
4   sessionInfo(),
5   file = file.path(
6     "output",
7     "session_info",
8     paste0("session_info_", timestamp, ".txt")
9   )
10 )
```

Example sessionInfo()

```
1 R version 4.5.2 (2025-10-31)
2 Platform: x86_64-pc-linux-gnu
3 Running under: Ubuntu 26.04.1 LTS
4
5 [...]
6
7 attached base packages:
8 [1] stats      graphics  grDevices utils      datasets  methods   base
9
10 other attached packages:
11 [1] socviz_2.0.0    fs_2.1.0      lubridate_1.9.5 forcats_1.0.1 stringr_1.6.0
12 [6] dplyr_1.2.1     purrr_1.2.2   readr_2.2.0     tidyr_1.3.2   tibble_3.3.1
13 [11] ggplot2_4.0.3  tidyverse_2.0.0
14
15 loaded via a namespace (and not attached):
16 [1] gtable_0.3.6      compiler_4.5.2  tidyselect_1.2.1 dichromat_2.0-0.1
17 [5] scales_1.4.0      R6_2.6.1        generics_0.1.4   pillar_1.11.1
18 [9] RColorBrewer_1.1-3 tzdb_0.5.0      rlang_1.2.0      stringi_1.8.7
```

Preserve the package environment II

Run deliberately when setting up or maintaining a project:

```
1 renv::init()  
2 renv::snapshot()
```

This will automatically scan all your R scripts for used libraries and stores these information about your R packages in the `renv.lock` file.

In a copy of the project on another machine one can run

```
1 renv::restore()
```

Now it downloads and installs the recorded package versions, or retrieves them from an available package cache.

This does not solve all problems (e.g. R version, operating system) but most of them

4 · Text mining

Three perspectives on text mining

Perspective	Main idea
Discovery	Extract information across written resources to discover something previously unknown.
Pattern finding	Look for regularities in text, as data mining looks for patterns in data.
Methods	Use linguistic, statistical and machine-learning techniques to structure textual information.

Text mining connects a **research question**, a **text collection** and an **explicit analytical procedure**.

Why analyse text?

- Sources include news websites, social media, medical notes and scanned printed documents.
- Text contains information that is not already organised as analytical variables.
- Industry applications include search, translation, sentiment analysis and question answering.
- Research applications include cognitive modelling, language bias and systematic literature reviews.

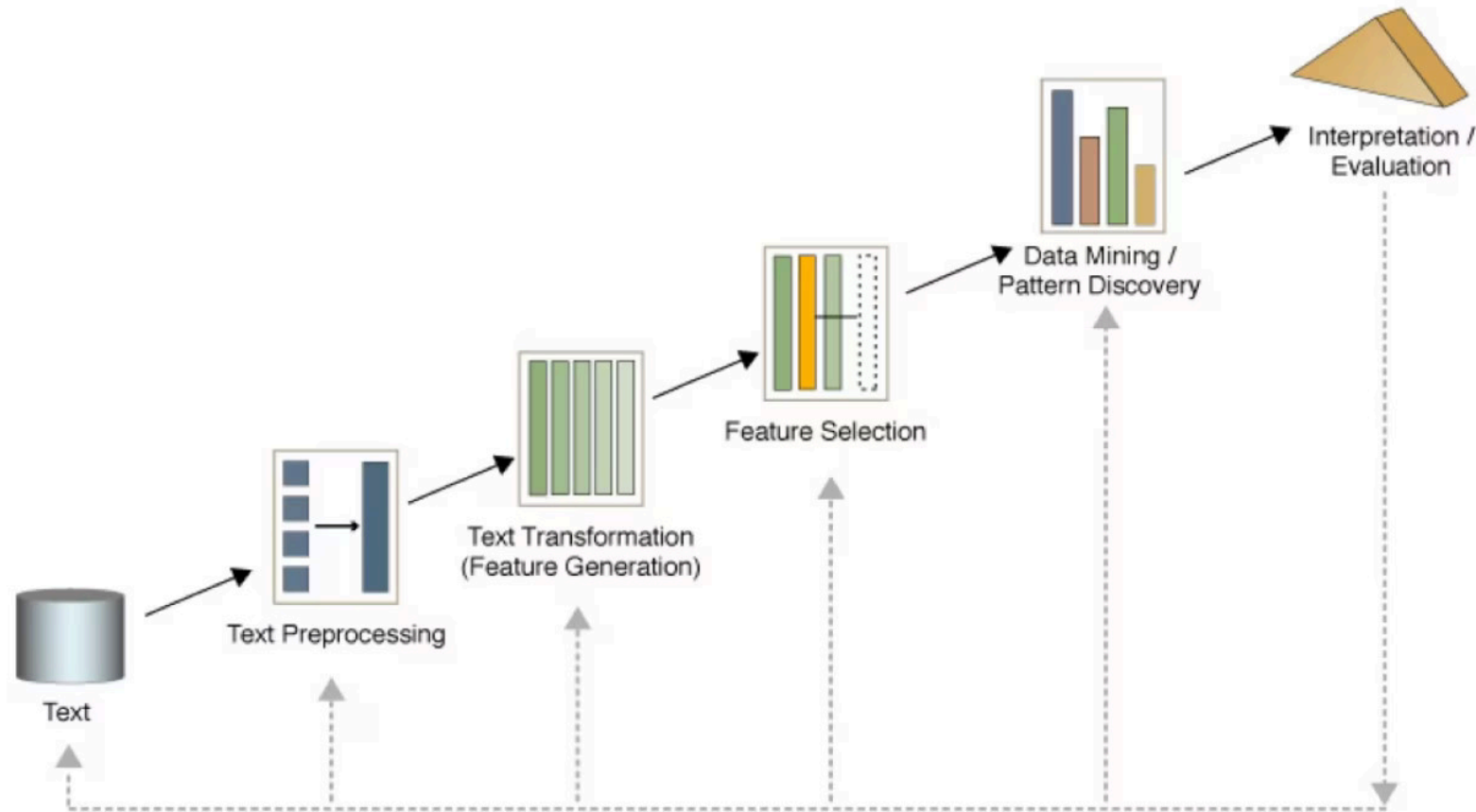
Scale makes automated analysis useful; language makes it difficult.

What makes text difficult?

Challenge	Source example or consequence
High dimensionality	Many possible words and phrases
Relationships	“Jumbo merges with Hema” versus “Jumbo is bought by Hema”
Synonyms	car, automobile, vehicle: related, but not identical in every context
Ambiguity	Dutch <i>kapsalon</i> : hair salon or a fast-food dish
Homographs	<i>bat</i> : animal or sports equipment

Also consider misspellings, abbreviations, negation and spelling variants.

The text-mining workflow



Text collection, preprocessing, feature generation, feature selection, pattern discovery, and interpretation or evaluation, with feedback to earlier stages.

A range of NLP tasks

Task family	Examples from the source
Structure and prediction	feature selection, classification, clustering, topic modelling
Attitudes and responsibility	sentiment analysis, bias detection, responsible text mining
Generating or transforming text	summarisation, translation
Supporting interaction and research	question answering, chatbots, automatic reviewing with ASReview

Each task requires its own definition of a useful result.

Before modelling

Start with the basics of text processing.

5 · Regular expressions

What is regex

- **Powerful** and a very useful tool for text (pre)processing
- Used in pretty much **every pipeline** involving text
- Typical applications:
 - Extract numbers, email-like strings or IP-address candidates.
 - Validate the shape of text entered into an interface.
 - Reformat dates into a consistent representation.
 - Find names or addresses for a reviewed pseudonymisation workflow.
- Cryptic & takes a lot of practice!

renv 1.3.1

Get started

Reference

Articles ▾

Changelog

Search fo



renv

Overview

The `renv` package¹ helps you create reproducible environments for your R projects. Use `renv` to make your R projects more isolated, portable and reproducible.

- **Isolated:** Installing a new or updated package for one project won't break your other projects, and vice versa. That's because `renv` gives each project its own private library.
- **Portable:** Easily transport your projects from one computer to another, even across different platforms. `renv` makes it easy to install the packages your project depends on.
- **Reproducible:** `renv` records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

Installation

Install the latest version of `renv` from CRAN with:

```
install.packages("renv")
```

Alternatively, install the development version from [r-universe](#) with:

```
install.packages("renv", repos = "https://rstudio.r-universe.dev")
```

LINKS

[View on CRAN](#)
[Browse source code](#)
[Report a bug](#)

LICENSE

[MIT](#) + file [LICENSE](#)

CITATION

[Citing `renv`](#)

DEVELOPERS

Kevin Ushey
Author, maintainer 
[Hadley Wickham](#)
Author 
 **posit**
Copyright holder, funder

Assume we want to scrape the developers.

Website - As HTML

[illegible]

We usually end up with something like this.

We want to extract this information

```
1 library(stringr)
2 str_extract(html, regex(r"(<div class=\"developers\">.*?</div>)", dotall = TRUE))
```

```
[1] "<div class=\"developers\">\n<h2 data-toc-skip>Developers</h2>\n<ul class=\"list-
unstyled\">\n<li>Kevin Ushey <br><small class=\"roles\"> Author, maintainer </small> <a
href=\"https://orcid.org/0000-0003-2880-7407\" target=\"orcid.widget\" aria-label=\"ORCID\"
class=\"external-link\"><span class=\"fab fa-orcid orcid\" aria-hidden=\"true\"></span></a>
</li>\n<li>\n<a href=\"https://hadley.nz\" class=\"external-link\">Hadley Wickham</a> <br>
<small class=\"roles\"> Author </small> <a href=\"https://orcid.org/0000-0003-4757-117X\"
target=\"orcid.widget\" aria-label=\"ORCID\" class=\"external-link\"><span class=\"fab fa-
orcid orcid\" aria-hidden=\"true\"></span></a>  </li>\n<li>\n<a href=\"https://www.posit.co\"
class=\"external-link\"><img src=\"https://www.tidyverse.org/posit-logo.svg\" alt=\"Posit\"
height=\"16\" width=\"62\" style=\"margin-bottom: 3px;\"></a> <br><small class=\"roles\">
Copyright holder, funder </small>  </li>\n</ul>\n</div>"
```

Literal searches and alternatives

Regex	Matches
<code>spain</code>	that literal substring
<code>[abc]</code>	one of a, b or c
<code>[sS]pain</code>	spain or Spain
<code>spain Spain</code>	either complete alternative

Square brackets `[]` select **one character**; `|` separates alternatives.

Character ranges and negated classes

Regex	Meaning
[A-Z]	one uppercase ASCII letter
[a-z]	one lowercase ASCII letter
[0-9]	one ASCII digit
[^A-Z]	one character that is not an uppercase ASCII letter
[^Ss]	one character other than S or s
[^.]	one character other than a literal period

Inside `[]`, a leading `^` negates the class; otherwise it can be literal.

Optional elements and wildcards

Regex element	The preceding character or pattern can occur	regex	pattern
<code>?</code>	never or once	<code>colou?r</code>	<code>color</code> or <code>colour</code>
<code>*</code>	never or more	<code>colou*r</code>	<code>color</code> or <code>colour</code> , <code>colouur</code> , ...
<code>+</code>	once or more	<code>colou+r</code>	<code>colour</code> , <code>colouur</code> , ...
<code>.</code>	wildcard , any single character	<code>beg.n</code>	<code>begin</code> , <code>began</code> , <code>beg1n</code> , <code>begXn</code> , ...

Repetition

Regex element	The preceding character or pattern occur	regex
<code>{n}</code>	exactly n times	<code>[0-9]{4}</code>
<code>{n,m}</code>	between n and m times	<code>[0-9]{2,4}</code>
<code>{n,}</code>	at least n times	<code>[0-9]{2,}</code>

A quantifier applies to the preceding character, class or group.

Anchors and word boundaries

regex	Typical use
<code>^dog</code>	starts with dog
<code>dog\$</code>	ends with dog
<code>^dog\$</code>	dog as the entire single-line input
<code>\bdog\b</code>	dog at word boundaries
<code>\B</code>	a position that is not a word boundary

```
1 stringr::str_detect(c("dog", "dogs", "hotdog", "(dog)"), r"^dog$")
```

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

```
1 stringr::str_detect(c("dog", "dogs", "hotdog", "(dog)"), r"^dog")
```

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

Strings and raw strings

In **R** the backslash `\` is an **escape character**. To represent a literal backslash in a regular string, we must write `\\`

Therefore, to search for `\bdog\b`, we need to write `\\bdog\\b`.

In more complex regular expressions, these additional backslashes can make code difficult to read. **Raw strings** provide a simpler alternative.

- Regular string: `"my_string"`
- Raw string: `r"(my_string)"` Raw strings begin with `r` and use parentheses (either `(, [` or `{`) to delimit their content. Backslashes are interpreted literally.

```
1 str1 <- "\\bdog\\b"
2 str2 <- r"(\bdog\b)"
3 identical(str1,str2)
```

```
[1] TRUE
```

Using raw strings makes regular expressions easier to read, write, and maintain.

Common character classes

Regex	Meaning
<code>\d / \D</code>	digit / non-digit
<code>\w / \W</code>	word character / non-word character
<code>\s / \S</code>	whitespace / non-whitespace

R examples: `"\\d"`, `"\\w"`, `"\\s"` or `r"(\d)"`, `r"(\w)"`, `r"(\s)"`

Use `[0-9]` when the requirement is specifically **ASCII digits**.

Escaping special characters

Intended match	Regex
a literal asterisk	<code>*</code>
a literal period	<code>\.</code>
a literal question mark	<code>\?</code>
newline	<code>\n</code>
tab	<code>\t</code>

Attention: Inside `[]` the special characters work normal (see next slide)

Escaping special characters - Examples

```
1 stringr::str_extract_all("Dr. Livingstone, I presume?", ".")
```

```
[[1]]  
[1] "D" "r" "." " " "L" "i" "v" "i" "n" "g" "s" "t" "o" "n" "e" ", " " " " "I" " " "  
[20] "p" "r" "e" "s" "u" "m" "e" "?"
```

```
1 stringr::str_extract_all("Dr. Livingstone, I presume?", "\\.")
```

```
[[1]]  
[1] "."
```

```
1 stringr::str_extract_all("Dr. Livingstone, I presume?", "[.]")
```

```
[[1]]  
[1] "."
```

Grouping and operator precedence

Typical binding order, strongest first:

1. Parentheses: `()`.
2. Quantifiers: `*`, `+`, `?`, `{}`.
3. Sequences and anchors.
4. Alternatives: `|`.

```
1 stringr::str_detect(c("cat", "dog", "catfish"), "^(cat|dog)$")
```

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

```
1 stringr::str_detect(c("cat", "dog", "catfish"), "^cat|dog$")
```

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

Grouping determines whether an anchor applies to both alternatives.

Read these regex examples

Regex	Possible matches
hello	hello as a substring
gray grey	gray, grey
gr(a e)y	gray, grey
gr[ae]y	gray, grey
b[aeiou]bble	babble, bebble, bibble, bobble, bubble
[bchm-pP]at ot	bat, cat, hat, mat, nat, oat, pat, Pat, ot
colou?r	color, colour
(.*?)	everything that appears inside of () like (), (hat) or (sic!)

Regex practice

- Use **Regex Crossword** for pattern-reading practice.

References and further practice

- **Tidyverse Style Guide**: files, syntax, functions and pipes.
- **Tidy Data**: observations and variables.
- **purrr**: map and **walk**: iteration.
- **here**: declare the **project root**: portable paths.
- **renv introduction**: package environments.
- **stringr** regex reference.