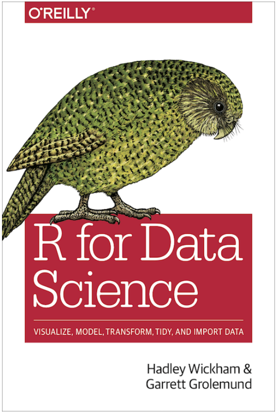
Socio-Informatics 348

Data Transformation
Regex

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading


O'REILLY
R for Data Science
VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA
Hadley Wickham & Garrett Grolemund

R for Data Science, Chapter 15

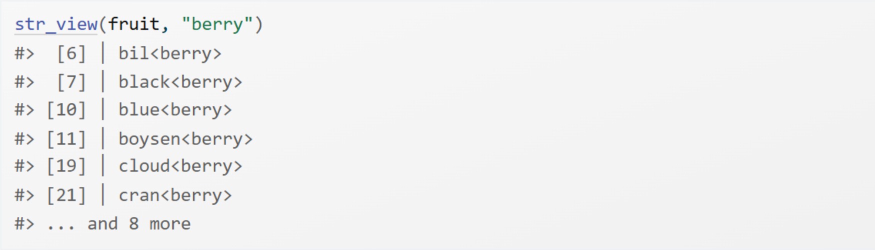
Why Regular Expressions?

- Regular expressions (regex/regexp): concise, powerful language for string patterns.
- Use cases: extract, detect, count, replace text.

Pattern Basics

Use `str_view()` **to do basic pattern matching.**

- **Literal characters:** match themselves exactly (e.g., `berry`).

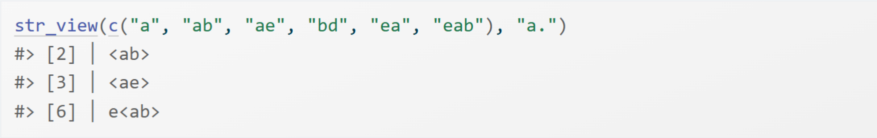


```
str_view(fruit, "berry")
#>  [6] │ bil<berry>
#>  [7] │ black<berry>
#> [10] │ blue<berry>
#> [11] │ boysen<berry>
#> [19] │ cloud<berry>
#> [21] │ cran<berry>
#> ... and 8 more
```

- **Metacharacters:** Punctuation and special meanings
- **Quantifiers:** How many times can a pattern match?

Pattern Basics

- **Literal characters:** match themselves exactly (e.g., `berry`).
- **Metacharacters:** Punctuation and special characters `.` `+` `*` `[` `]` `?` etc.
- **Dot (`.`):** any single character.

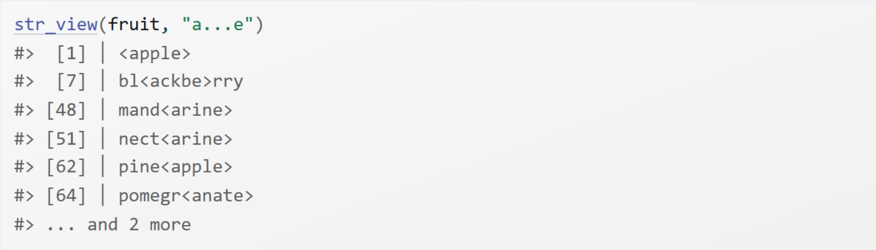


```
str_view(c("a", "ab", "ae", "bd", "ea", "eab"), "a.")
#> [2] │ <ab>
#> [3] │ <ae>
#> [6] │ e<ab>
```

- **Quantifiers:** How many times can a pattern match?

Pattern Basics

- **Literal characters:** match themselves exactly (e.g., `berry`).
- **Metacharacters:** Punctuation and special characters `.` `+` `*` `[` `]` `?` etc.
- **Dot (`.`):** any single character.

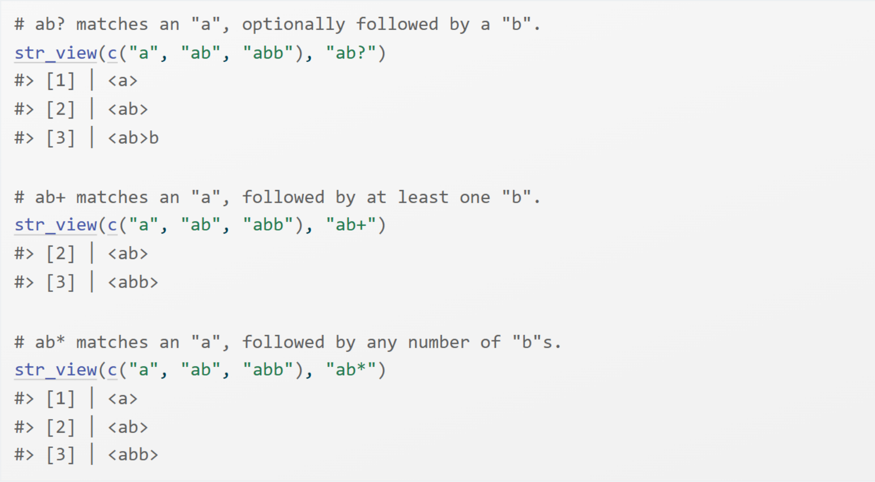


```
str_view(fruit, "a...e")
#>  [1] │ <apple>
#>  [7] │ bl<ackbe>rry
#> [48] │ mand<arine>
#> [51] │ nect<arine>
#> [62] │ pine<apple>
#> [64] │ pomegr<anate>
#> ... and 2 more
```

- **Quantifiers:** How many times can a pattern match?

Pattern Basics

- **Quantifiers:** How many times can a pattern match?
  ? optional, + one or more, ∗ zero or more.



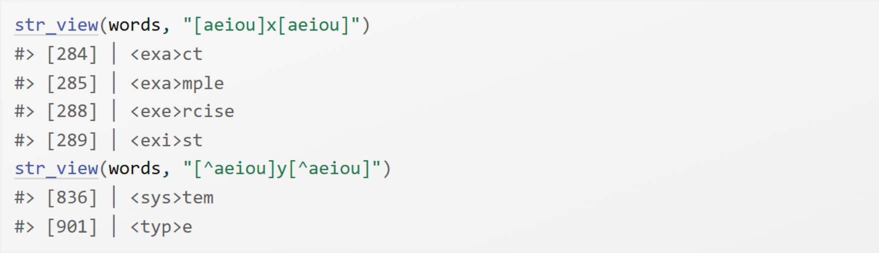
```
# ab? matches an "a", optionally followed by a "b".
str_view(c("a", "ab", "abb"), "ab?")
#> [1] │ <a>
#> [2] │ <ab>
#> [3] │ <ab>b

# ab+ matches an "a", followed by at least one "b".
str_view(c("a", "ab", "abb"), "ab+")
#> [2] │ <ab>
#> [3] │ <abb>

# ab* matches an "a", followed by any number of "b"s.
str_view(c("a", "ab", "abb"), "ab*")
#> [1] │ <a>
#> [2] │ <ab>
#> [3] │ <abb>
```

Character Sets and Alternation

- Character classes are defined by square brackets: `[abcd]`.
- Find the words containing an "x" surrounded by vowels, or a "y" surrounded by consonants:

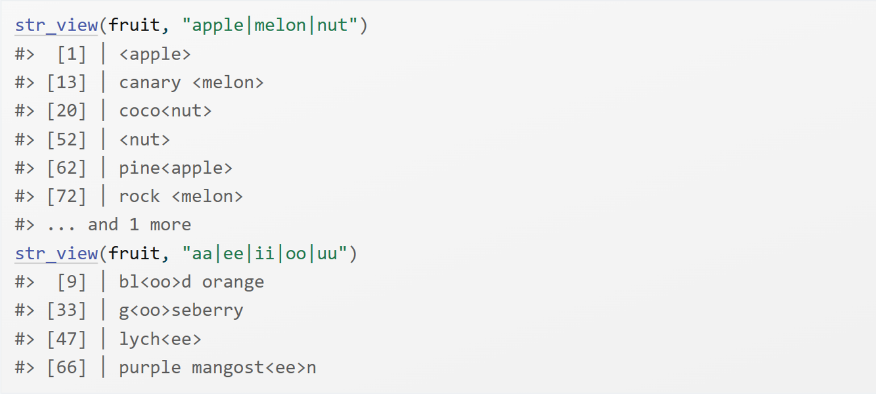


```
str_view(words, "[aeiou]x[aeiou]")
#> [284] │ <exa>ct
#> [285] │ <exa>mple
#> [288] │ <exe>rcise
#> [289] │ <exi>st
str_view(words, "[^aeiou]y[^aeiou]")
#> [836] │ <sys>tem
#> [901] │ <typ>e
```

- Note: invert the match with ˆ inside brackets: `[ˆaeiou]`.

Character Sets and Alternation

- Alternation is defined by vertical bar: `a|b|c`.
- Look for fruits containing ”apple”, ”melon”, or ”nut”, or a repeated vowel:



```
str_view(fruit, "apple|melon|nut")
#>  [1] │ <apple>
#> [13] │ canary <melon>
#> [20] │ coco<nut>
#> [52] │ <nut>
#> [62] │ pine<apple>
#> [72] │ rock <melon>
#> ... and 1 more
str_view(fruit, "aa|ee|ii|oo|uu")
#>  [9] │ bl<oo>d orange
#> [33] │ g<oo>seberry
#> [47] │ lych<ee>
#> [66] │ purple mangost<ee>n
```

Key Regex Functions

- `str_detect()` – logical vector.
- `str_subset()` – return matches.
- `str_which()` – positions.
- `str_count()` – number of matches.
- `str_replace()`, `str_replace_all()`.
- `separate_wider_regex()` – extract to columns.

str_detect()

- str_detect(string, pattern) returns TRUE/FALSE if pattern is found in string.

```
str_detect(c("a", "b", "c"), "[aeiou]")
#> [1]  TRUE FALSE FALSE
```

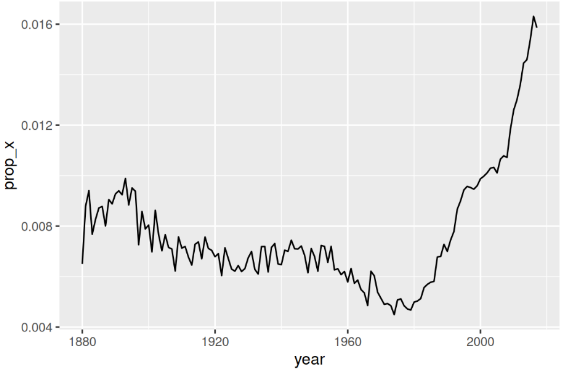
- Useful for filtering with filter().

```
babynames |>
  filter(str_detect(name, "x")) |>
  count(name, wt = n, sort = TRUE)
```

str_detect()

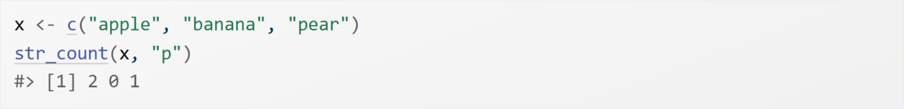
- useful with sum() and mean().

```
babynames |>
  group_by(year) |>
  summarize(prop_x = mean(str_detect(name, "x"))) |>
  ggplot(aes(x = year, y = prop_x)) +
  geom_line()
```



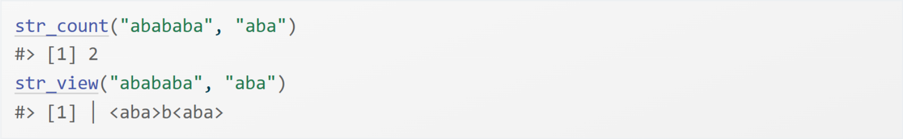
`str_count()`

- `str_count(string, pattern)` counts occurrences of pattern in string.



```
x <- c("apple", "banana", "pear")
str_count(x, "p")
#> [1] 2 0 1
```

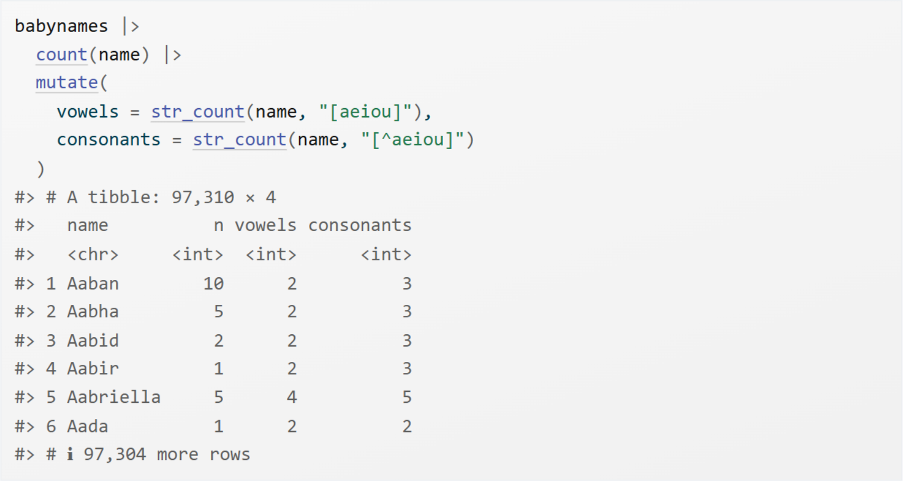
- Note that patterns cannot overlap.



```
str_count("abababa", "aba")
#> [1] 2
str_view("abababa", "aba")
#> [1] │ <aba>b<aba>
```

str_count()

- Use with mutate() to create new variables.



```
babynames |>
  count(name) |>
  mutate(
    vowels = str_count(name, "[aeiou]"),
    consonants = str_count(name, "[^aeiou]")
  )
#> # A tibble: 97,310 × 4
#>   name          n vowels consonants
#>   <chr>     <int>  <int>      <int>
#> 1 Aaban        10      2          3
#> 2 Aabha         5      2          3
#> 3 Aabid         2      2          3
#> 4 Aabir         1      2          3
#> 5 Aabriella     5      4          5
#> 6 Aada          1      2          2
#> # ℹ 97,304 more rows
```

- What do you notice about the number of vowels?

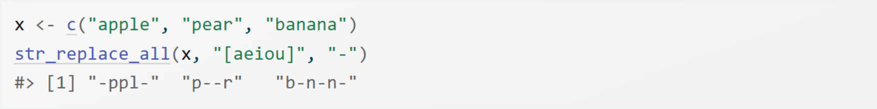
`str_count()`

Solve the issue:

- Include uppercase vowels to character class `str_count(name, "[aeiouAEIOU]")`.
- Tell the function to ignore case `str_count(name, regex("[aeiou]", ignore_case = TRUE))`.
- Convert names to lower case before counting `str_count(str_to_lower(name), "[aeiou]")`.

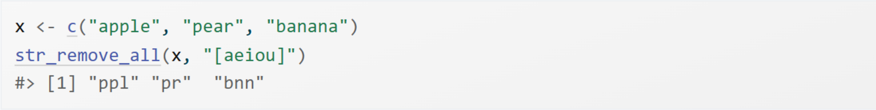
str_replace() and str_remove()

- Replace first match:
  str_replace(string, pattern, replacement)
- Replace all matches:
  str_replace_all(string, pattern, replacement)



```
x <- c("apple", "pear", "banana")
str_replace_all(x, "[aeiou]", "-")
#> [1] "-ppl-"  "p--r"   "b-n-n-"
```

- Remove matches: set replacement to empty string "" OR use str_remove() and str_remove_all().

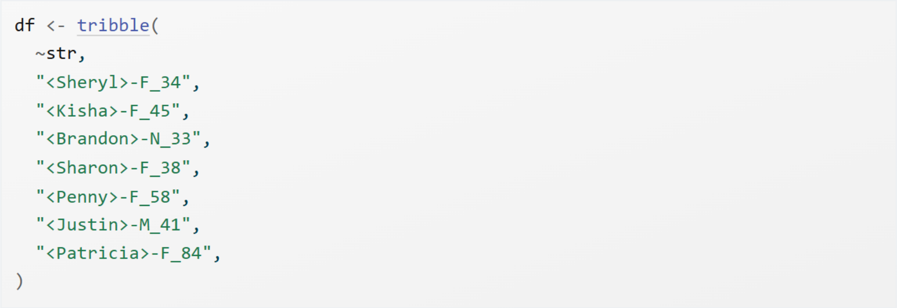


```
x <- c("apple", "pear", "banana")
str_remove_all(x, "[aeiou]")
#> [1] "ppl" "pr"  "bnn"
```

- Very useful for data cleaning with mutate().

Extract variables with separate_wider_regex()

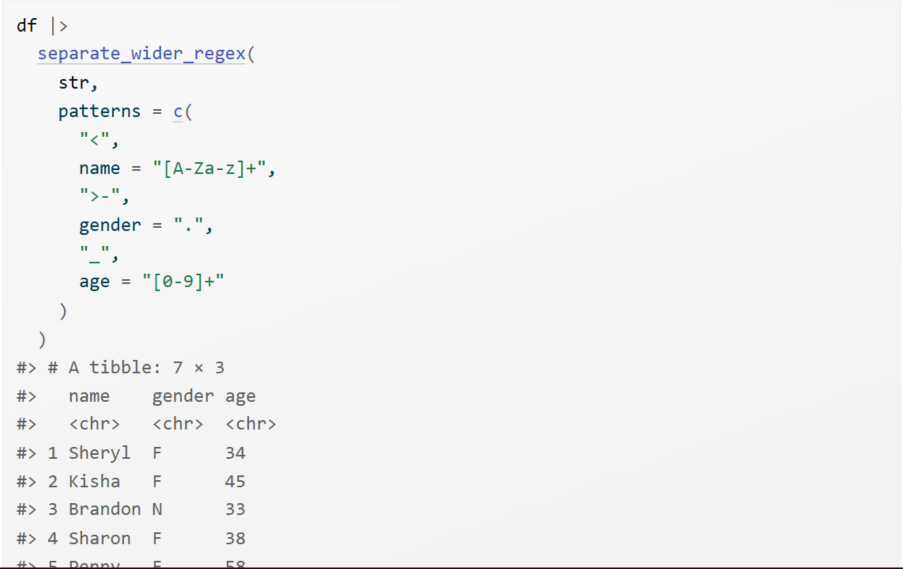
- Similar to separate_wider_position and separate_wider_delim.



```
df <- tribble(
  ~str,
  "<Sheryl>-F_34",
  "<Kisha>-F_45",
  "<Brandon>-N_33",
  "<Sharon>-F_38",
  "<Penny>-F_58",
  "<Justin>-M_41",
  "<Patricia>-F_84",
)
```

Extract variables with separate_wider_regex()

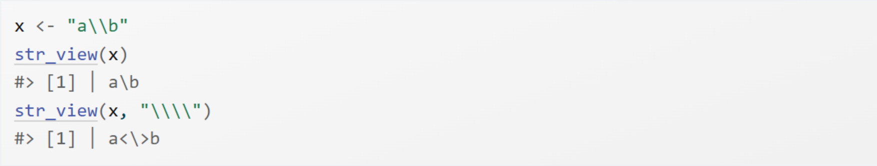
- Similar to separate_wider_position and separate_wider_delim.



```
df |>
  separate_wider_regex(
    str,
    patterns = c(
      "<",
      name = "[A-Za-z]+",
      ">-",
      gender = ".",
      "_",
      age = "[0-9]+"
    )
  )
#> # A tibble: 7 × 3
#>   name    gender age
#>   <chr>   <chr>  <chr>
#> 1 Sheryl  F      34
#> 2 Kisha   F      45
#> 3 Brandon N      33
#> 4 Sharon  F      38
#> 5 Penny   F      58
```

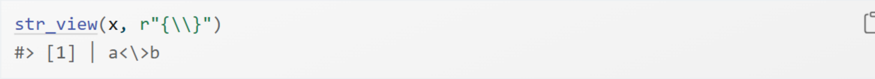
Escaping

- Escape metacharacters with backslash – can get tricky!

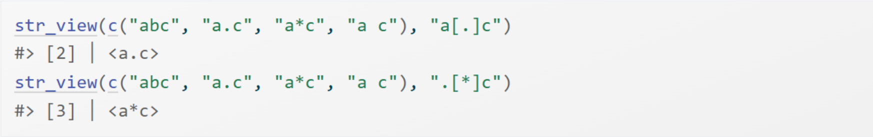


```
x <- "a\\b"
str_view(x)
#> [1] │ a\b
str_view(x, "\\\\")
#> [1] │ a<\>b
```

- Alternative approaches – raw strings and character classes.



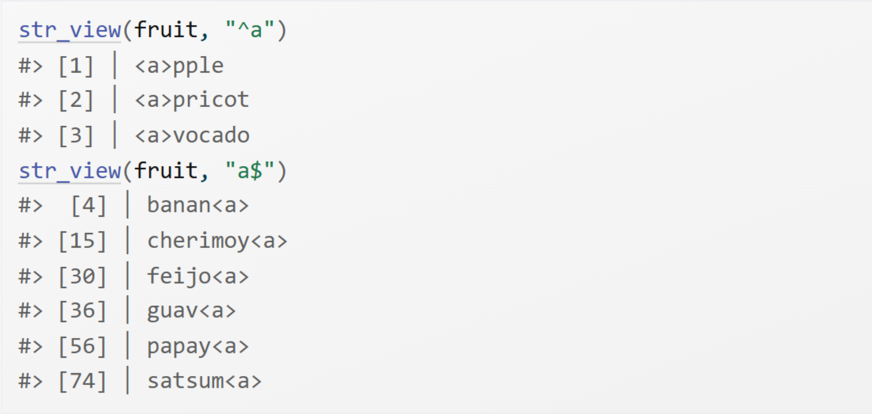
```
str_view(x, r"{\\}")
#> [1] │ a<\>b
```



```
str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c")
#> [2] │ <a.c>
str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c")
#> [3] │ <a*c>
```

Anchors and Word Boundaries

- Anchors: ˆ (start), $ (end).

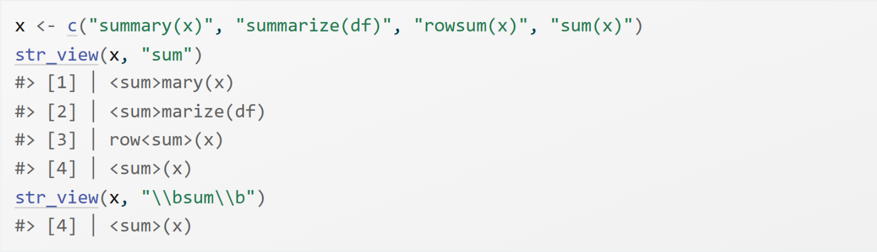


```
str_view(fruit, "^a")
#> [1] │ <a>pple
#> [2] │ <a>pricot
#> [3] │ <a>vocado
str_view(fruit, "a$")
#>  [4] │ banan<a>
#> [15] │ cherimoy<a>
#> [30] │ feijo<a>
#> [36] │ guav<a>
#> [56] │ papay<a>
#> [74] │ satsum<a>
```

- Full-string match: `^pattern$`.

Anchors and Word Boundaries

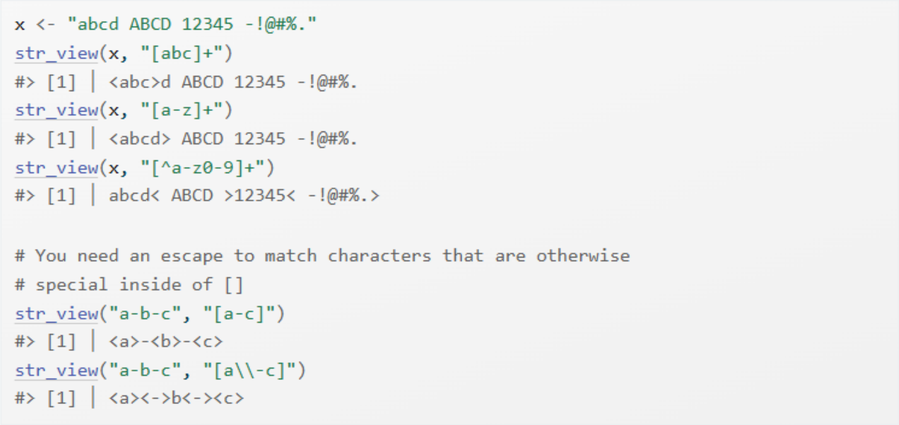
- Word boundary:



```
x <- c("summary(x)", "summarize(df)", "rowsum(x)", "sum(x)")
str_view(x, "sum")
#> [1] │ <sum>mary(x)
#> [2] │ <sum>marize(df)
#> [3] │ row<sum>(x)
#> [4] │ <sum>(x)
str_view(x, "\\bsum\\b")
#> [4] │ <sum>(x)
```

More on character sets/classes

- - (range)
- ˆ (negation).
- \ (escape).

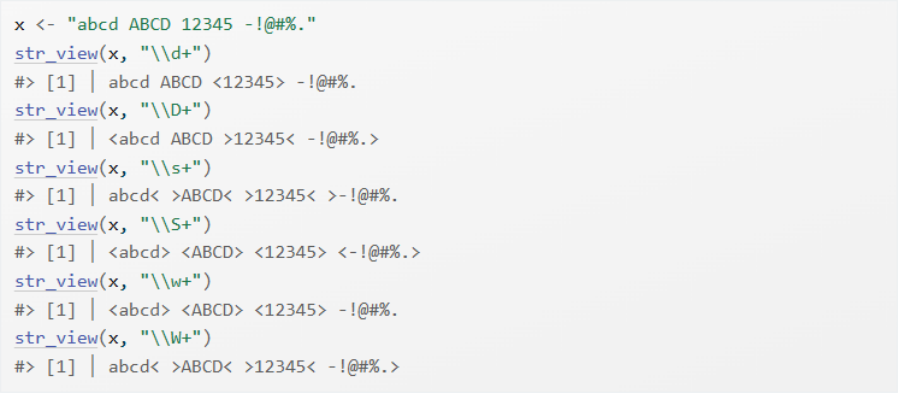


```
x <- "abcd ABCD 12345 -!@#%."
str_view(x, "[abc]+")
#> [1] │ <abc>d ABCD 12345 -!@#%.
str_view(x, "[a-z]+")
#> [1] │ <abcd> ABCD 12345 -!@#%.
str_view(x, "[^a-z0-9]+")
#> [1] │ abcd< ABCD >12345< -!@#%.>

# You need an escape to match characters that are otherwise
# special inside of []
str_view("a-b-c", "[a-c]")
#> [1] │ <a>-<b>-<c>
str_view("a-b-c", "[a\\-c]")
#> [1] │ <a><->b<-><c>
```

More on character sets/classes

- Shorthand: `\d` digit, `\s` whitespace, `\w` word char.
- Uppercase negates: `\D`, `\S`, `\W`.



```
x <- "abcd ABCD 12345 -!@#%."
str_view(x, "\\d+")
#> [1] │ abcd ABCD <12345> -!@#%.
str_view(x, "\\D+")
#> [1] │ <abcd ABCD >12345< -!@#%.>
str_view(x, "\\s+")
#> [1] │ abcd< >ABCD< >12345< >-!@#%.
str_view(x, "\\S+")
#> [1] │ <abcd> <ABCD> <12345> <-!@#%.>
str_view(x, "\\w+")
#> [1] │ <abcd> <ABCD> <12345> -!@#%.
str_view(x, "\\W+")
#> [1] │ abcd< >ABCD< >12345< -!@#%.>
```

Quantifiers

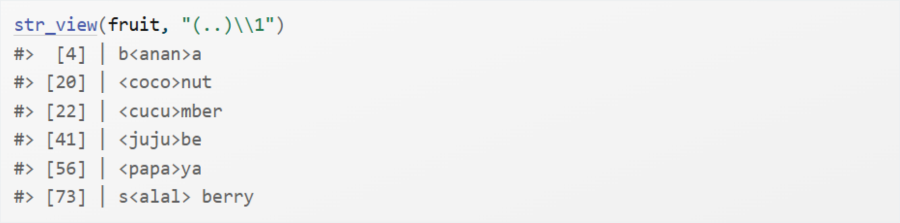
- Remember: `?`, `+`, `*`.
- Exact counts: `{n}`. E.g. `\d{3}` for exactly 3 digits.
- At least n: `{n,}`. E.g. `\d{3,}` for 3 or more digits.
- Range: `{n,m}`. E.g. `\d{3,5}` for between 3 and 5 digits.

Operator Precedence

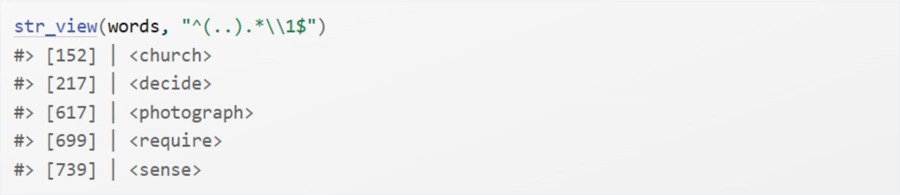
- Precedence: quantifiers > concatenation > alternation.
- Example: `ab+` = `a(b+)`.
- Example: `^a|b$` = `(^a)|(b$)`.
- Use parentheses for grouping.

Grouping and Capturing

- Parentheses capture submatches/groupings.
- Backreference refers to grouping: \1, \2, ...



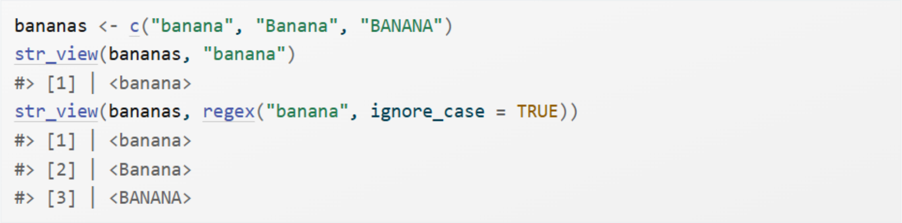
```
str_view(fruit, "(..)\\1")
#>  [4] │ b<anan>a
#> [20] │ <coco>nut
#> [22] │ <cucu>mber
#> [41] │ <juju>be
#> [56] │ <papa>ya
#> [73] │ s<alal> berry
```



```
str_view(words, "^(..).*\\1$")
#> [152] │ <church>
#> [217] │ <decide>
#> [617] │ <photograph>
#> [699] │ <require>
#> [739] │ <sense>
```

Pattern Control

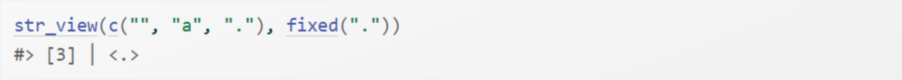
- Flags: `ignore_case=TRUE`, `dotall=TRUE`, `multiline=TRUE`, `comments=TRUE`.
- Use `regex()` to set flags **around a pattern**.
- See more in `?regex`.



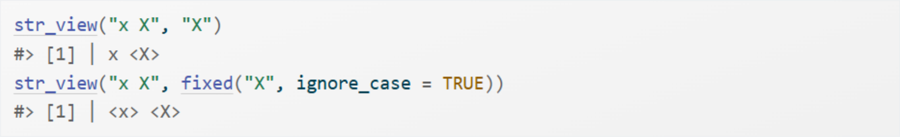
```
bananas <- c("banana", "Banana", "BANANA")
str_view(bananas, "banana")
#> [1] │ <banana>
str_view(bananas, regex("banana", ignore_case = TRUE))
#> [1] │ <banana>
#> [2] │ <Banana>
#> [3] │ <BANANA>
```

Pattern Control

- `fixed()` for literal matches.



```
str_view(c("", "a", "."), fixed("."))
#> [3] │ <.>
```



```
str_view("x X", "X")
#> [1] │ x <X>
str_view("x X", fixed("X", ignore_case = TRUE))
#> [1] │ <x> <X>
```