Lecture 3 - dates & times, strings, as well as factors

Contents

Lecture learning objectives:

By the end of this lecture and worksheet 3, students should be able to:

- Manipulate dates and times using the {lubridate} package
- Be able to modify strings in a data frame using regular expressions and the {stringr} package

Skip to main content

- Cast categorical columns in a data frame as factors when appropriate, and manipulate factor levels as needed in preparation for data visualisation and statistical analysis (using base R and {forcats} package functions)

```
library(tidyverse)
library(gapminder)
options(repr.matrix.max.rows = 10)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.2     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.0
✔ ggplot2   3.4.3     ✔ tibble    3.2.1
✔ lubridate 1.9.2     ✔ tidyr     1.3.0
✔ purrr     1.0.2
```

```
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Tibbles versus data frames

You have seen and heard me talk about data frames and tibbles in R, and sometimes I carelessly interchange the terms. Let's take moment to discuss the difference between the two!

Tibbles are special data frames, with special/extra properties. The two important ones you will care about are:

- In RStudio, tibbles only output the first 10 rows

Skip to main content

When you create a data frame using base R functions, either via `data.frame` or one of the base R `read.*` functions, you get an objects whose class is data frame:

```
example <- data.frame(a = c(1, 5, 9), b = "z", "a", "t")
example

class(example)
```

A data.frame: 3 × 4

a	b	X.a.	X.t.
<dbl>	<chr>	<chr>	<chr>
1	z	a	t
5	z	a	t
9	z	a	t

'data.frame'

Tibbles inherit from the data frame class (meaning that have many of the same properties as data frames), but they also have the extra properties I just discussed:

```
example2 <- tibble(a = c(1, 5, 9), b = "z", "a", "t")
example

class(example2)
```

Skip to main content

A data.frame: 3 × 4

a	b	X.a.	X.t.
<dbl>	<chr>	<chr>	<chr>
1	z	a	t
5	z	a	t
9	z	a	t

'tbl_df' · 'tbl' · 'data.frame'

Note: there are **some** tidyverse functions that will coerce a data frame to a tibble, because what the user is asking for is not possible with a data frame. One such example is `group_by` (which we will learn about next week):

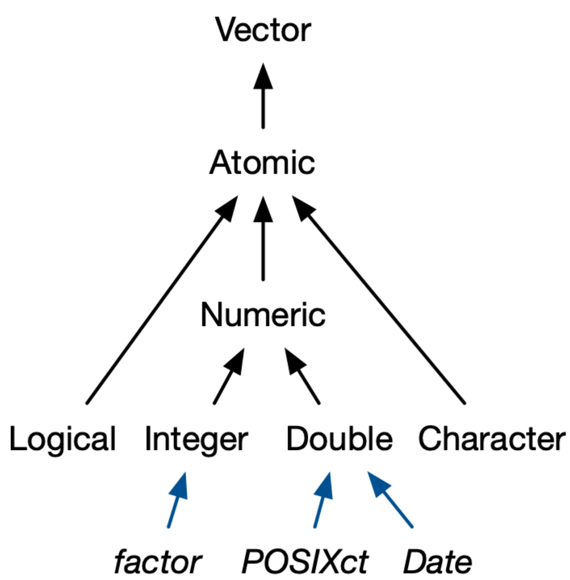
```
group_by(example, a) %>%
    class()
```

'grouped_df' · 'tbl_df' · 'tbl' · 'data.frame'

Rule of thumb: if you want a tibble, it's on you to know that and express that explicitly with `as_tibble()` (if it's a data frame to start out with).

Skip to main content

Dates and times



Source: [Advanced R](#) by Hadley Wickham

Working with dates and times

Your weapon: The lubridate package (CRAN; [GitHub](#); [main vignette](#)).

```
library(lubridate)
```

Get your hands on some dates or date-times

Use `lubridate`'s `today` to get today's date, without any time:

Skip to main content

```
today()
```

2024-08-27

```
class(today())
```

'Date'

Use `lubridate`'s `now` to get RIGHT NOW, meaning the date and the time:

```
now()
```

```
[1] "2024-08-27 15:05:15 PDT"
```

```
class(now())
```

'POSIXct' · 'POSIXt'

Get date or date-time from character

Use the `lubridate` helpers to convert character or unquoted numbers into dates or date-times:

```
ymd("2017-01-31")
```

Skip to main content

```
mdy("January 31st, 2017")
```

2017-01-31

```
dmy("31-Jan-2017")
```

2017-01-31

```
ymd(20170131)
```

2017-01-31

```
ymd_hms("2017-01-31 20:11:59")
```

```
[1] "2017-01-31 20:11:59 UTC"
```

```
mdy_hm("01/31/2017 08:01")
```

```
[1] "2017-01-31 08:01:00 UTC"
```

You can also force the creation of a date-time from a date by supplying a timezone:

```
class(ymd(20170131, tz = "UTC"))
```

Skip to main content

Build date or date-time from parts

Instead of a single string, sometimes you'll have the individual components of the date-time spread across multiple columns.

```
dates <- tibble(year = c(2015, 2016, 2017, 2018, 2019),
                month = c(9, 9, 9, 9, 9),
                day = c(3, 4, 2, 6, 3))

dates
```

A tibble: 5 × 3

year	month	day
<dbl>	<dbl>	<dbl>
2015	9	3
2016	9	4
2017	9	2
2018	9	6
2019	9	3

To create a date/time from this sort of input, use `make_date` for dates, or `make_datetime` for date-times:

```
# make a single date from year, month and day
dates %>%
    mutate(date = make_date(year, month, day))
```

Skip to main content

A tibble: 5 × 4

year	month	day	date
<dbl>	<dbl>	<dbl>	<date>
2015	9	3	2015-09-03
2016	9	4	2016-09-04
2017	9	2	2017-09-02
2018	9	6	2018-09-06
2019	9	3	2019-09-03

Getting components from a date or date-time

Sometimes you have the date or date-time and you want to extract a component, such as year or day.

```
datetime <- ymd_hms("2016-07-08 12:34:56")
datetime
```

```
[1] "2016-07-08 12:34:56 UTC"
```

```
year(datetime)
```

2016

Skip to main content

7

```
mday(datetime)
```

8

```
yday(datetime)
```

190

```
wday(datetime, label = TRUE, abbr = FALSE)
```

Friday

▶ **Levels**:

For `month` and `wday` you can set `label = TRUE` to return the abbreviated name of the month or day of the week. Set `abbr = FALSE` to return the full name.

More date and time wrangling possibilities:

- Tools for working with time spans, to calculate durations, periods and intervals
- Tools for dealing with time zones

Skip to main content

String manipulations

The recommended tools:

- **stringr package:** A core package in the tidyverse. Main functions start with str_. Auto-complete is your friend. `stringr` cheatsheet
- **tidyr package:** Useful for functions that split one character vector into many and vice versa: `separate`, `unite`, `extract`. `tidyr` cheatsheet
- **Base functions:** `nchar`, `strsplit`, `substr`, `paste`, and `paste0`.

Regex-free string manipulation with stringr and tidyr

Basic string manipulation tasks:

- Study a single character vector
  - How long are the strings?
  - Presence/absence of a literal string
- Operate on a single character vector
  - Keep/discard elements that contain a literal string
  - Split into two or more character vectors using a fixed delimiter
  - Snip out pieces of the strings based on character position
  - Collapse into a single string
- Operate on two or more character vectors
  - Glue them together element-wise to get a new character vector.

Skip to main content

`fruit`, `words`, and `sentences` are character vectors that ship with `stringr` for practicing.

NOTE - we will be working with vectors today. If you want to operate on data frames, you will need to use these functions inside of data frame/tibble functions, like `filter` and `mutate`.

Detect or filter on a target string

Determine presence/absence of a literal string with `str_detect`. Spoiler: later we see `str_detect` also detects regular expressions.

Which fruits actually use the word "fruit"?

```
# detect "fruit"
#typeof(fruit)
#fruit
str_detect(fruit, "fruit")
```

FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · TRUE · FALSE

What's the easiest way to get the actual fruits that match? Use `str_subset` to keep only the matching elements. Note we are storing this new vector `my_fruit` to use in later examples!

```
# subset "fruit"
```

Skip to main content

```
my_fruit
```

'breadfruit' · 'dragonfruit' · 'grapefruit' · 'jackfruit' · 'kiwi fruit' · 'passionfruit' · 'star fruit' · 'ugli fruit'

String splitting by delimiter

Use `stringr::str_split` to split strings on a delimiter.

Some of our fruits are compound words, like “grapefruit”, but some have two words, like “ugli fruit”. Here we split on a single space ” “, but show use of a regular expression later.

```
# split on " "
str_split(my_fruit, " ")
```

1. 'breadfruit'
2. 'dragonfruit'
3. 'grapefruit'
4. 'jackfruit'
5. 'kiwi' · 'fruit'
6. 'passionfruit'
7. 'star' · 'fruit'
8. 'ugli' · 'fruit'

It’s bummer that we get a list back. But it must be so! In full generality, split strings must return list, because who knows how many pieces there will be?

If you are willing to commit to the number of pieces, you can use `str_split_fixed` and get a character matrix. You're

Skip to main content

```
str_split_fixed(my_fruit, pattern = " ", n = 2)
```

A matrix: 8 × 2 of type chr

breadfruit	
dragonfruit	
grapefruit	
jackfruit	
kiwi	fruit
passionfruit	
star	fruit
ugli	fruit

If the to-be-split variable lives in a data frame, `tidyr::separate` will split it into 2 or more variables:

```
# separate on " "
my_fruit[5] <- "yellow kiwi fruit"
my_fruit
tibble(unsplit = my_fruit) %>%
    separate(unsplit, into = c("pre", "post"), sep = " ")
```

'breadfruit' · 'dragonfruit' · 'grapefruit' · 'jackfruit' · 'yellow kiwi fruit' · 'passionfruit' · 'star fruit' · 'ugli fruit'

```
Warning message:
“Expected 2 pieces. Additional pieces discarded in 1 rows [5].”
```

Skip to main content

```
Warning message:
“Expected 2 pieces. Missing pieces filled with `NA` in 5 rows [1, 2, 3, 4, 6].”
```

A tibble: 8 × 2

pre	post
<chr>	<chr>
breadfruit	NA
dragonfruit	NA
grapefruit	NA
jackfruit	NA
yellow	kiwi
passionfruit	NA
star	fruit
ugli	fruit

Substring extraction (and replacement) by position

Count characters in your strings with `str_length`. *Note this is different from the length of the character vector itself.*

```
# get length of each string
str_length(my_fruit)
my_fruit
```

Skip to main content

'breadfruit' · 'dragonfruit' · 'grapefruit' · 'jackfruit' · 'yellow kiwi fruit' · 'passionfruit' · 'star fruit' · 'ugli fruit'

You can snip out substrings based on character position with `str_sub`.

```
# remove first three strings
str_sub(my_fruit, 1, 3)
```

'bre' · 'dra' · 'gra' · 'jac' · 'yel' · 'pas' · 'sta' · 'ugl'

Finally, `str_sub` also works for assignment, i.e. on the left hand side of `<-`

```
# replace three characters with AAA
str_sub(my_fruit, 1, 3)  <- "AAA"
my_fruit
```

'AAAadfruit' · 'AAAgonfruit' · 'AAApefruit' · 'AAAkfruit' · 'AAAlow kiwi fruit' · 'AAAsionfruit' · 'AAAr fruit' · 'AAAi fruit'

Collapse a vector

You can collapse a character vector of length n > 1 to a single string with `str_c`, which also has other uses (see the next section).

```
# collapse a character vector into one
head(fruit) %>%
    str_c(collapse = "-")
```

'apple-apricot-avocado-banana-bell pepper-bilberry'

Skip to main content

Create a character vector by catenating multiple vectors

If you have two or more character vectors of the same length, you can glue them together element-wise, to get a new vector of that length. Here are some … awful smoothie flavors?

```
# concatenate character vectors
fruit[1:4]
fruit[5:8]
str_c(fruit[1:4], fruit[5:8], sep = "")
```

'apple' · 'apricot' · 'avocado' · 'banana'

'bell pepper' · 'bilberry' · 'blackberry' · 'blackcurrant'

'applebell pepper' · 'apricotbilberry' · 'avocadoblackberry' · 'bananablackcurrant'

If the to-be-combined vectors are variables in a data frame, you can use `tidyr::unite` to make a single new variable from them.

```
# concatenate character vectors when they are in a data frame
tibble(fruit1 = fruit[1:4],
       fruit2 = fruit[5:8]) %>%
    unite("flavour_combo", fruit1, fruit2, sep = " & ")
```

Skip to main content

A tibble: 4 × 1

flavour_combo
<chr>
apple & bell pepper
apricot & bilberry
avocado & blackberry
banana & blackcurrant

Substring replacement

You can replace a pattern with `str_replace`. Here we use an explicit string-to-replace, but later we revisit with a regular expression.

```
# replace fruit with vegetable
my_fruit <- str_subset(fruit, "fruit")
my_fruit
str_replace(my_fruit, "fruit", "vegetable")
```

'breadfruit' · 'dragonfruit' · 'grapefruit' · 'jackfruit' · 'kiwi fruit' · 'passionfruit' · 'star fruit' · 'ugli fruit'

'breadvegetable' · 'dragonvegetable' · 'grapevegetable' · 'jackvegetable' · 'kiwi vegetable' · 'passionvegetable' · 'star vegetable' · 'ugli vegetable'

- A special case that comes up a lot is replacing `NA`, for which there is `str_replace_na`.
- If the NA-afflicted variable lives in a data frame, you can use `tidyr::replace_na`.

Skip to main content

Other `str_*` functions?

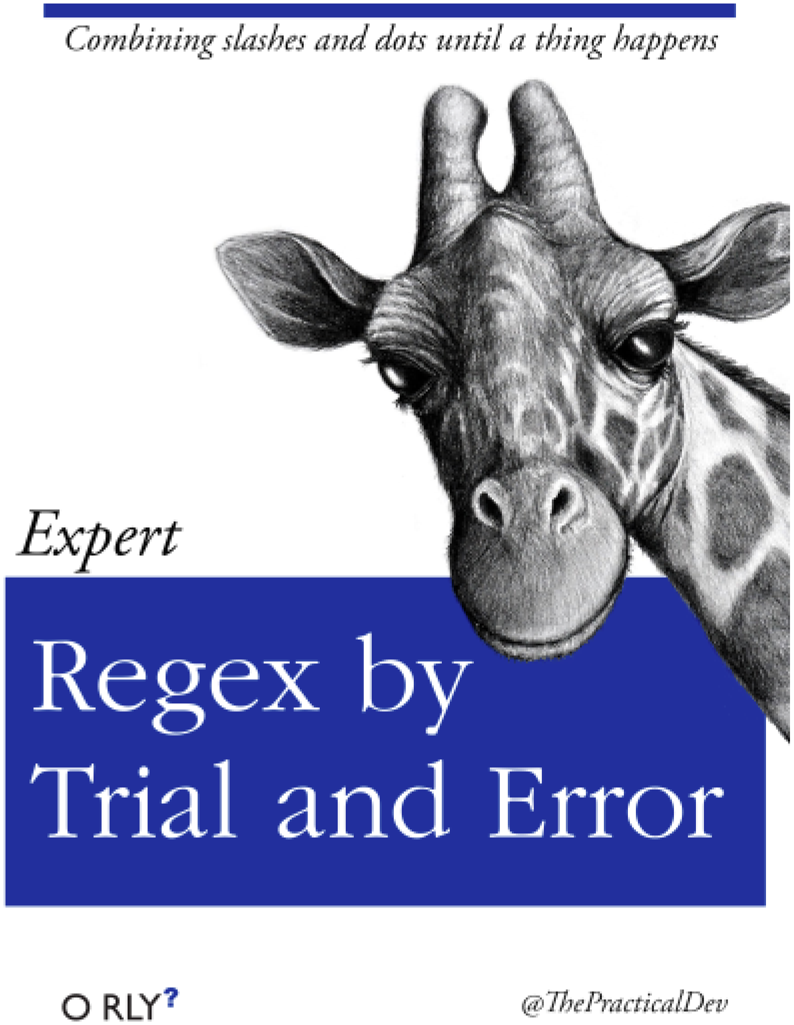
There are many many other useful `str_*` functions from the `stringr` package. Too many to go through them all here. If these shown in lecture aren't what you need, then you should try `?str` + tab to see the possibilities:

```
?str_
```

Regular expressions with stringr

or…

Skip to main content



Examples with gapminder

```
library(gapminder)
head(gapminder)
```

Skip to main content

A tibble: 6 × 6

country	continent	year	lifeExp	pop	gdpPercap
<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
Afghanistan	Asia	1952	28.801	8425333	779.4453
Afghanistan	Asia	1957	30.332	9240934	820.8530
Afghanistan	Asia	1962	31.997	10267083	853.1007
Afghanistan	Asia	1967	34.020	11537966	836.1971
Afghanistan	Asia	1972	36.088	13079460	739.9811
Afghanistan	Asia	1977	38.438	14880372	786.1134

Filtering rows with `str_detect`

Let's filter for rows where the country name starts with "AL":

```
library(tidyverse)
library(gapminder)
```

```
# detect countries that start with "AL"
gapminder %>%
    filter(str_detect(country, "^Al")) %>%
    pull(country) %>%
    unique() %>%
    length()
```

Skip to main content

And now rows where the country ends in `tan`:

```
# detect countries that end with "tan"
gapminder %>%
    filter(str_detect(country, "tan$"))
```

A tibble: 24 × 6

country	continent	year	lifeExp	pop	gdpPercap
<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
Afghanistan	Asia	1952	28.801	8425333	779.4453
Afghanistan	Asia	1957	30.332	9240934	820.8530
Afghanistan	Asia	1962	31.997	10267083	853.1007
Afghanistan	Asia	1967	34.020	11537966	836.1971
Afghanistan	Asia	1972	36.088	13079460	739.9811
⋮	⋮	⋮	⋮	⋮	⋮
Pakistan	Asia	1987	58.245	105186881	1704.687
Pakistan	Asia	1992	60.838	120065004	1971.829
Pakistan	Asia	1997	61.818	135564834	2049.351
Pakistan	Asia	2002	63.610	153403524	2092.712
Pakistan	Asia	2007	65.483	169270617	2605.948

Or countries containing “, Dem. Rep.” :

Skip to main content

```
# detect countries that contain ", Dem. Rep."
gapminder %>%
    filter(str_detect(country, "\\, Dem. Rep."))
```

A tibble: 24 × 6

country	continent	year	lifeExp	pop	gdpPercap
<fct>	<fct>	<int>	<dbl>	<int>	<dbl>
Congo, Dem. Rep.	Africa	1952	39.143	14100005	780.5423
Congo, Dem. Rep.	Africa	1957	40.652	15577932	905.8602
Congo, Dem. Rep.	Africa	1962	42.122	17486434	896.3146
Congo, Dem. Rep.	Africa	1967	44.056	19941073	861.5932
Congo, Dem. Rep.	Africa	1972	45.989	23007669	904.8961
⋮	⋮	⋮	⋮	⋮	⋮
Korea, Dem. Rep.	Asia	1987	70.647	19067554	4106.492
Korea, Dem. Rep.	Asia	1992	69.978	20711375	3726.064
Korea, Dem. Rep.	Asia	1997	67.727	21585105	1690.757
Korea, Dem. Rep.	Asia	2002	66.662	22215365	1646.758
Korea, Dem. Rep.	Asia	2007	67.297	23301725	1593.065

Replace “, Dem. Rep.” with ” Democratic Republic”:

```
# replace ", Dem. Rep." with " Democratic Republic"
gapminder %>%
```

Skip to main content

```
                                " Democratic Republic")) %>%
    filter(country == "Korea Democratic Republic")
```

A tibble: 12 × 6

country	continent	year	lifeExp	pop	gdpPercap
<chr>	<fct>	<int>	<dbl>	<int>	<dbl>
Korea Democratic Republic	Asia	1952	50.056	8865488	1088.278
Korea Democratic Republic	Asia	1957	54.081	9411381	1571.135
Korea Democratic Republic	Asia	1962	56.656	10917494	1621.694
Korea Democratic Republic	Asia	1967	59.942	12617009	2143.541
Korea Democratic Republic	Asia	1972	63.983	14781241	3701.622
⋮	⋮	⋮	⋮	⋮	⋮
Korea Democratic Republic	Asia	1987	70.647	19067554	4106.492
Korea Democratic Republic	Asia	1992	69.978	20711375	3726.064
Korea Democratic Republic	Asia	1997	67.727	21585105	1690.757
Korea Democratic Republic	Asia	2002	66.662	22215365	1646.758
Korea Democratic Republic	Asia	2007	67.297	23301725	1593.065

Extract matches (optional)

To extract the actual text of a match, use `str_extract`.

Skip to main content

```r
typeof(sentences)
head(sentences)
```

'character'

'The birch canoe slid on the smooth planks.' · 'Glue the sheet to the dark blue background.' · 'It\'s easy to tell the depth of a well.' · 'These days a chicken leg is a rare dish.' · 'Rice is often served in round bowls.' · 'The juice of lemons makes fine punch.'

Say we want to extract all of the colours used in the sentences. We can do this by creating a pattern which would match them, and passing that and our vector of sentences to `str_extract`:

```r
colours <- "red|orange|yellow|green|blue|purple"
```

```r
# extract colours used in sentences
str_extract(sentences, colours)
```

NA · 'blue' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · 'blue' · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'blue' · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'yellow' · NA · NA · NA · 'red' · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' ·
NA · 'green' · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · 'blue' · 'red' · NA · 'red' · 'red' · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA ·

Skip to main content

NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'yellow' ·
NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · 'orange' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA ·
NA · NA · NA · NA · NA · NA · NA · 'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA ·
'red' · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA · NA

`str_extract` only returns the first match for each element. To return all matches from an element we need to use `str_extract_all`:

```
# extract all colours used in sentences
str_extract_all(sentences, colours)
```

1.
2. 'blue'
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.

Skip to main content

16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26. 'blue'
27.
28. 'red'
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.

Skip to main content

44. 'red'
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.

Skip to main content

72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82. 'red'
83.
84.
85.
86.
87.
88.
89.
90.
91.
92. 'blue'
93.
94.
95.
96.
97.

Skip to main content

100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112. 'yellow'
113.
114.
115.
116. 'red'
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.

Skip to main content

28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46. 'red'
47.
48. 'green'
49. 'red'
50.
51.
52.
53.

Skip to main content

56.
57.
58.
59.
60. 'red'
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74. 'blue'
75. 'red'
76.
77. 'red'
78. 'red'
79.
80.
81.

Skip to main content

184. 'red'
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204. 'blue'
205.
206.
207.
208.
209.
210

Skip to main content

212.
213.
214.
215. 'red'
216.
217. 'blue' · 'red'
218.
219.
220. 'red'
221.
222.
223.
224. 'green'
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.

Skip to main content

240.
241.
242.
243.
244.
245.
246.
247. 'red'
248.
249.
250.
251.
252.
253.
254.
255. 'red'
256. 'red'
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.

268.
269.
270.
271.
272.
273.
274. 'red'
275.
276.
277. 'red'
278.
279. 'red'
280.
281.
282.
283.
284.
285.
286.
287.
288. 'green'
289.
290.
291.
292.
293. 'red'

Skip to main content

296.
297.
298.
299.
300.
301.
302. 'green'
303.
304.
305.
306.
307.
308.
309.
310.
311. 'red'
312.
313.
314.
315.
316.
317.
318.
319. 'purple'
320.
321.

Skip to main content

324.
325.
326.
327.
328.
329.
330. 'green'
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345. 'red'
346.
347.
348.
349.
350.

Skip to main content

352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368. 'red'
369.
370.
371.
372. 'red'
373.
374.
375.
376.
377.
378.

Skip to main content

380.
381.
382.
383.
384.
385.
386.
387. 'red'
388. 'red'
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406

Skip to main content

Skip to main content

436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452. 'blue'
453.
454.
455.
456.
457.
458.
459.
460.
461.

Skip to main content

464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485. 'red'
486.
487.
488.
489.

Skip to main content

492.
493.
494. 'red'
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512. 'red'
513.
514.
515.
516.
517.
518.

Skip to main content

520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539. 'red'
540.
541.
542.
543.
544.
545.

Skip to main content

548.
549.
550.
551. 'red'
552.
553.
554.
555.
556.
557.
558.
559.
560.
561. 'green'
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574

Skip to main content

576. 'green' · 'red'
577.
578.
579.
580.
581.
582. 'red'
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.

Skip to main content

604.
605.
606.
607.
608.
609.
610.
611. 'red'
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630

Skip to main content

32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42. 'red'
43.
44. 'orange' · 'red'
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.

Skip to main content

60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74. 'red'
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.

Skip to main content

688. 'red'
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705. 'red'
706.
707.
708.
709.
710.
711.
712.
713.
714.

Skip to main content

’16.

’17.

’18.

’19.

’20.

Note: `str_extract` returns a character vector, whereas `str_extract_all` returns a litst. This is because when asking for multiple matches back, you do not know how many you will get, and thus we cannot expect a rectangular shape.

Capture groups (optional)

You can also use parentheses to extract parts of a complex match.

For example, imagine we want to extract nouns from the sentences. As a heuristic, we’ll look for any word that comes after “a” or “the”. Defining a “word” in a regular expression is a little tricky, so here I use a simple approximation: a sequence of at least one character that isn’t a space.

```
# extract nouns from sentences
noun <- "(a|the) ([^ ]+)"

str_match(sentences, noun) %>%
    head()
```

Skip to main content

A matrix: 6 × 3 of type chr

the smooth	the	smooth
the sheet	the	sheet
the depth	the	depth
a chicken	a	chicken
NA	NA	NA
NA	NA	NA

Like `str_extract`, if you want all matches for each string, you'll need `str_match_all`

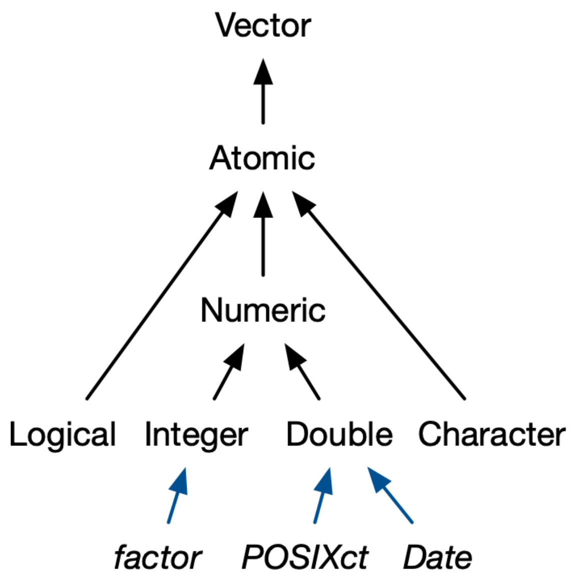
Skip to main content

Summary of string manipulation functions covered so far:

function	description
`str_detect`	Detects elements in a vector that match a pattern, returns a vector of logicals
`srt_subset`	Detects and returns elements in a vector that match a pattern
`str_split`	Split strings in a vector on a delimiter. Returns a list (used `str_split_fixed` to get a matrix)
`separate`	Split character vectors from a data frame on a delimiter which get returned as additional columns in the data frame
`str_length`	Counts the number of characters for each element of a character vector, and returns a numeric vector of the counts
`str_sub`	Remove substrings based on character position
`str_c`	Collapse and/or concatenate elements from a character vector(s)
`unite`	Concatenate elements from character vectors from a data frame to create a single column
`str_replace`	Replace a pattern in a vector of character vectors with a given string
`str_extract`	Extract the actual text of a match from a character vector
`str_match`	Use capture groups to extract parts of a complex match from a character vector, returns the match and the capture groups as columns of a matrix

Skip to main content

Factors



Source: [Advanced R](#) by Hadley Wickham

Be the boss of your factors

- I love and hate factors
- I love them for data visualization and statistics because I do not need to make dummy variables
- I hate them because if you are not careful, they fool you because they look like character vectors. And when you treat them like character vectors you get cryptic error messages, like we saw when we tried to do a conditional mutate on the gapminder data set

Tidyverse philosophy for factors

Skip to main content

- Factors are not that useful until you are at the end of your data wrangling, before that you want character vectors so you can do string manipulations
- Tidyverse functions, like `tibble`, and `read_csv` give you columns with strings as character vectors, Base R functions like `data.frame` and `read.csv`

Factor inspection

Get to know your factor before you start touching it! It's polite. Let's use `gapminder$continent` as our example.

```
str(gapminder$continent)
```

```
Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
```

```
levels(gapminder$continent)
```

'Africa' · 'Americas' · 'Asia' · 'Europe' · 'Oceania'

```
nlevels(gapminder$continent)
```

5

```
class(gapminder$continent)
```

'factor'

Skip to main content

Dropping unused levels

Just because you drop all the rows corresponding to a specific factor level, the levels of the factor itself do not change. Sometimes all these unused levels can come back to haunt you later, e.g., in figure legends.

Watch what happens to the levels of `country` when we filter Gapminder to a handful of countries:

```
nlevels(gapminder$country)
```

142

```
h_countries <- gapminder %>%
    filter(country %in% c("Egypt", "Haiti", "Romania", "Thailand", "Venezuela"))

nlevels(h_countries$country)
```

142

huh? Even though `h_gap` only has data for a handful of countries, we are still schlepping around all 142 levels from the original `gapminder` tibble.

How to get rid of them? We'll use the `forcats::fct_drop` function to do this:

```
h_countries$country %>% nlevels()
```

142

```
h_countries$country %>%
```

```
    nlevels
```

5

Change order of the levels, principled

By default, factor levels are ordered alphabetically. Which might as well be random, when you think about it! It is preferable to order the levels according to some principle:

- Frequency. Make the most common level the first and so on.
- Another variable. Order factor levels according to a summary statistic for another variable. Example: order Gapminder countries by life expectancy.

First, let's order continent by frequency, forwards and backwards. This is often a great idea for tables and figures, esp. frequency barplots.

```
## default order is alphabetical
gapminder$continent %>%
    levels()
```

'Africa' · 'Americas' · 'Asia' · 'Europe' · 'Oceania'

Let's use `forcats::fct_infreq` to order by frequency:

```
gapminder$continent %>%
    fct_infreq() %>%
    levels()

gap2 <- gapminder %>%
```

Skip to main content

```
gap2$continent %>%
    levels()
```

'Africa' · 'Asia' · 'Europe' · 'Americas' · 'Oceania'

'Africa' · 'Asia' · 'Europe' · 'Americas' · 'Oceania'

Or reverse frequency:

```
gapminder$continent %>%
  fct_infreq() %>%
  fct_rev() %>%
  levels()
```

'Oceania' · 'Americas' · 'Europe' · 'Asia' · 'Africa'

Order one variable by another

You can use `forcats::fct_reorder` to order one variable by another.

The factor is the grouping variable and the default summarizing function is `median` but you can specify something else.

```
## order countries by median life expectancy
fct_reorder(gapminder$country, gapminder$lifeExp) %>%
    levels() %>%
    head()
```

'Sierra Leone' · 'Guinea-Bissau' · 'Afghanistan' · 'Angola' · 'Somalia' · 'Guinea'

Using `min` instead to reorder the factors:

```r
## order accoring to minimum life exp instead of median
fct_reorder(gapminder$country, gapminder$lifeExp, min) %>%
    levels() %>%
    head()
```

'Rwanda' · 'Afghanistan' · 'Gambia' · 'Angola' · 'Sierra Leone' · 'Cambodia'

Change order of the levels, “because I said so”

Sometimes you just want to hoist one or more levels to the front. Why? Because I said so (sometimes really useful when creating visualizations).

Reminding ourselves of the level order for `gapminder$continent`:

```r
gapminder$continent %>% levels()
```

'Africa' · 'Americas' · 'Asia' · 'Europe' · 'Oceania'

Reorder and put Asia and Africa first:

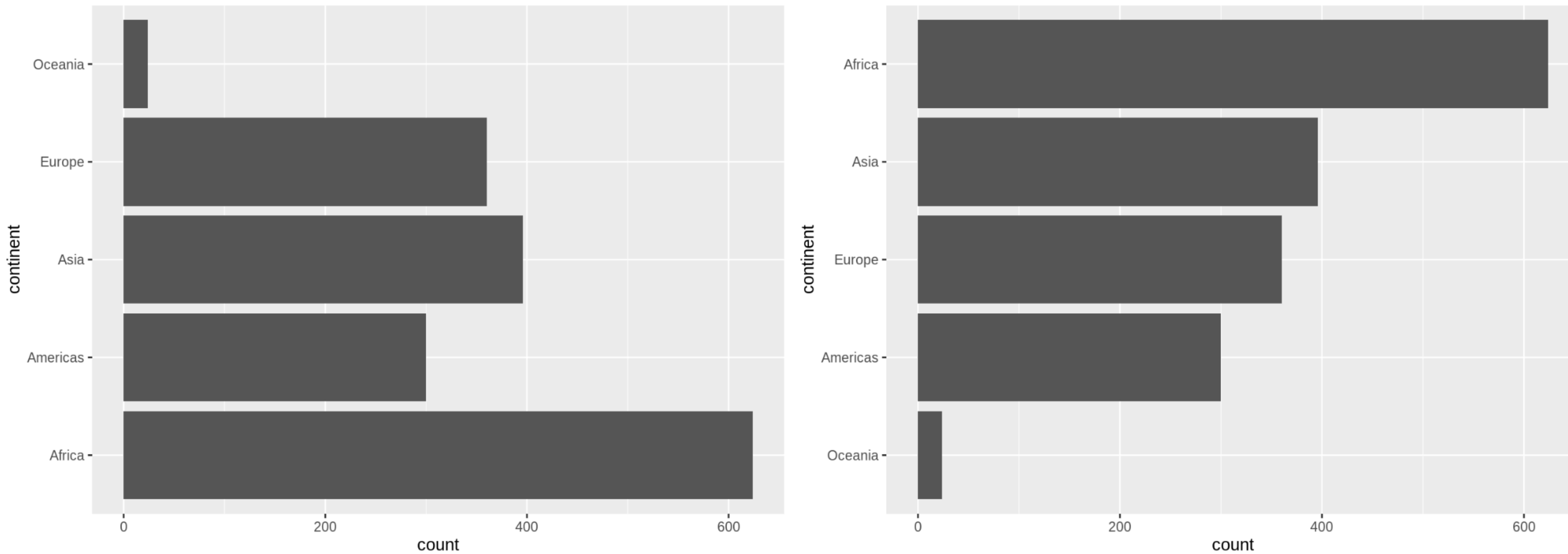
```r
gapminder$continent %>%
    fct_relevel("Asia", "Africa")  %>%
    levels()
```

'Asia' · 'Africa' · 'Americas' · 'Europe' · 'Oceania'

Skip to main content

Why do we need to know how to do this?

- Factor levels impact statistical analysis & data visualization!
- For example, these two barcharts of frequency by continent differ only in the order of the continents. Which do you prefer? Discuss with your neighbour.



What did we learn today?

- The differences between data frames and tibbles
- The beautiful {lubridate} package for working with dates and times
- Tools for manipulating and working with character data in the {stringr} and {tidyr} packages
- How to take control of our factors using {forcats} and how to investigate factors using base R functions

Skip to main content

Attributions

- Stat 545 created by Jenny Bryan
- R for Data Science by Garrett Grolemund & Hadley Wickham

Previous
Lecture 2 - Key datatypes & operators in R

Next
Lecture 4 - Combining data frames (binds and joins) & base R control flow