Strings, Factors, and Dates

Week 8 (11/19/25)

Artwork by Allison Horst

Stepfanie M. Aguillon

Outline of today's class

- Introduction to strings with `stringr`
- Introduction to factors with `forcats`
- Brief introduction to dates with `lubridate`

`tidyverse` package for working with dates

`tidyverse` package for working with factors

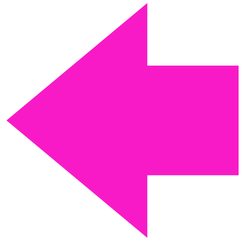
`tidyverse` package for working with strings

Introduction to `stringr`

Artwork by Allison Horst

Types of data in R: what are they?

Numeric	`1.25, 7, 10.5, 3.14159`
Integer	`1, 2, 3, 4, 5`
Character	`"Jan", "Feb", "Mar"`
Logical	`TRUE, FALSE`



Important `stringr` functionality

- Detect matches within strings
- Manipulate whole strings or parts of strings
- Subset strings or change their lengths
- Join or split different strings
- Order strings

String manipulation with stringr : : **CHEATSHEET**

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

str_detect(string, **pattern**, negate = FALSE**)** Detect the presence of a pattern match in a string. Also **str_like()**. str_detect(fruit, "a")

str_starts(string, **pattern**, negate = FALSE**)** Detect the presence of a pattern match at the beginning of a string. Also **str_ends()**. str_starts(fruit, "a")

str_which(string, **pattern**, negate = FALSE**)** Find the indexes of strings that contain a pattern match. str_which(fruit, "a")

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all()**. str_locate(fruit, "a")

str_count(string, **pattern**) Count the number of matches in a string. str_count(fruit, "a")

Subset Strings

str_sub(string, start = 1L, end = -1L**)** Extract substrings from a character vector. str_sub(fruit, 1, 3); str_sub(fruit, -2)

str_subset(string, **pattern**, negate = FALSE**)** Return only the strings that contain a pattern match. str_subset(fruit, "p")

str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all()** to return every pattern match. str_extract(fruit, "[aeiou]")

str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all()**. str_match(sentences, "(a|the) ([^ +])")

Manage Lengths

str_length(string**)** The width of strings (i.e. number of code points, which generally equals the number of characters). str_length(fruit)

str_pad(string, width, side = c("left", "right", "both"), pad = " "**)** Pad strings to constant width. str_pad(fruit, 17)

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "..."**)** Truncate the width of strings, replacing content with ellipsis. str_trunc(sentences, 6)

str_trim(string, side = c("both", "left", "right"**))** Trim whitespace from the start and/or end of a string. str_trim(str_pad(fruit, 17))

str_squish(string**)** Trim whitespace from each end and collapse multiple spaces into single spaces. str_squish(str_pad(fruit, 17, "both"))

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. str_sub(fruit, 1, 3) <- "str"

str_replace(string, **pattern**, replacement**)** Replace the first matched pattern in each string. Also **str_remove()**. str_replace(fruit, "p", "-")

str_replace_all(string, **pattern**, replacement**)** Replace all matched patterns in each string. Also **str_remove_all()**. str_replace_all(fruit, "p", "-")

str_to_lower(string, locale = "en"**)**[1] Convert strings to lower case. str_to_lower(sentences)

str_to_upper(string, locale = "en"**)**[1] Convert strings to upper case. str_to_upper(sentences)

str_to_title(string, locale = "en"**)**[1] Convert strings to title case. Also **str_to_sentence()**. str_to_title(sentences)

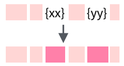
Join and Split

str_c(..., sep = "", collapse = NULL**)** Join multiple strings into a single string. str_c(letters, LETTERS)

str_flatten(string, collapse = ""**)** Combines into a single string, separated by collapse. str_flatten(fruit, ", ")

str_dup(string, times**)** Repeat strings times times. Also **str_unique()** to remove duplicates. str_dup(fruit, times = 2)

str_split_fixed(string, **pattern**, n**)** Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split()** to return a list of substrings and **str_split_i()** to return the ith substring. str_split_fixed(sentences, " ", n=3)



str_glue(..., .sep = "", .envir = parent.frame()**)** Create a string from strings and {expressions} to evaluate. str_glue("Pi is {pi}")

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA"**)** Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...**)**[1] Return the vector of indexes that sorts a character vector. fruit[str_order(fruit)]

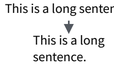
str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...**)**[1] Sort a character vector. str_sort(fruit)

Helpers

str_conv(string, encoding**)** Override the encoding of a string. str_conv(fruit,"ISO-8859-1")

str_view(string, **pattern**, match = NA**)** View HTML rendering of all regex matches. str_view(sentences, "[aeiou]")

str_equal(x, y, locale = "en", ignore_case = FALSE, ...**)**[1] Determine if two strings are equivalent. str_equal(c("a", "b"), c("a", "c"))



str_wrap(string, width = 80, indent = 0, exdent = 0**)** Wrap strings into nicely formatted paragraphs. str_wrap(sentences, 20)

[1] See **bit.ly/ISO639-1** for a complete list of locales.

Explore `stringr` using the `fruit` dataset

```
fruit
```

```
 [1] "apple"              "apricot"          "avocado"          "banana"           "bell pepper"
 [6] "bilberry"           "blackberry"       "blackcurrant"     "blood orange"     "blueberry"
[11] "boysenberry"        "breadfruit"       "canary melon"     "cantaloupe"       "cherimoya"
[16] "cherry"             "chili pepper"     "clementine"       "cloudberry"       "coconut"
[21] "cranberry"          "cucumber"         "currant"          "damson"           "date"
[26] "dragonfruit"        "durian"           "eggplant"         "elderberry"       "feijoa"
[31] "fig"                "goji berry"       "gooseberry"       "grape"            "grapefruit"
[36] "guava"              "honeydew"         "huckleberry"      "jackfruit"        "jambul"
[41] "jujube"             "kiwi fruit"       "kumquat"          "lemon"            "lime"
[46] "loquat"             "lychee"           "mandarine"        "mango"            "mulberry"
[51] "nectarine"          "nut"              "olive"            "orange"           "pamelo"
[56] "papaya"             "passionfruit"     "peach"            "pear"             "persimmon"
[61] "physalis"           "pineapple"        "plum"             "pomegranate"      "pomelo"
[66] "purple mangosteen" "quince"           "raisin"           "rambutan"         "raspberry"
[71] "redcurrant"         "rock melon"       "salal berry"      "satsuma"          "star fruit"
[76] "strawberry"         "tamarillo"        "tangerine"        "ugli fruit"       "watermelon"
```

Detecting matches in strings

`str_detect()` **detect** a pattern in a string

`str_which()` identify **which** strings have a particular pattern

`str_count()` **count** the number of pattern matches

Detecting matches in strings

```
str_detect(<data>, <"pattern">)
```

```
str_detect(fruit, "a")
```

detect if a fruit string as an "a"

```
str_which(fruit, "a")
```

identify **which** fruit string as an "a"

```
str_count(fruit, "a")
```

count the number of "a"s in a fruit string

Manipulating strings

`str_replace()` **replace** the first matched pattern in a string

`str_to_lower()`
`str_to_upper()`
`str_to_title()`

convert strings to a different **case** (uppercase, lowercase, title case)

Mutating strings

```
str_replace(<data>, <"pattern">, <"replacement">)
```

```
str_replace(fruit, "a", "-")
```

replace first "a" in fruit strings with "-"

```
str_to_*(<data>)
```

```
str_to_upper(fruit)
```

converts all fruit strings to **uppercase**

Subsetting strings

`str_subset()` return the **subset** of strings with a pattern match

`str_extract()` **extract** the first pattern match found in a string

Subsetting strings

```
str_subset(<data>, <"pattern">)
```

```
str_subset(fruit, "a")
```

return the **subset** of fruit strings with an "a"

```
str_extract(fruit, "a")
```

extracts "a" when present in a fruit string (or returns NA)

Joining or splitting strings

`str_c()` **concatenate** (join) multiple strings together

`str_split()` **split** a single string into multiple strings

Joining or splitting strings

```
str_c(<data1>, <data2>, sep = <"delimiter">)
```

```
str_c(fruit, fruit, sep = ",")
```

concatenate fruit strings together separated by a ","

```
str_split(<data>, <"pattern">)
```

```
str_split(fruit, "a")
```

splits all fruit strings at "a"s

Changing the lengths of strings

`str_length()` determine the **length** of a string (number of characters)

`str_trim()` **trims** spaces from the start/end of a string

Changing the lengths of strings

```
str_length(<data>)
```

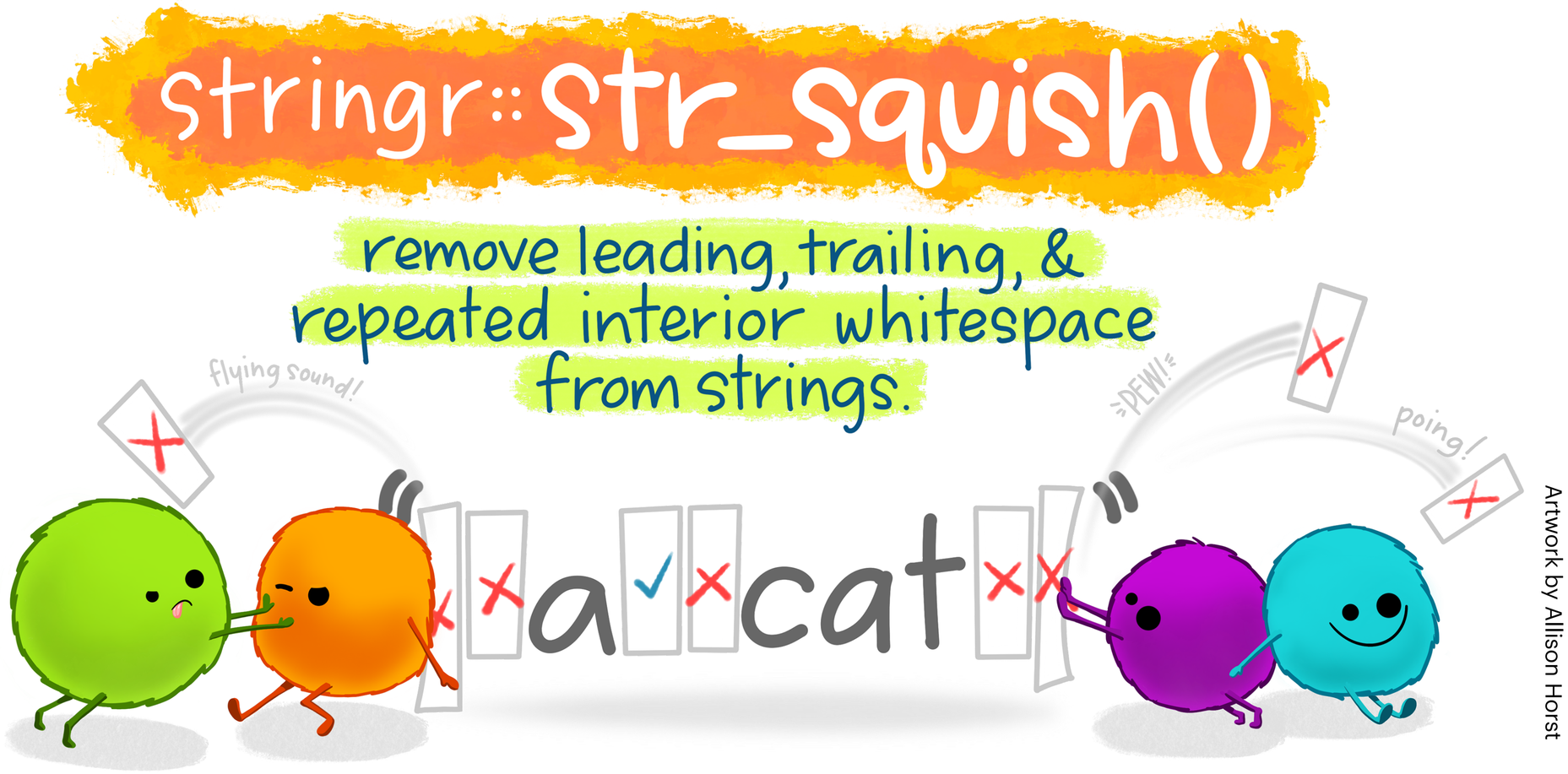
```
str_length(fruit)
```

determine the **length** of each fruit string

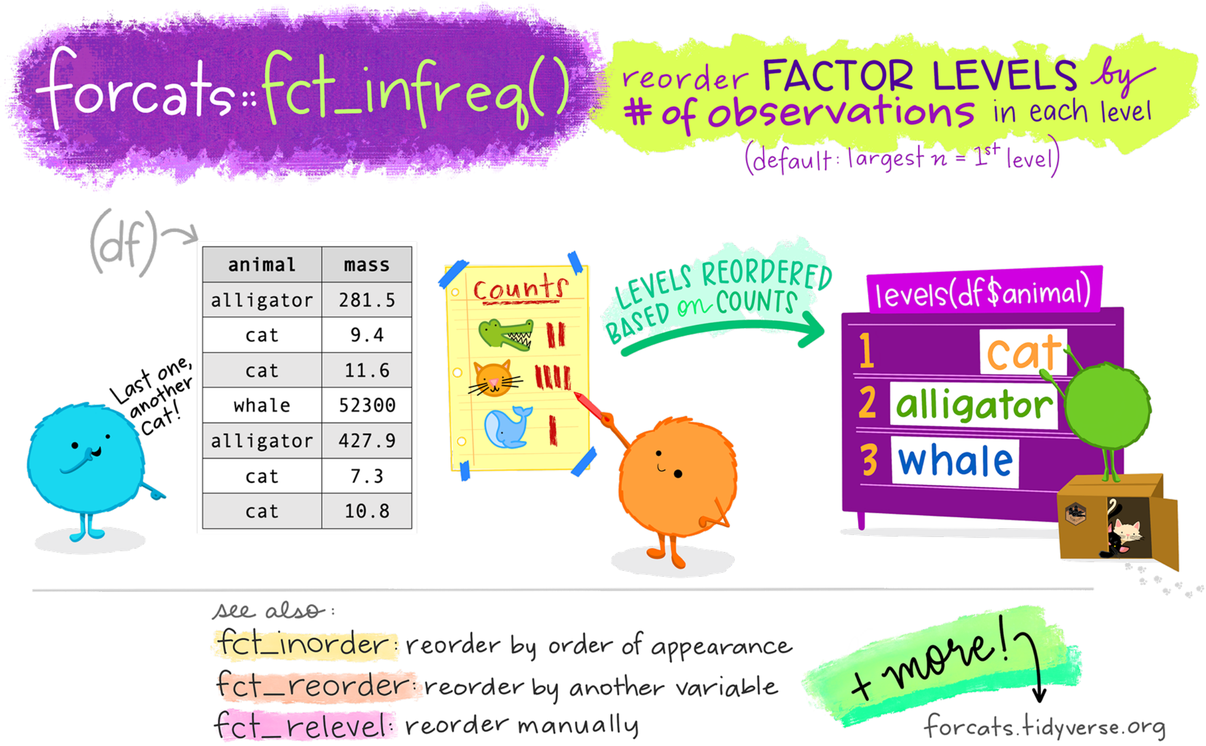
```
str_trim(fruit)
```

nothing! (there's no spaces at the start/end of the strings)

Changing the lengths of strings



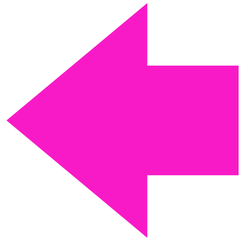
Introduction to `forcats`



Artwork by Allison Horst

Types of data in R: what are they?

Numeric	`1.25, 7, 10.5, 3.14159`
Integer	`1, 2, 3, 4, 5`
Character	`"Jan", "Feb", "Mar"`
Logical	`TRUE, FALSE`



Characters vs. Factors

Character

```
"Jan", "Feb", "Mar"
```

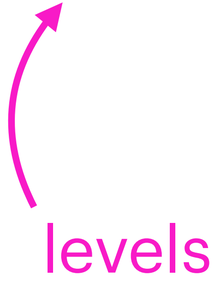
Categorical variables

Factor

```
"Jan", "Feb", "Mar"
```

Categorical variables with a known set of possible values

(“Jan” ... “Dec”)



levels

Important `forcats` functionality

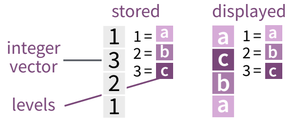
- Inspect or change the details of a factor variable
- Reorder the levels of a factor in a variety of ways
- Change, add, or drop levels of a factor

Factors with forcats : : **CHEATSHEET**

The **forcats** package provides tools for working with factors, which are R's data structure for categorical data.

Factors

R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the levels associated with them.



Create a factor with factor()

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA**)** Convert a vector to a factor. Also **as_factor()**.

```
f <- factor(c("a", "c", "b", "a"),
            levels = c("a", "b", "c"))
```

Return its levels with levels()

levels(x) Return/set the levels of a factor. levels(f); levels(f) <- c("x","y","z")

Use unclass() to see its structure

Inspect Factors

fct_count(f, sort = FALSE, prop = FALSE**)** Count the number of values with each level. fct_count(f)

fct_match(f, lvls**)** Check for lvls in f. fct_match(f, "a")

fct_unique(f) Return the unique values, removing duplicates. fct_unique(f)

Combine Factors

fct_c(…**)** Combine factors with different levels. Also **fct_cross()**.

```
f1 <- factor(c("a", "c"))
f2 <- factor(c("b", "a"))
fct_c(f1, f2)
```

fct_unify(fs, levels = lvls_union(fs)**)** Standardize levels across a list of factors.
fct_unify(list(f2, f1))

Change the order of levels

fct_relevel(.f, ..., after = 0L**)** Manually reorder factor levels.
fct_relevel(f, c("b", "c", "a"))

fct_infreq(f, ordered = NA**)** Reorder levels by the frequency in which they appear in the data (highest frequency first). Also **fct_inseq()**.

```
f3 <- factor(c("c", "c", "a"))
fct_infreq(f3)
```

fct_inorder(f, ordered = NA**)** Reorder levels by order in which they appear in the data.
fct_inorder(f2)

fct_rev(f) Reverse level order.

```
f4 <- factor(c("a","b","c"))
fct_rev(f4)
```

fct_shift(f) Shift levels to left or right, wrapping around end.
fct_shift(f4)

fct_shuffle(f, n = 1L**)** Randomly permute order of factor levels.
fct_shuffle(f4)

fct_reorder(.f, .x, .fun = median, ..., .desc = FALSE**)** Reorder levels by their relationship with another variable.

```
boxplot(
  PlantGrowth,
  weight ~ fct_reorder(group, weight)
)
```

fct_reorder2(.f, .x, .y, .fun = last2, ..., .desc = TRUE**)** Reorder levels by their final values when plotted with two other variables.

```
ggplot(
  diamonds,
  aes(
    carat, price,
    color = fct_reorder2(color, carat, price)
  )) +
  geom_smooth()
```

Change the value of levels

fct_recode(.f, ...**)** Manually change levels. Also **fct_relabel()** which obeys purrr::map syntax to apply a function or expression to each level.

```
fct_recode(f, v = "a", x = "b", z = "c")
fct_relabel(f, ~ paste0("x", .x))
```

fct_anon(f, prefix = ""**)** Anonymize levels with random integers.
fct_anon(f)

fct_collapse(.f, …, other_level = NULL**)** Collapse levels into manually defined groups.
fct_collapse(f, x = c("a", "b"))

fct_lump_min(f, min, w = NULL, other_level = "Other"**)** Lumps together factors that appear fewer than min times. Also **fct_lump_n(), fct_lump_prop(),** and **fct_lump_lowfreq()**.
fct_lump_min(f, min = 2)

fct_other(f, keep, drop, other_level = "Other"**)** Replace levels with "other."
fct_other(f, keep = c("a", "b"))

Add or drop levels

fct_drop(f, only**)** Drop unused levels.

```
f5 <- factor(c("a","b"),c("a","b","x"))
f6 <- fct_drop(f5)
```

fct_expand(f, …**)** Add levels to a factor.
fct_expand(f6, "x")

fct_na_value_to_level(f, level = "(Missing)"**)** Assigns a level to NAs to ensure they appear in plots, etc.

```
f7 <- factor(c("a", "b", NA))
fct_na_value_to_level(f7, level = "(Missing)")
```

Explore `forcats` using the `gss_cat` dataset

`gss_cat`

```
# A tibble: 21,483 × 9
    year marital         age race  rincome        partyid            relig            denom      tvhours
   <int> <fct>         <int> <fct> <fct>          <fct>              <fct>            <fct>        <int>
 1  2000 Never married    26 White $8000 to 9999  Ind,near rep       Protestant       Southern…       12
 2  2000 Divorced         48 White $8000 to 9999  Not str republican Protestant       Baptist-…       NA
 3  2000 Widowed          67 White Not applicable Independent        Protestant       No denom…        2
 4  2000 Never married    39 White Not applicable Ind,near rep       Orthodox-christian Not appl…        4
 5  2000 Divorced         25 White Not applicable Not str democrat   None             Not appl…        1
 6  2000 Married          25 White $20000 - 24999 Strong democrat    Protestant       Southern…       NA
 7  2000 Never married    36 White $25000 or more Not str republican Christian        Not appl…        3
 8  2000 Divorced         44 White $7000 to 7999  Ind,near dem       Protestant       Lutheran…       NA
 9  2000 Married          44 White $25000 or more Not str democrat   Protestant       Other            0
10  2000 Married          47 White $25000 or more Strong republican  Protestant       Southern…        3
```

Inspecting a factor variable

`fct_count()` **count** the number of observations in each level

`fct_match()` check if an observation **matches** a provided level

Inspecting a factor variable

```
fct_count(<data>)
```

```
fct_count(gss_cat$marital)
```

count observations within each level of “marital”

```
fct_match(<data>, <“level”>)
```

```
fct_match(gss_cat$marital, “Married”)
```

identify observations with a **match** for the level “Married”

Changing the order of factor levels

`fct_rev()` **rev**erse the order of the levels

`fct_shuffle()` randomly **shuffle** the order of the levels

`fct_relevel()` manually **re**-order the **levels**

Changing the order of factor levels

```
fct_rev(<data>)
```

```
fct_rev(gss_cat$marital)
```

reverses the level order of “marital”

```
fct_shuffle(gss_cat$marital)
```

shuffles the level order of “marital”

```
fct_relevel(<data>, <new_lvl_order>)
```

Need to wrap `levels()` around these to see a result!

Changing the order of factor levels

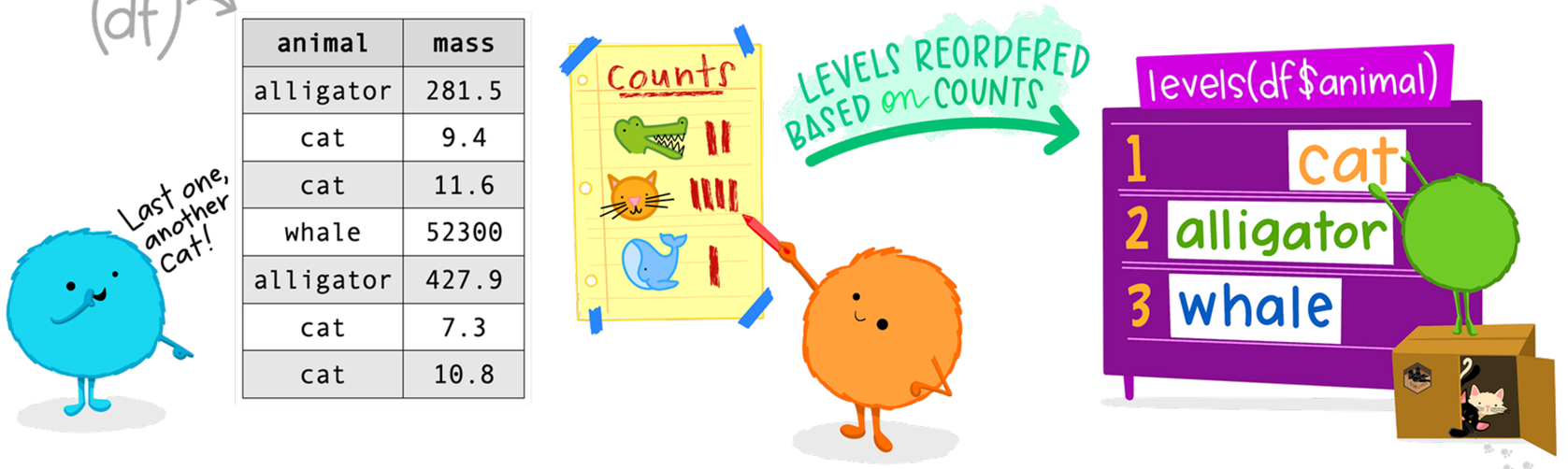
forcats::fct_infreq()

reorder FACTOR LEVELS by # of observations in each level

(default: largest n = 1st level)

(df) →

animal	mass
alligator	281.5
cat	9.4
cat	11.6
whale	52300
alligator	427.9
cat	7.3
cat	10.8



see also:

fct_inorder: reorder by order of appearance

fct_reorder: reorder by another variable

fct_relevel: reorder manually

+ more!

forcats.tidyverse.org

Artwork by Allison Horst

Changing the name of factor levels

`fct_recode()` **recode** the name of a factor level

`fct_lump_min()` **lump** together factor levels based on frequency

Changing the name of factor levels

```
fct_recode(<data>, <new_name> = <"old_name">)
```

```
fct_recode(gss_cat$marital, widowed = "Widowed")
```

recodes the name of the "Widowed" level of "marital" as "widowed"

```
fct_lump_min(<data>, <minimum_count>)
```

```
fct_lump_min(gss_cat$marital, 1000)
```

lumps levels of "marital" with less than 1000 observations into "Other"

Need to wrap `levels()` around these to see a result!

Create a factor variable in a new dataset

you can make a factor column
in your `readr` command!


```
read_tsv()

read_csv()

read_delim()
```

COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor(**levels, ordered = FALSE**)** - "f"
- **col_datetime(**format = ""**)** - "T"
- **col_date(**format = ""**)** - "D"
- **col_time(**format = ""**)** - "t"
- **col_skip()** - "-", "_"
- **col_guess()** - "?"

Create a factor variable in a new dataset

```
read_csv("dataset", col_types = list(
 <column_name> = col_factor()))
```

Introduction to `lubridate`

Artwork by Allison Horst

Why talk about dates and times?

“At first glance, dates and times seem simple. You use them all the time in your regular life, and they don’t cause much confusion. However, the more you learn about dates and times, the more complicated they seem to get!”

R for Data Science

Types of date/time variables in the `tidyverse`

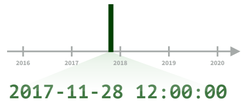
Date	`"2017-11-28"`
Time	`00:01:25`
Date-Time	`"2017-11-28 12:00:00 UTC"`

Important `lubridate` functionality

- Parse columns as dates and/or times
- Pull date/time info from these columns
- Timespan math!

Dates and times with lubridate : : **CHEATSHEET**

Date-times



`2017-11-28 12:00:00`

`2017-11-28 12:00:00`
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

dt <- ***as_datetime(****1511870400****)***
"2017-11-28 12:00:00 UTC"

`2017-11-28`
A **date** is a day stored as the number of days since 1970-01-01

d <- ***as_date(****17498****)***
"2017-11-28"

`12:00:00`
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::****as_hms(****85****)***
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a tz argument to set the time zone, e.g. ymd(x, tz = "UTC").

`2017-11-28T14:02:00` **ymd_hms(), ymd_hm(), ymd_h().** ymd_hms("2017-11-28T14:02:00")

`2017-22-12 10:00:00` **ydm_hms(), ydm_hm(), ydm_h().** ydm_hms("2017-22-12 10:00:00")

`11/28/2017 1:02:03` **mdy_hms(), mdy_hm(), mdy_h().** mdy_hms("11/28/2017 1:02:03")

`1 Jan 2017 23:59:59` **dmy_hms(), dmy_hm(), dmy_h().** dmy_hms("1 Jan 2017 23:59:59")

`20170131` **ymd(), ydm().** ymd(20170131)

`July 4th, 2000` **mdy(), myd().** mdy("July 4th, 2000")

`4th of July '99` **dmy(), dym().** dmy("4th of July '99")

`2001: Q3` **yq()** Q for quarter. yq("2001: Q3")

`07-2020` **my(), ym().** my("07-2020")

`2:01` hms::**hms()** Also lubridate::**hms(), hm()** and **ms()**, which return periods.* hms::hms(seconds = 0, minutes = 1, hours = 2)

`2017.5` **date_decimal(**decimal, tz = "UTC"**)** date_decimal(2017.5)

now(tzone = ""**)** Current time in tz (defaults to system tz). now()

today(tzone = ""**)** Current date in a tz (defaults to system tz). today()

fast_strptime() Faster strptime. fast_strptime("9/1/01", "%y/%m/%d")

parse_date_time() Easier strptime. parse_date_time("09-01-01", "ymd")

GET AND SET COMPONENTS

Use an accessor function to get a component.
```
d ## "2017-11-28"
day(d) ## 28
```

Assign into an accessor function to change a component in place.
```
day(d) <- 1
d ## "2017-11-01"
```

`2018-01-31 11:59:59` **date(**x**)** Date component. date(dt)

`2018-01-31 11:59:59` **year(**x**)** Year. year(dt)
isoyear(x**)** The ISO 8601 year.
epiyear(x**)** Epidemiological year.

`2018-01-31 11:59:59` **month(**x, label, abbr**)** Month. month(dt)

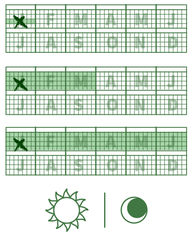
`2018-01-31 11:59:59` **day(**x**)** Day of month. day(dt)
wday(x, label, abbr**)** Day of week.
qday(x**)** Day of quarter.

`2018-01-31 11:59:59` **hour(**x**)** Hour. hour(dt)

`2018-01-31 11:59:59` **minute(**x**)** Minutes. minute(dt)

`2018-01-31 11:59:59` **second(**x**)** Seconds. second(dt)

`2018-01-31 11:59:59 UTC` **tz(**x**)** Time zone. tz(dt)



week(x**)** Week of the year. week(dt)
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

quarter(x**)** Quarter. quarter(dt)

semester(x, with_year = FALSE**)** Semester. semester(dt)

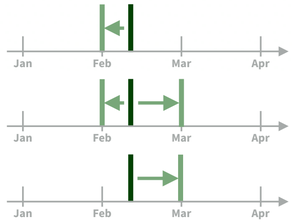
am(x**)** Is it in the am? am(dt)
pm(x**)** Is it in the pm? pm(dt)

dst(x**)** Is it daylight savings? dst(d)

leap_year(x**)** Is it a leap year? leap_year(d)

update(object, ..., simple = FALSE**)** update(dt, mday = 2, hour = 1)

Round Date-times



floor_date(x, unit = "second"**)** Round down to nearest unit. floor_date(dt, unit = "month")

round_date(x, unit = "second"**)** Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL**)** Round up to nearest unit. ceiling_date(dt, unit = "month")

Valid units are second, minute, hour, day, week, month, bimonth, quarter, season, halfyear and year.

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE**)** Roll back to last day of previous month. Also **rollforward()**. rollback(dt)

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date()** and **stamp_time()**.

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

Tip: use a date with day > 12

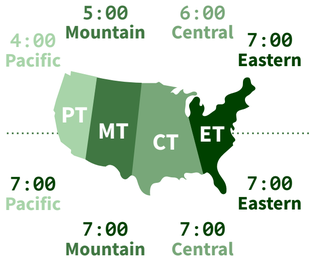
Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. OlsonNames()

Sys.timezone() Gets current time zone.



with_tz(time, tzone = ""**)** Get the **same date-time** in a new time zone (a new clock time). Also **local_time(**dt, tz, units**)**. with_tz(dt, "US/Pacific")

force_tz(time, tzone = ""**)** Get the **same clock time** in a new time zone (a new date-time). Also **force_tzs()**. force_tz(dt, "US/Pacific")

lubridate Cheatsheet

Create a date/time variable in a new dataset

you can make a date/time column
in your `readr` command!

```
read_tsv()

read_csv()

read_delim()
```

COLUMN TYPES

Each column type has a function and corresponding string abbreviation.

- **col_logical()** - "l"
- **col_integer()** - "i"
- **col_double()** - "d"
- **col_number()** - "n"
- **col_character()** - "c"
- **col_factor(**levels, ordered = FALSE**)** - "f"
- **col_datetime(**format = ""**)** - "T"
- **col_date(**format = ""**)** - "D"
- **col_time(**format = ""**)** - "t"
- **col_skip()** - "-", "_"
- **col_guess()** - "?"

Create a date/time variable in a new dataset

```
read_csv("dataset", col_types = list(
 <column_name> = col_date()))
```

`col_time()` `col_date()` `col_datetime()`

Create a date/time variable in a new dataset

Type	Code	Meaning	Example
Year	%Y	4 digit year	2021
	%y	2 digit year	21
Month	%m	Number	2
	%b	Abbreviated name	Feb
	%B	Full name	February
Day	%d	One or two digits	2
	%e	Two digits	02
Time	%H	24-hour hour	13
	%I	12-hour hour	1
	%p	AM/PM	pm
	%M	Minutes	35

Also need to indicate how your date/ time is written

```
col_date("%m/%d/%Y")
```

e.g., for dates like 11/19/2025

	%S	Seconds	45
	%0S	Seconds with decimal component	45.35
	%Z	Time zone name	America/Chicago
	%z	Offset from UTC	+0800
Other	%.	Skip one non-digit	:
	%*	Skip any number of non-digits	

R for Data Science

Convert a character variable to a date

ymd_hms(), ymd_hm(), ymd_h().
ymd_hms("2017-11-28T14:02:00")

ydm_hms(), ydm_hm(), ydm_h().
ydm_hms("2017-22-12 10:00:00")

mdy_hms(), mdy_hm(), mdy_h().
mdy_hms("11/28/2017 1:02:03")

dmy_hms(), dmy_hm(), dmy_h().
dmy_hms("1 Jan 2017 23:59:59")

ymd(), ydm(). ymd(20170131)

mdy(), myd(). mdy("July 4th, 2000")

dmy(), dym(). dmy("4th of July '99")

yq() Q for quarter. yq("2001: Q3")

my(), ym(). my("07-2020")

DATE

y = year

m = month

d = day

TIME

h = hour

m = minute

s = second

lubridate Cheatsheet

Pull date/time info from a date/time variable

```
year(<variable>)
month(<variable>)
day(<variable>)

hour(<variable>)
minute(<variable>)
second(<variable>)
```

date or date/time variables

time or date/time variables

Further reading

- Wickham & Grolemund 2017, *R for Data Science*, Chapter 14
- String manipulation with `stringr` Cheatsheet
- Wickham & Grolemund 2017, *R for Data Science*, Chapter 16
- Factors with `forcats` Cheatsheet
- Wickham & Grolemund 2017, *R for Data Science*, Chapter 17
- Dates and times with `lubridate` Cheatsheet