Package 'tidyEmoji'

September 17, 2026

Type Package

Title Discover, Count, Categorise, Score, Translate and Relate Emoji in Text

Version 0.4.0

Description A tidy toolkit for working with the emoji in any text column, such as social-media posts, product reviews, chat logs or survey responses. Unicode is awkward to handle and not every code point is an emoji, which makes emoji statistics fiddly to obtain. 'tidyEmoji' extracts, counts, categorises, sentiment-scores and emotion-scores emoji, converts them to and from text (for accessibility and NLP preprocessing), searches the emoji catalogue, maps emoji co-occurrence and sequences (graph-ready edge lists and n-grams), measures where and how densely emoji are used, and builds document-by-emoji feature tables for machine learning, with grapheme-aware detection (so skin-tone and multi-person sequences stay intact), returning tidy data frames that slot straight into a 'tidyverse' workflow. It also quantifies how much annotators disagreed about an emoji (interpretation risk), extracts the words around each emoji, tracks emoji use over time, measures text-emoji sentiment mismatch, and applies explicit emoji-preprocessing policies for language-model pipelines. The bundled emoji sentiment lexicon is from the Emoji Sentiment Ranking of Kralj Novak et al. (2015) <doi:10.1371/journal.pone.0144296>, released under CC BY-SA 4.0; the emotion lexicon is from EmoTag1200 of Shoeb & de Melo (2020) <https://aclanthology.org/2020.emnlp-main.720/>, released under the MIT licence.

License GPL (>= 3)

URL https://pursuitofdatascience.github.io/tidyEmoji/, https://github.com/PursuitOfDataScience/tidyEmoji

BugReports https://github.com/PursuitOfDataScience/tidyEmoji/issues

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports dplyr (>= 1.1.0), emoji (>= 16.0.0), lifecycle (>= 1.0.3), rlang, stats, tibble, tidyr (>= 1.3.0), utils

Suggests rmarkdown, knitr, testthat (>= 3.0.0), commonmark, xml2, ggplot2, readr (>= 2.0.0), forcats, stringr

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Youzhi Yu [aut, cre]

Maintainer Youzhi Yu <yuyouzhi666@icloud.com>

Repository CRAN

Date/Publication 2026-09-17 14:50:02 UTC

Contents

as_emoji_name . . . 3
as_emoji_type . . . 5
category_unicode_crosswalk . . . 6
emoji_adoption_lag . . . 6
emoji_ambiguity . . . 8
emoji_categorize . . . 10
emoji_collocations . . . 11
emoji_congruence . . . 13
emoji_context . . . 14
emoji_cooccurrence . . . 16
emoji_density . . . 17
emoji_dfm . . . 18
emoji_emotion . . . 20
emoji_emotion_label . . . 21
emoji_emotion_lexicon . . . 23
emoji_extract_nest . . . 24
emoji_extract_unnest . . . 25
emoji_faceness . . . 26
emoji_filter . . . 27
emoji_flag_ambiguous . . . 28
emoji_frequency . . . 29
emoji_incongruity . . . 30
emoji_incongruity_profile . . . 32
emoji_lexicons . . . 34
emoji_ngrams . . . 35
emoji_pairs . . . 36
emoji_position . . . 37
emoji_provenance . . . 39

emoji_ratio 40
emoji_risk 41
emoji_sanitize 42
emoji_score 45
emoji_search 47
emoji_seasonality 48
emoji_sentiment 50
emoji_sentiment_lexicon 51
emoji_summary 53
emoji_tokens 54
emoji_token_cost 55
emoji_to_text 56
emoji_trend 57
emoji_turnover 59
emoji_type 61
emoji_unicode_crosswalk 62
emoji_unicode_releases 63
emoji_unicode_version 64
emoji_version_profile 65
register_emoji_lexicon 66
text_to_emoji 67
top_n_emojis 68

Index **71**

`as_emoji_name` *Vector helpers: convert emoji to/from names and shortcodes*

Description

Small vector-level helpers for ad-hoc use. They do not take a data frame.

Usage

```
as_emoji_name(x)

as_emoji_shortcode(x)

as_emoji(x)
```

Arguments

`x`	A character vector of emoji glyphs (for `as_emoji_name`, `as_emoji_shortcode`) or of shortcodes/names (for `as_emoji`).

Details

- `as_emoji_name(x)` maps emoji glyphs to their Unicode names.
- `as_emoji_shortcode(x)` maps emoji glyphs to a shortcode.
- `as_emoji(x)` maps names/shortcodes to the emoji glyph (emojize).

All three resolve through `emoji_key()`, so qualified emoji (carrying `U+FE0F`) and unqualified forms resolve identically. Unmatched inputs return `NA`.

`as_emoji()` accepts either namespace in the same argument, and 464 strings belong to both – they are the exact Unicode name of one emoji and a shortcode alias of another. It resolves them in a fixed order: **exact Unicode name first**, then shortcode, then **emoji**'s own name table. An exact name match is the stronger signal, so `as_emoji("dog")` is the emoji actually *named* "dog" (a dog, `U+1F415`), not the one whose alias is `:dog:` (a dog face, `U+1F436`).

For 17 of those 464 strings the two namespaces disagree, and there `as_emoji()` and `text_to_emoji()` differ *by design*: a `:dog:` token is explicitly delimited as a shortcode, so `text_to_emoji()` reads it in the shortcode namespace and produces the dog face. The pattern is an emoji whose name is a bare noun versus the "... face" variant that carries the alias (`cat`, `cow`, `pig`, `tiger`, `mouse`, `rabbit`), or a plain object versus a decorated one (`umbrella`, `snowman`, `calendar`, `sunglasses`). Pass a shortcode through `text_to_emoji()`, or the full Unicode name (`"dog face"`) to `as_emoji()`, if you need one namespace specifically.

Value

A character vector the same length as `x`.

Which shortcode you get

`as_emoji_shortcode()` returns one shortcode *per emoji*, not per spelling: the first alias of the emoji's fully-qualified (RGI) form. That is what makes it agree with `emoji_to_text()` and survive a round trip, but it is not always the first alias of the glyph you passed in. 344 codepoint keys have a different first alias on each of their two spellings, so for 175 of the catalogue's 5042 rows the two answers differ: `as_emoji_shortcode("\u2764")` is `"heart"`, the qualified heart's alias, where the bare `U+2764` row's own first alias is `"red_heart"`.

`emoji_search()` reports the other one – the matched row's own alias, since a search result is a row – so the two verbs can disagree on the same glyph. Both resolve back to the same emoji through `text_to_emoji()`, which reads every alias, so where both answer the disagreement is cosmetic. Where they differ in substance is the 189 rows that have no alias: `emoji_search()` reports `NA` for those, while `as_emoji_shortcode()` still answers for all 189, having borrowed the alias of the glyph's other spelling.

See Also

`emoji_to_text()`, `text_to_emoji()` for the data-frame verbs.

Examples

```
as_emoji_name(c("\U0001f600", "\u2764\ufe0f"))
as_emoji_shortcode(c("\U0001f600", "\u2764\ufe0f"))
as_emoji(c("grinning", "heart"))
```

`as_emoji_type` *Functional type of an emoji glyph*

Description

`as_emoji_type(x)` maps emoji glyphs to a small functional vocabulary – `"face"`, `"gesture"`, `"person"`, `"nature"`, `"food"`, `"place"`, `"activity"`, `"object"`, `"symbol"`, `"flag"`, `"component"` – recoded from the Unicode group and subgroup. The distinction that matters most in the literature is `face` (emotional) against `object` (semantic).

Usage

```
as_emoji_type(x)
```

Arguments

`x`	A character vector of emoji glyphs.

Details

The recode is: faces and costumed characters in *Smileys & Emotion* become `face` and the rest of that group (hearts, the anger symbol, ...) becomes `symbol`; hands and gesturing people in *People & Body* become `gesture` and the rest `person`; the remaining Unicode groups map one-to-one. Glyphs the reference table does not know return `NA`.

Value

A character vector the same length as `x`.

See Also

`emoji_type()` for the data-frame verb, `emoji_faceness()` for the per-row share, `emoji_categorize()` for the raw Unicode categories.

Examples

```
as_emoji_type(c("\U0001f600", "\U0001f44d", "\U0001f355", "\u2764\ufe0f"))
```

category_unicode_crosswalk *Emoji category to unicode crosswalk*

Description

A table with one row per Unicode category, listing every emoji glyph in that category as a single |-separated string.

Usage

```
category_unicode_crosswalk
```

Format

A data frame with two columns:

category The Unicode category (10 categories).

unicodes The emoji glyphs in the category, separated by |.

Source

Derived from the emojis table of the **emoji** package; rebuilt by data-raw/crosswalks.R.

emoji_adoption_lag *How long did this population take to adopt each emoji?*

Description

emoji_adoption_lag() compares the date an emoji was first used in your corpus with the date Unicode released it, giving a per-glyph adoption lag in days.

Usage

```
emoji_adoption_lag(data, text, time)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.

`text` The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

`time` Unquoted column of dates or date-times (`Date`, `POSIXct`, or character in `"YYYY-MM-DD"` form). A date-time is bucketed by the calendar day it *displays* as in its own timezone, not by its UTC day: an emoji posted at 23:30 New York time belongs to that day, not to the next one.

"Its own timezone" means the column's `tzone` attribute. A `POSIXct` created without one – which is what `as.POSIXct("2024-01-01 23:30")` and most CSV readers give you – has no timezone of its own, so R displays it in the session's, and the buckets follow. The same column then gives hour 23 on one machine and hour 4 on another. Tag the column (`as.POSIXct(x, tz = "UTC")`, or `lubridate::force_tz()`) if the result has to be reproducible; a `Date` column is immune either way.

A character column must lead with a four-digit year: `"2024-01-01"` or `"2024/01/01"`, with one- or two-digit month and day, and any trailing time ignored. Values that do not parse warn and are dropped – but a column in which *nothing* reads as a date is an error rather than a column of `NA`, since there would be no time axis left. Note that `"01/02/2024"` is in the second group: convert a column written that way with `as.Date()` and its own `format` first.

Details

A lag is only as good as the corpus window: an emoji released before your data begins will look adopted on day one, so read the lag together with `n` and the span of your data. Negative lags mean the corpus contains a glyph before its official release date – usually a vendor shipping early, or a timestamp problem worth investigating.

Occurrences whose time is missing or unparseable are dropped.

Glyphs are canonicalised through the package's codepoint key, so two spellings of one emoji make one row rather than two; see *Which spelling comes back* in tidyEmoji.

Value

A tibble with one row per emoji, most frequent first and ties broken by the glyph so the order is fully determined: `emoji`, `name`, `n`, `version`, `release_date`, `first_seen` and `lag_days`. `lag_days` is `NA` when the release date of the version is unknown.

See Also

`emoji_version_profile()`, `emoji_unicode_releases()`.

Examples

```
df <- data.frame(
```

```
  when = as.Date(c("2021-01-01", "2022-06-01")),
  text = c("\U0001f600", "\U0001f97a")
)
emoji_adoption_lag(df, text, when)
```

`emoji_ambiguity` *How ambiguous is each emoji?*

Description

`emoji_ambiguity()` reports, for every emoji in the Emoji Sentiment Ranking (see emoji_sentiment_lexicon), how much its human annotators disagreed about whether it was negative, neutral or positive. Miller et al. (2016) found that readers of the same rendering disagree about a quarter of the time; the bundled lexicon keeps the raw annotation counts behind its collapsed score, so that disagreement can be reported as a number rather than described as a caveat.

Usage

```
emoji_ambiguity(x = NULL, measure = "entropy")
```

Arguments

`x`	Optional character vector of emoji glyphs to report on. The default, `NULL`, returns every emoji in the lexicon, most ambiguous first. Glyphs absent from the lexicon come back with `NA` statistics.
`measure`	Which ambiguity statistic to put in the `ambiguity` column: one of `"entropy"` (default), `"gini"`, `"neutral_share"` or `"ci_width"`.

Details

The four measures are computed from the annotation shares (`p_neg, p_neu, p_pos`):

- `"entropy"` (the default) is Shannon entropy in nats: 0 when the annotators were unanimous, `log(3)` (about 1.0986) when they split evenly three ways.
- `"gini"` is the Gini impurity, `1 - sum(p^2)`: 0 when unanimous, 2/3 at maximum disagreement.
- `"neutral_share"` is `p_neu` on its own, for the "is this emoji simply uninformative?" question.
- `"ci_width"` is the width of a 95% Wald interval around the glyph's sentiment score. Unlike the other three it shrinks as the number of annotations grows, so it answers "how well do we know this score?" rather than "how much do readers disagree?".

`rank` is always computed over the whole lexicon (1 = most ambiguous), so a rank keeps its meaning when `x` selects a handful of glyphs.

Read `n_annotations` before you read the ranking. The lexicon's annotation counts are wildly uneven – the median glyph has 18, and 69% have fewer than 50 – and the first three measures are shape statistics that do not care how many annotations produced the shape. A glyph seen by three annotators who split one-one-one scores the maximum entropy of `log(3)` on that evidence alone,

which is why five of the rows tied at `rank = 1` have 3, 3, 3, 9 and 15 annotations, and why 11 of the top 20 have fewer than 50.

Three of those five are not emoji at all. The lexicon was built from 2015 tweets and 233 of its 969 rows are characters absent from the reference table – box-drawing characters, dingbats, enclosed letters – so the head of the ranking can show a glyph such as `U+250C` that no corpus this package analyses will ever yield. They carry 6% of the lexicon's annotations and 86% of them have fewer than 50, so the `n_annotations` filter recommended here removes 200 of the 233 as a side effect. See the *Detection limitations* section of emoji_sentiment_lexicon for what the rest of them are.

The bias is at the *top* of the ranking specifically, not across it: a thinly annotated glyph is usually unanimous, so entropy is positively correlated with the annotation count overall (Spearman 0.56). What three annotators can do that thousands cannot is hit the exact maximum. So filter on `n_annotations` before interpreting the head of the table, as the introduction vignette does. That advice applies to `"ci_width"` too, and the reason is worth stating plainly, because the obvious reading of a confidence width is that it has already accounted for thin evidence:

`"ci_width"` is a **Wald** interval, so it scales as `1 / sqrt(n)` only *at a given spread*, and it is exactly zero wherever the spread is zero. For a glyph whose annotators were unanimous the estimated variance is 0 whatever `n` is, so the interval has zero width on one annotation just as on eight thousand. That is the textbook degeneracy of the Wald interval at a boundary proportion, not a property of the data: **166 of the lexicon's 969 rows report `ci_width = 0`, and their annotation counts run from 1 to 68.** So `ci_width` does not rescue a thin glyph – ranked ascending it puts the thinnest unanimous ones first, as the most certain rows in the table. `n_annotations` remains the column to filter on; `ci_width` separates well-known from poorly-known scores only among glyphs that are not unanimous.

The same degeneracy runs the other way at the top of the scale. A Wald width is not bounded by the range of the quantity it measures, so **4 of the 969 rows report a `ci_width` wider than 2** while the sentiment score itself can only run from -1 to 1. All four have 3 annotations. Read that as "this score is not known at all" rather than as a width, and filter on `n_annotations` as above.

Value

A tibble with columns `emoji`, `key` (the codepoint-normalised join key), `n_annotations`, `p_neg`, `p_neu`, `p_pos`, `ambiguity` and `rank`. `rank` is 1 for the most ambiguous emoji; glyphs with identical `ambiguity` **share** the lowest rank of their group and the next distinct value skips ahead accordingly (`rank()`'s `ties.method = "min"`), so ranks are not necessarily consecutive. With `x = NULL` rows are ordered by `rank` with ties broken by the glyph, so the order is deterministic; with `x` supplied the result has one row per element of `x`, in the same order.

References

Miller H, Thebault-Spieker J, Chang S, Johnson I, Terveen L, Hecht B (2016). "Blissfully Happy" or "Ready to Fight": Varying Interpretations of Emoji. *ICWSM 2016*.

See Also

`emoji_risk()` for the per-row version, `emoji_flag_ambiguous()` for the emoji in your own corpus, and `emoji_sentiment()` with `se = TRUE` for the uncertainty around a score.

Examples

```
head(emoji_ambiguity())

# the head of that table is glyphs a handful of annotators disagreed about;
# filter on n_annotations before reading it as a finding
amb <- emoji_ambiguity()
head(amb[amb$n_annotations >= 500, ])

emoji_ambiguity(c("\U0001f602", "\U0001f643"))
head(emoji_ambiguity(measure = "ci_width"))
```

`emoji_categorize` *Categorise each row by the emoji categories it contains*

Description

`emoji_categorize()` keeps the rows of `data` that contain emoji and adds a `.emoji_category` column listing the distinct Unicode categories present in that row (for example "Smileys & Emotion"), separated by `|` when a row spans more than one category.

Usage

```
emoji_categorize(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

A row is kept because it contains an emoji, not because that emoji could be categorised. If none of a row's emoji is in the reference table the row is kept with `.emoji_category` set to `NA`. Dropping it would silently shrink the corpus, and by exactly the rows a user whose Unicode coverage is behind most needs to see.

Three different things reach that `NA`, and only the first is a catalogue question:

1. A zero-width-joiner sequence **newer than your installed emoji package**, since detection is grapheme-aware and does not require the sequence to be catalogued. Use `emoji_provenance()` to check which catalogue you are matching against; upgrading `emoji` fixes it.
2. An **invalid regional-indicator pair** such as `U+1F1FD U+1F1FD`. It is a well-formed grapheme cluster and no catalogue will ever contain it, so no upgrade helps.
3. A **non-RGI ZWJ join** – components joined in a combination Unicode does not recommend. These arrive as separate occurrences rather than one.

See the Detection section of tidyEmoji for both false-positive cases.

Value

`data`, as a tibble, filtered to the rows containing at least one emoji, with an added `.emoji_category` column. That column is `NA` for a row whose emoji are all absent from the reference table.

See Also

`emoji_type()` for the smaller functional vocabulary recoded from the same group and subgroup, and `emoji_faceness()` for the face share it reduces to; `emoji_frequency()` for per-glyph counts carrying `group`; category_unicode_crosswalk for the ten categories themselves.

Examples

```
df <- data.frame(text = c("smile \U0001f600",
                          "flag \U0001f3c1\U0001f600",
                          "nothing"))
emoji_categorize(df, text)
```

---

emoji_collocations *Which words keep company with which emoji*

Description

`emoji_collocations()` counts the words that appear near each emoji across a corpus and scores the association with pointwise mutual information. It is the corpus-derived alternative to importing a fixed sense inventory: the senses come from *your* texts, so they cannot be stale and carry no licence baggage.

Usage

```
emoji_collocations(
  data,
  text,
  window = 5,
  min_n = 3,
  measure = c("pmi", "count")
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
window	Context window on each side, in words. Default `5`.
min_n	Minimum number of co-occurrences for a pair to be reported. Default `3`.
measure	Sort order: `"pmi"` (default) or `"count"`. Both columns are always returned.

Details

Each emoji occurrence contributes its context window (see `emoji_context()`). A word is counted once per occurrence however often it repeats inside that window. Words are lower-cased and stripped of leading and trailing punctuation; no stopword list is applied, because which stopwords are right is a decision for your analysis, not for this package – filter the result with **tidytext**'s `stop_words` if you want one.

PMI is `log(n(e, w) * N / (n(e) * n(w)))`, with `N` the total number of emoji-word co-occurrence events. Marginals are computed over *all* co-occurrences before `min_n` filters the rows, so a rare pairing is scored against the full corpus rather than against the surviving subset.

Glyphs are canonicalised through the package's codepoint key, so qualified and unqualified forms of the same emoji share one row.

Value

A tibble with columns `emoji`, `word`, `n` (co-occurrences) and `pmi`, shaped like `widyr::pairwise_count()` output so it drops into existing tidytext workflows. Rows are sorted by `measure` descending, then by the other of the two descending, then by the glyph and the word, so the order is fully determined. Ties in `pmi` are common: every pair seen the same number of times with the same marginals scores alike.

See Also

`emoji_context()` for the occurrence-level windows this aggregates.

Examples

```
df <- data.frame(text = c("cold coffee \U0001f622",
                          "coffee again \U0001f622",
```

```
                            "warm tea \U0001f60a"))
emoji_collocations(df, text, min_n = 1)
```

`emoji_congruence` *Text-emoji congruence*

Description

`emoji_congruence()` is `emoji_incongruity()` under the framing used in the marketing and eWOM literature, where the finding is that a *mismatch* between a review's words and its emoji lowers perceived helpfulness and authenticity. Same engine, same columns, plus `.emoji_congruent`.

Usage

```
emoji_congruence(
  data,
  text,
  text_score,
  method = c("difference", "sign_flip"),
  scale,
  where = c("all", "final"),
  threshold = 1
)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`text_score`	Unquoted numeric column holding the text's own sentiment.
`method`	`"difference"` (default) for the continuous gap, or `"sign_flip"` for the categorical polarity-flip feature.
`scale`	How to make the two scores comparable: `"rank"`, `"zscore"` or `"none"`. Required – there is no sensible default. `"rank"` and `"zscore"` are computed over the rows carrying both an emoji score and a `text_score`, not over the whole corpus, so rows with no scorable emoji cannot shift the answer for the rows that have one.

where	"all" (default) scores every emoji in the row; "final" scores only the trailing run of emoji that ends the text.
threshold	For method = "difference", the absolute gap at or above which .emoji_incongruent is TRUE. Default 1, a full polarity swing on the rank scale. "sign_flip" has no gap to cut, so supplying both warns rather than letting half the call do nothing silently. A threshold at or below 0 flags every scored row, abs(gap) never being negative.

Value

data, as a tibble, with everything emoji_incongruity() adds plus .emoji_congruent, the negation of .emoji_incongruent.

See Also

emoji_incongruity().

Examples

```
df <- data.frame(
  text = c("lovely stay \U0001f600", "terrible room \U0001f600"),
  score = c(0.8, -0.9)
)
emoji_congruence(df, text, score, scale = "none")
```

---

emoji_context *The text around each emoji occurrence*

Description

emoji_context() returns one row per emoji occurrence with a window of the text on either side of it. It is the primitive the context-dependent analyses need: emoji are polysemous, and what a glyph means in a message is decided by its co-text, not by a lexicon.

Usage

```
emoji_context(
  data,
  text,
  window = 5,
  unit = c("word", "char"),
  keep_text = FALSE
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
window	Size of the context window on each side, in tokens (unit = "word") or characters (unit = "char"). Default 5.
unit	"word" (default) or "char". "Character" means *code point*, the unit nchar() and substr() count and the one emoji_density() and emoji_ratio() measure in. A character window can therefore begin or end part-way through a grapheme cluster: where an e carries a combining acute (U+0065 U+0301), window = 1 returns the bare U+0301, a diacritic with nothing to sit on. Emoji themselves are safe from this, being masked out whole (see Details), and so is unit = "word". If the window is going to be read by a person rather than tokenised, ask for words, or for a few more characters than you need.
keep_text	If TRUE, also return the row's original text column. Default FALSE.

Details

Windows are taken from the text with *all* emoji blanked out, so a neighbouring emoji never lands in a context window and character offsets stay exact. With unit = "word" a token is a maximal run of non-whitespace characters, the same definition emoji_density() uses; with unit = "char" the window is a literal code-point count after trimming the whitespace next to the emoji.

Tokenisation stops there on purpose. If you need stemming, stopword removal or sentence splitting, pass the result to **tokenizers** or **tidytext** rather than expecting this verb to grow a tokeniser.

Value

A tibble with one row per emoji occurrence, in reading order, and columns .row_number (position of the entry in data), .position (the code-point offset at which the emoji starts, the unit substr() takes, so substr(text, .position, .position + nchar(.emoji) - 1) is the glyph; emoji_ngrams()'s column of the same name counts emoji instead), .emoji, .emoji_context_left, .emoji_context_right and .emoji_context (the two sides joined by a space – the co-text without the glyph). Rows with no emoji contribute nothing. The columns of data are not carried, so a grouping is not either – join back on .row_number to recover them.

See Also

emoji_collocations() for the corpus-level view; emoji_position() for where emoji sit in a text.

Examples

```
df <- data.frame(text = c("the coffee was cold \U0001f622 again",
                          "no emoji here"))
emoji_context(df, text, window = 2)
emoji_context(df, text, window = 6, unit = "char")
```

---

emoji_cooccurrence *Emoji co-occurrence counts, with an optional diagonal*

Description

emoji_cooccurrence() is emoji_pairs() under the name the matrix form goes by, with one argument added and one taken away.

Usage

```
emoji_cooccurrence(data, text, doc_id = NULL, diagonal = FALSE, sort = TRUE)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
doc_id	Optional unquoted column identifying documents. Rows sharing a value are treated as one document. Default: each row is a document. The result has a row per pair, so it grows with the *square* of the distinct emoji in a document: a day or a conversation is cheap, and pooling a whole corpus under one id is not. 800 distinct emoji in one document is 319,600 pairs and a few seconds; 3790 would be 7.2 million.
diagonal	If TRUE, include one item1 == item2 row per emoji with its document frequency. Default FALSE.

`sort` If `TRUE` (default), sort by descending `n` (ties broken by `item1`, `item2` so the order is deterministic). `FALSE` sorts by `item1` then `item2` instead – still a fixed order, computed in the C locale, not the order the pairs happened to be counted in.

Details

Added: `diagonal = TRUE` also returns the `item1 == item2` rows, whose `n` is the number of documents containing that emoji (the diagonal of the co-occurrence matrix, i.e. its document frequency).

Taken away: there is no `directed` here. A co-occurrence matrix is symmetric, so an ordered pair has no meaning on it and the diagonal this verb exists to add would not either. Use `emoji_pairs()` when you want `directed = TRUE`; the off-diagonal rows the two verbs return are otherwise identical.

Glyphs are canonicalised through the package's codepoint key, so two spellings of one emoji make one row rather than two; see *Which spelling comes back* in tidyEmoji.

Value

A tibble with columns `item1`, `item2` and `n`.

See Also

`emoji_pairs()`, `emoji_ngrams()`.

Examples

```
df <- data.frame(text = c("\U0001f602\U0001f60d", "\U0001f602"))
emoji_cooccurrence(df, text, diagonal = TRUE)
```

`emoji_density` *Emoji density per character and per token*

Description

`emoji_density()` measures how emoji-heavy each text is: the number of emoji per character and per whitespace-delimited token. Rows with no emoji get densities of 0; rows whose text is `NA` or empty get `NA`.

Usage

```
emoji_density(data, text)
```

Arguments

`data` A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.

`text` The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

"Character" here means *code point*, the unit `nchar()` counts, so a multi-code-point emoji inflates the denominator by all of its code points. The same visible text therefore gives different answers depending on how the emoji is built: `"hi <emoji>"` is four graphemes either way, but `.emoji_per_char` is 0.25 for a single-code-point smiley, 0.200 for a two-code-point flag and 0.100 for a seven-code-point ZWJ family. It is not exotic – 115 of the 560 emoji-bearing rows in the corpus behind the introduction vignette contain a multi-code-point emoji.

This is the same basis `emoji_ratio()` uses and states, and the opposite of the one `emoji_position()` uses: `.emoji_rel_position` counts each emoji as one position, because a proportion of the message has to. If you want a density that does not move with an emoji's internal length, `.emoji_per_token` is immune – all three examples above give 0.5.

Value

`data`, as a tibble, with added columns `.emoji_n`, `.emoji_per_char` (emoji per character, i.e. per code point, of text) and `.emoji_per_token` (emoji per whitespace-delimited token). A token is a maximal run of characters outside Unicode's `White_Space` property; see `emoji_ratio()` for the exact set, which does not vary with the locale.

See Also

`emoji_position()`, `emoji_ratio()`.

Examples

```
df <- data.frame(text = c("hi \U0001f600", "\U0001f600\U0001f600", "plain"))
emoji_density(df, text)
```

---

emoji_dfm *Document-by-emoji feature matrix*

Description

`emoji_dfm()` turns a text column into a wide, model-ready table with one row per document and one column per emoji, weighted by raw counts, binary presence or tf-idf. All documents are kept, including those with no emoji (all-zero rows), so the result aligns row-for-row with the corpus and can be bound to outcome columns for tidymodels-style workflows.

Usage

```
emoji_dfm(data, text, doc_id = NULL, weighting = c("count", "binary", "tfidf"))
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`doc_id`	Optional unquoted column identifying documents; rows sharing a value are aggregated into one document. Default: each row is a document.
`weighting`	One of `"count"` (default), `"binary"` or `"tfidf"`.

Details

By default every row of `data` is a document and the first output column, `.row_number`, is its position in `data` (matching `emoji_extract_unnest()`). Give `doc_id` to aggregate rows sharing an id into one document; the id column keeps its name, and documents appear in the order their id is first seen in `data`, never in the session's collation order. Emoji columns are named by the glyph itself, canonicalised through the package's codepoint key (so qualified and unqualified forms count as one feature), and ordered by descending total count (ties broken by glyph).

For `weighting = "tfidf"`, the cell for emoji e in document d is `count(d, e) * log(N / df(e))`, where `N` is the number of documents and `df(e)` the number of documents containing e. An emoji that appears in every document therefore scores 0.

Value

A tibble with one row per document: `.row_number` (or the `doc_id` column) followed by one numeric column per emoji, ordered by descending total count across the corpus with ties broken by the glyph. That ordering is computed in the C locale, so the column order does not depend on the session's collation and is safe to index by position. Zero emoji in the corpus yields just the document column.

See Also

`emoji_frequency()` for corpus totals; `emoji_tokens()` for the long form this widens.

Examples

```
df <- data.frame(text = c("\U0001f600\U0001f600 fun", "\U0001f621",
                          "no emoji"))
emoji_dfm(df, text)
emoji_dfm(df, text, weighting = "binary")
emoji_dfm(df, text, weighting = "tfidf")
```

`emoji_emotion` *Emoji emotion profiles (the 8 Plutchik emotions)*

Description

`emoji_emotion()` scores each row's emoji across the eight Plutchik emotions (anger, anticipation, disgust, fear, joy, sadness, surprise, trust) using the bundled EmoTag1200 lexicon (Shoeb & de Melo, 2020). Scores each range from 0 to 1 and are averaged over the emoji in the row that appear in the lexicon.

Usage

```
emoji_emotion(data, text, lexicon = "emotag1200", long = FALSE)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`lexicon`	Lexicon to use. Either a string naming a bundled lexicon (`"emotag1200"`, the default), the name of a registered lexicon (see `register_emoji_lexicon()`), or a data frame. A custom lexicon must have an `emoji` column and one column per emotion (any subset of the eight Plutchik emotions); it is joined through the same codepoint-normalised key as the bundled one.
`long`	If `TRUE`, return one row per (row, emotion) in long form with columns `.emoji_emotion` (the emotion name) and `.emoji_score` (its mean). Default `FALSE` adds eight `.emoji_<emotion>` columns plus `.emoji_n` and `.emoji_n_scored`.

Details

The lexicon is 150 glyphs, about 4% of the distinct emoji tidyEmoji can detect, so a row of post-2018 emoji will score NA and still be a row full of emoji. Read .emoji_n_scored alongside .emoji_n before concluding a corpus carries no emotion; see emoji_emotion_lexicon for the figure and its denominator.

Value

data, as a tibble. With long = FALSE (the default), eight emotion columns – .emoji_anger, .emoji_anticipation, .emoji_disgust, .emoji_fear, .emoji_joy, .emoji_sadness, .emoji_surprise, .emoji_trust – plus .emoji_n and .emoji_n_scored, one row per input row. With long = TRUE, one row per input row *per emotion*, carrying .emoji_emotion and .emoji_score in place of the eight columns **and of the two counts** – the long form returns neither .emoji_n nor .emoji_n_scored. Rows without emoji, or whose emoji are absent from the lexicon, receive NA scores.

.emoji_n_scored is what tells those two apart, as in emoji_sentiment(): 0 means the row had emoji the lexicon could not score, NA that it had no emoji to score. Since the long form omits it, read the counts from a long = FALSE call on the same data (the rows are in the same order) when the distinction matters – on a 150-glyph lexicon it usually does.

References

Shoeb AAM, de Melo G (2020). EmoTag1200: Understanding the Association between Emojis and Emotions. *EMNLP 2020*. https://aclanthology.org/2020.emnlp-main.720/. Data released under the MIT licence.

See Also

emoji_emotion_lexicon for the underlying scores; emoji_emotion_label() for the dominant emotion per row; emoji_sentiment() for valence.

Examples

```
df <- data.frame(text = c("love it \U0001f60d", "scary \U0001f628", "meh"))
emoji_emotion(df, text)
emoji_emotion(df, text, long = TRUE)
```

---

emoji_emotion_label *The dominant emoji emotion per row*

Description

emoji_emotion_label() adds .emoji_emotion, the emotion with the highest mean score among the row's emoji (using emoji_emotion()). Ties are broken in Plutchik order; a row with nothing scorable, or with no emotion ahead of the others, receives NA.

Usage

```
emoji_emotion_label(data, text, lexicon = "emotag1200")
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
lexicon	Passed to emoji_emotion().

Details

Ties are broken in Plutchik order – the order the eight emotions are listed in throughout the package (anger, anticipation, disgust, fear, joy, sadness, surprise, trust) – so the winner is deterministic and does not depend on the row's position in the data. It happens: **3 of the bundled lexicon's 150 glyphs tie for their top emotion**, and because Plutchik order is alphabetical the tie-break quietly favours the early names. U+1F3A4 scores anticipation and joy at 0.39 and is labelled anticipation; U+1F619 scores joy and trust at 0.83 and is labelled joy. So read .emoji_n_scored alongside the label, and reach for emoji_emotion() when a near-tie would change your reading: a single winning name cannot show one.

A row whose scored emotions are *all* equal is the one case with no winner to break a tie between, and it gets NA rather than the first name in the order. An emoji scored zero on all eight is the obvious example. That needs a custom lexicon to reach, the bundled one having no such glyph, and .emoji_n_scored still separates it from a row with nothing to score.

Value

data, as a tibble, with .emoji_emotion (the winning emotion, or NA when nothing was scorable) added, alongside the .emoji_n and .emoji_n_scored counts it inherits from emoji_emotion(). The eight per-emotion columns are *not* returned – the label is the point – unless they were already in data, which is what emoji_emotion() |> emoji_emotion_label() gives you: the profile and the label side by side.

See Also

emoji_emotion() for the eight scores this collapses, and the coverage caveat that applies to both; emoji_emotion_lexicon for the underlying data; emoji_sentiment() for valence instead of emotion.

Examples

```
df <- data.frame(text = c("love it \U0001f60d", "scary \U0001f628", "meh"))
```

```
emoji_emotion_label(df, text)
```

`emoji_emotion_lexicon` *Emoji emotion lexicon (EmoTag1200)*

Description

Human-annotated emotion-association scores (each from 0 to 1) for the eight Plutchik emotions (anger, anticipation, disgust, fear, joy, sadness, surprise, trust), for the 150 most popular Twitter emoji, from EmoTag1200.

Usage

```
emoji_emotion_lexicon
```

Format

A data frame with one row per emoji and the columns:

key Codepoint-normalised key (U+FE0F stripped) for robust joining.

emoji The emoji glyph (unqualified form, as stored by the source).

name The emoji's Unicode name.

anger, anticipation, disgust, fear, joy, sadness, surprise, trust Emotion-association scores, each from 0 to 1.

How much of the catalogue this covers

150 glyphs, about 4% of the distinct emoji tidyEmoji can detect (3790 distinct codepoint keys in the reference table of **emoji** 16.0.0; see `emoji_provenance()` for the version you have). That is not a defect – EmoTag1200 is a carefully annotated 150-glyph resource – but it is worth knowing *before* concluding that a corpus carries no emotion: a modern corpus is mostly post-2018 glyphs that no bundled lexicon has seen. `emoji_emotion()` reports this per row rather than hiding it: `.emoji_n_scored` is `0` when a row has emoji the lexicon cannot score, and `NA` only when the row has no emoji at all.

Source

Shoeb AAM, de Melo G (2020). EmoTag1200: Understanding the Association between Emojis and Emotions. *EMNLP 2020*. https://aclanthology.org/2020.emnlp-main.720/. Data from https://github.com/abushoeb/EmoTag, released under the MIT licence. Processed by `data-raw/emoji_emotion_lexicon.R`.

`emoji_extract_nest` *Add a list-column of the emoji found in each row*

Description

`emoji_extract_nest()` returns `data` unchanged except for an added list-column, `.emoji_unicode`, holding the emoji found in each row. Detection is grapheme-aware, so skin-tone modifiers and ZWJ sequences (for example family emoji) are kept intact as a single emoji.

Usage

```
emoji_extract_nest(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

`data`, as a tibble, with an added list-column `.emoji_unicode`. A grouped input stays grouped.

See Also

`emoji_extract_unnest()` for a long, counted form and `emoji_tokens()` for one row per emoji with metadata.

Examples

```
df <- data.frame(text = c("hi \U0001f600\U0001f603", "none"))
emoji_extract_nest(df, text)
```

`emoji_extract_unnest` *Emoji counts per row, in long (tidy) form*

Description

`emoji_extract_unnest()` returns one row per (row, emoji) pair with a count, dropping rows that contain no emoji. `.row_number` refers to the position of the entry in `data`.

Usage

```
emoji_extract_unnest(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

A tibble with columns `.row_number`, `.emoji_unicode` and `.emoji_count`. The columns of `data` are not carried, so a grouping is not either – join back on `.row_number` to recover them. `emoji_extract_nest()` keeps your rows, and your grouping, instead.

See Also

`emoji_extract_nest()` for the same emoji as a list-column that keeps your rows, and `emoji_tokens()` for one row per occurrence with metadata attached; `emoji_frequency()` for corpus-level counts.

Examples

```
df <- data.frame(text = c("hi \U0001f600\U0001f600", "none", "\U0001f44b"))
emoji_extract_unnest(df, text)
```

`emoji_faceness` *How face-heavy is each row's emoji use?*

Description

`emoji_faceness()` reports the share of a row's emoji that are faces. Face emoji act as emotional signals and object emoji as semantic ones, and the two have measurably different effects on engagement, so the split is worth a column of its own.

Usage

```
emoji_faceness(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

`data`, as a tibble, with added columns `.emoji_n`, `.emoji_n_typed` (emoji whose type is known), `.emoji_n_face` and `.emoji_faceness` (`.emoji_n_face / .emoji_n_typed`). Rows with no emoji get `NA`.

`.emoji_n_typed` distinguishes the two ways a share can be missing, as `.emoji_n_scored` does in `emoji_sentiment()`: `0` means the row had emoji whose type the recode does not know, `NA` that it had no emoji at all. `.emoji_faceness` is `NA` in both cases – a share of no typable emoji is not 0, it is unknown – so read the count before the share. See `emoji_type()` for when a glyph can be untypable.

See Also

`emoji_type()`, `as_emoji_type()`.

Examples

```
df <- data.frame(text = c("\U0001f600\U0001f355", "\U0001f600", "none"))
emoji_faceness(df, text)
```

`emoji_filter` *Keep only the rows whose text contains emoji*

Description

`emoji_filter()` returns the rows of `data` whose text column contains at least one emoji, preserving every original column. `emoji_tweets()` is a synonym retained for backward compatibility.

Usage

```
emoji_filter(data, text)

emoji_tweets(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

A tibble containing only the rows with at least one emoji, with every original column kept. A grouped input stays grouped, as it would through `dplyr::filter()`.

See Also

`emoji_summary()` for the counts this filter is derived from; `emoji_ratio()` to find the rows that are *only* emoji; `emoji_extract_unnest()` and `emoji_tokens()` for the emoji themselves.

Examples

```
df <- data.frame(text = c("hi \U0001f600", "no emoji", "bye \U0001f44b"))
emoji_filter(df, text)
```

`emoji_flag_ambiguous` *Which emoji in this corpus are most likely to be misread?*

Description

`emoji_flag_ambiguous()` crosses the emoji actually present in a text column with their annotation-disagreement statistics and returns the most ambiguous ones first. It is the content-QA shortlist: the glyphs worth a second look before a campaign ships or a coding scheme is fixed.

Usage

```
emoji_flag_ambiguous(data, text, top_n = 10, measure = "entropy")
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`top_n`	Number of emoji to return, most ambiguous first. `NULL` returns all of them. When a tie straddles the cut the glyph decides which emoji fall inside it, in the C locale, as in `top_n_emojis()`; a corpus with fewer emoji than this returns every one of them rather than padding, and `0` returns no rows at all.
`measure`	Ambiguity statistic to rank by; see `emoji_ambiguity()`.

Value

A tibble with columns `emoji`, `name`, `n` (occurrences in the corpus), `n_annotations`, `ambiguity` and `rank` (the glyph's rank in the whole lexicon, carried over from `emoji_ambiguity()`, so tied glyphs share a rank). Rows are ordered by descending `ambiguity`, then descending `n`, then the glyph, so the order is deterministic. Emoji absent from the lexicon cannot be ranked and are dropped.

See Also

`emoji_ambiguity()`, `emoji_risk()`.

Examples

```
df <- data.frame(text = c("ok \U0001f643", "yay \U0001f600 \U0001f643",
                          "hmm \U0001f612"))
emoji_flag_ambiguous(df, text, top_n = 3)
```

`emoji_frequency` *Frequency of every emoji in a text column*

Description

`emoji_frequency()` counts how often each emoji appears across the whole text column (an entry containing the same emoji twice contributes 2) and returns a tibble sorted by descending count, with each emoji's name, shortcode and category.

Usage

```
emoji_frequency(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

A tibble with columns `emoji`, `name`, `shortcode`, `group` and `n`, sorted by descending `n` with ties broken by the glyph so the order is deterministic.

See Also

`top_n_emojis()` for just the most frequent emoji.

Examples

```
df <- data.frame(text = c("\U0001f600\U0001f600", "\U0001f621"))
emoji_frequency(df, text)
```

`emoji_incongruity` *Text-emoji sentiment mismatch*

Description

`emoji_incongruity()` measures the signed gap between the sentiment a row's emoji carry and the sentiment of its text. It is the sarcasm-detection feature the NLP literature keeps rediscovering, and the mismatch variable the marketing literature calls (in)congruence.

Usage

```
emoji_incongruity(
  data,
  text,
  text_score,
  method = c("difference", "sign_flip"),
  scale,
  where = c("all", "final"),
  threshold = 1
)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`text_score`	Unquoted numeric column holding the text's own sentiment.
`method`	`"difference"` (default) for the continuous gap, or `"sign_flip"` for the categorical polarity-flip feature.
`scale`	How to make the two scores comparable: `"rank"`, `"zscore"` or `"none"`. Required – there is no sensible default. `"rank"` and `"zscore"` are computed over the rows carrying both an emoji score and a `text_score`, not over the whole corpus, so rows with no scorable emoji cannot shift the answer for the rows that have one.
`where`	`"all"` (default) scores every emoji in the row; `"final"` scores only the trailing run of emoji that ends the text.

threshold — For method = "difference", the absolute gap at or above which .emoji_incongruent is TRUE. Default 1, a full polarity swing on the rank scale. "sign_flip" has no gap to cut, so supplying both warns rather than letting half the call do nothing silently. A threshold at or below 0 flags every scored row, abs(gap) never being negative.

Details

.emoji_incongruity is emoji - text after scaling, so it is positive when the emoji is the more positive of the two. "sign_flip" is the categorical version most sarcasm papers use and is computed on the *unscaled* scores, where the sign means something.

A row with no scorable emoji gets NA, never 0: a neutral emoji and no emoji at all are different states, and collapsing them silently biases every downstream model. The same applies to a missing text_score, and to an infinite one – a scorer that overflows is reported and treated as missing rather than left to turn every other row's z-score into Inf.

With where = "final" only the run of emoji that ends the text is scored: both the illocutionary-force account of emoji and the P600 evidence on ironic emoji are specifically about sentence-final glyphs. A text whose emoji sit mid-sentence then has nothing eligible to score, so it gets NA and .emoji_n_scored = NA, while .emoji_n still counts every emoji in the row.

"Ends the text" is literal: only whitespace may follow the last glyph, so "great \U0001f602" has a final run and "great \U0001f602." does not – a trailing full stop, bracket or quote mark disqualifies it. The run itself extends back over any glyphs separated from each other by whitespace alone, so "great \U0001f602 \U0001f60d" contributes both. If your corpus punctuates after emoji, strip trailing punctuation before scoring, or use where = "all".

Value

data, as a tibble, with added columns .emoji_n, .emoji_n_scored, .emoji_sentiment, .emoji_incongruity, .emoji_polarity_flip and .emoji_incongruent.

.emoji_n_scored distinguishes the two ways the answer can be missing, as it does in emoji_sentiment(): 0 means the row had emoji that the lexicon could not score, NA that it had no emoji to score. The four derived columns are NA in both cases, and also wherever text_score itself is missing or not finite.

You supply the text score

tidyEmoji deliberately does not score text. text_score is a column you produce with **tidytext** and AFINN or Bing, **sentimentr**, **vader**, or a transformer – which keeps the method choice visible in your script instead of buried in this package, and keeps our dependency footprint where it is.

Because those methods live on wildly different scales (AFINN runs -5 to 5, VADER -1 to 1, a model's logits on nothing in particular), scale has no default: you have to say how the two sides were made comparable. "rank" maps both to percentiles on [-1, 1] and is the safest choice for cross-method comparison; "zscore" standardises both; "none" compares the raw numbers, which is only meaningful if your text score already lives on the emoji lexicon's -1 to 1 scale.

"rank" and "zscore" are computed over the rows the comparison is defined on – those carrying both an emoji score and a text_score – not over the whole corpus. A percentile only means something relative to a population, and the population the gap lives in is the scored subset, so rows

with no scorable emoji cannot move the answer for the rows that have one. Subsetting the data to the scored rows before calling therefore gives the same numbers as calling on everything.

That invariance is narrow, and it is worth seeing where it stops. Both scalings are *relative*, so a gap's size depends on how many scored rows it was computed over and not only on the two scores. Dropping rows that were never scored changes nothing, because they were never in the population; dropping or adding *scored* rows changes every other row's answer, and so does comparing two corpora of different sizes. The sharpest case is duplicating a scored corpus exactly, which adds no information whatsoever and still multiplies every rank gap by `2 * (n - 1) / (2 * n - 1)`: 0.933 at `n = 8`, 0.995 at `n = 100`, 0.999 at `n = 500`. (`"zscore"` moves the other way, by `sqrt((2 * n - 1) / (2 * (n - 1)))`, because `sd()` divides by `n - 1`.) So compare gaps within one call, and where a number has to travel between corpora use `scale = "none"` with a text score already on the emoji lexicon's -1 to 1 scale.

References

An emoji centric approach to sarcasm detection in online discourse. *Scientific Reports* (2025). The influence of emoji meaning multipleness on perceived online review helpfulness. *Journal of Business Research* (2022).

See Also

`emoji_congruence()` for the same engine under the marketing framing; `emoji_incongruity_profile()` for which glyphs go against the grain; `emoji_sentiment()` for the emoji side on its own.

Examples

```
df <- data.frame(
  text = c("this is wonderful \U0001f621", "awful \U0001f621", "great \U0001f600"),
  score = c(0.9, -0.8, 0.7)
)
emoji_incongruity(df, text, score, scale = "none")
emoji_incongruity(df, text, score, scale = "none", method = "sign_flip")
```

---

emoji_incongruity_profile

Which emoji go against the grain of their text?

Description

`emoji_incongruity_profile()` aggregates `emoji_incongruity()` by glyph: for each emoji, how far from its host text's sentiment it typically sits, and how often it appears with the opposite polarity. Those are the candidate irony markers in your corpus.

Usage

```
emoji_incongruity_profile(
  data,
  text,
  text_score,
  method = c("difference", "sign_flip"),
  scale,
  where = c("all", "final"),
  threshold = 1,
  min_n = 5
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
text_score	Unquoted numeric column holding the text's own sentiment.
method	`"difference"` (default) for the continuous gap, or `"sign_flip"` for the categorical polarity-flip feature.
scale	How to make the two scores comparable: `"rank"`, `"zscore"` or `"none"`. Required – there is no sensible default. `"rank"` and `"zscore"` are computed over the rows carrying both an emoji score and a `text_score`, not over the whole corpus, so rows with no scorable emoji cannot shift the answer for the rows that have one.
where	`"all"` (default) scores every emoji in the row; `"final"` scores only the trailing run of emoji that ends the text.
threshold	For `method = "difference"`, the absolute gap at or above which `.emoji_incongruent` is `TRUE`. Default `1`, a full polarity swing on the rank scale. `"sign_flip"` has no gap to cut, so supplying both warns rather than letting half the call do nothing silently. A threshold at or below `0` flags every scored row, `abs(gap)` never being negative.
min_n	Minimum number of scored occurrences for an emoji to be reported. Default `5`.

Details

Incongruity is a property of a row, so every emoji in a row is credited with that row's gap. A glyph that habitually shares a message with a genuinely incongruent one will therefore inherit some of its score; read `n` alongside `flip_rate` before drawing conclusions from a handful of occurrences.

Glyphs are canonicalised through the package's codepoint key, so two spellings of one emoji make one row rather than two; see *Which spelling comes back* in tidyEmoji.

Value

A tibble with one row per emoji: `emoji`, `name`, `n` (scored occurrences), `mean_incongruity`, `sd_incongruity`, `n_flips` and `flip_rate`, sorted by descending `flip_rate`, then descending `n`, then the glyph, so the order is fully determined. The first key ties readily: every glyph that never flips scores `0`, and one that always does scores `1`.

See Also

`emoji_incongruity()`.

Examples

```
df <- data.frame(
  text = c("great \U0001f621", "lovely \U0001f621", "awful \U0001f621"),
  score = c(0.8, 0.7, -0.9)
)
emoji_incongruity_profile(df, text, score, scale = "none", min_n = 1)
```

---

emoji_lexicons *List bundled emoji lexicons*

Description

`emoji_lexicons()` returns a tibble describing the lexicons bundled with tidyEmoji and any user-registered ones: their name, type (sentiment or emotion), dimensions, number of emoji, source and licence.

Usage

```
emoji_lexicons()
```

Value

A tibble with columns `name`, `type`, `dimensions`, `n`, `source`, `licence`.

`n` is the lexicon's **row count**. For the two bundled ones that is also the number of emoji they score, 969 and 150, because each has one row per code-point key. A registered lexicon need not: two spellings of one emoji are two rows and score one glyph, and a row whose glyph yields no key at all (an empty string, an `NA`) is counted here and matched never. `length(unique(emoji_key(tbl$emoji)))` is the count of distinct emoji, the same distinction `emoji_provenance()` draws for `n_emoji`.

See Also

register_emoji_lexicon() to add your own; emoji_score() to score text against any lexicon.

Examples

```
emoji_lexicons()
```

---

emoji_ngrams *Consecutive emoji sequences (n-grams)*

Description

emoji_ngrams() slides a window of n over each row's emoji, in reading order (any text between the emoji is ignored), and returns one row per n-gram occurrence. Repeated emoji are kept: a row containing the same emoji twice in a row yields a bigram of that emoji with itself. This is the emoji analogue of tidytext::unnest_tokens(..., token = "ngrams") and feeds sequence / Markov-style analyses of how emoji chain together.

Usage

```
emoji_ngrams(data, text, n = 2, sep = " ")
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
n	Length of the n-gram window. Default 2 (bigrams).
sep	Separator between the glyphs of an n-gram. Default a space.

Details

Glyphs are canonicalised through the package's codepoint key, so two spellings of one emoji make one n-gram token rather than two; see *Which spelling comes back* in tidyEmoji.

Value

A tibble with columns `.row_number` (position of the entry in `data`), `.position` (where the n-gram starts within the row's emoji sequence) and `.emoji_ngram`. Rows with fewer than `n` emoji contribute nothing. The columns of `data` are not carried, so a grouping is not either – join back on `.row_number` to recover them. Unlike the corpus-wide verbs this one does not pool your rows, so there is no per-group answer being silently turned into a global one.

See Also

`emoji_pairs()` for order-free co-occurrence; `emoji_extract_unnest()` for the underlying one-emoji-per-row form.

Examples

```
df <- data.frame(text = c("\U0001f602\U0001f60d\U0001f389", "\U0001f602"))
emoji_ngrams(df, text)
emoji_ngrams(df, text, n = 3)
```

`emoji_pairs` *Co-occurring emoji pairs*

Description

`emoji_pairs()` returns a tidy edge list of the emoji that appear together in the same document: one row per pair with the number of documents in which the pair co-occurs. By default every row of `data` is a document; give `doc_id` to treat all rows sharing an id (a conversation, a user, a day) as one document. The output mirrors `widyr::pairwise_count()` (`item1`, `item2`, `n`) and pipes straight into `igraph::graph_from_data_frame()`, tidygraph or ggraph.

Usage

```
emoji_pairs(data, text, doc_id = NULL, directed = FALSE, sort = TRUE)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

doc_id — Optional unquoted column identifying documents. Rows sharing a value are treated as one document. Default: each row is a document.

The result has a row per pair, so it grows with the *square* of the distinct emoji in a document: a day or a conversation is cheap, and pooling a whole corpus under one id is not. 800 distinct emoji in one document is 319,600 pairs and a few seconds; 3790 would be 7.2 million.

directed — If TRUE, pairs are ordered by first appearance: a document where the tears-of-joy emoji appears before the heart-eyes emoji counts towards (tears-of-joy, heart-eyes), not the reverse. Default FALSE (unordered pairs, with item1 sorted before item2).

sort — If TRUE (default), sort by descending n (ties broken by item1, item2 so the order is deterministic). FALSE sorts by item1 then item2 instead – still a fixed order, computed in the C locale, not the order the pairs happened to be counted in.

Details

Glyphs are canonicalised through the package's codepoint key, so qualified and unqualified forms of the same emoji (with/without U+FE0F) count as one node. Pairs are between *distinct* emoji: repeats of the same emoji in a document do not pair with themselves (see emoji_cooccurrence() for the diagonal).

Value

A tibble with columns item1, item2 and n. Empty (but typed) when no document contains two distinct emoji.

See Also

emoji_cooccurrence() for the same counts with an optional diagonal; emoji_ngrams() for consecutive sequences.

Examples

```
df <- data.frame(text = c("fun \U0001f602\U0001f60d",
                          "\U0001f602\U0001f60d\U0001f389",
                          "just \U0001f602"))
emoji_pairs(df, text)
emoji_pairs(df, text, directed = TRUE)
```

---

emoji_position *Where do emoji sit within each text?*

Description

emoji_position() reports, for each row, the character position of the first and last emoji and the mean *relative* position of all emoji occurrences, from 0 (the very start of the text) to 1 (the very end). The Emoji Sentiment Ranking (Kralj Novak et al., 2015) tracks the same relative position, and it is a studied signal: emoji cluster near the end of messages.

Usage

```
emoji_position(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

`.emoji_first` and `.emoji_last` are code-point offsets, the unit `substr()` uses, so they can be fed straight back to it.

`.emoji_rel_position` is *not* measured in code points. Each emoji counts as one position however many code points it is built from, so an emoji that is the last thing in the text scores 1 whether it is a single-code-point smiley, a two-code-point flag or a seven-code-point family. Counting code points instead put a sentence-final family emoji a third of the way through its message. Everything that is not an emoji still counts one position per code point, so a combining accent elsewhere in the text counts twice; that affects the denominator only, and only for text carrying such marks.

A text that collapses to a *single* position cannot tell its start from its end, and `.emoji_rel_position` is `0` there by convention. That is exactly the row whose whole content is one emoji and nothing else, which a chat or reaction corpus is full of, so the filter this column exists for (`.emoji_rel_position > 0.8`, "the emoji ends the message") skips every one of them. `emoji_ratio()`'s `.emoji_only` finds that family of rows. One character either side is enough to resolve the ambiguity the convention settles: a trailing space scores the emoji `0`, a leading one scores it `1`, both on the ordinary path.

Positions are in *logical* (storage) order, not visual order. In a right-to-left script an emoji that is logically last renders at the reader's left, so "final" here means final in the string, not final on the screen.

Value

`data`, as a tibble, with added columns `.emoji_n`, `.emoji_first` and `.emoji_last` (code-point offsets where the first/last emoji start) and `.emoji_rel_position` (mean relative position in `[0, 1]`, counting each emoji as one position). Rows without emoji get `NA` positions.

See Also

emoji_density() and emoji_ratio() for intensity metrics.

Examples

```
df <- data.frame(text = c("\U0001f600 leading", "trailing \U0001f600",
                          "none"))
emoji_position(df, text)
```

---

emoji_provenance *Versions behind an emoji analysis, in one row*

Description

emoji_provenance() reports every version an emoji result depends on: tidyEmoji itself, the **emoji** package supplying the reference table, the Unicode emoji version that table reflects, the size of that table, and the bundled lexicons. It is meant to be pasted into a methods section or stored beside a result.

Usage

```
emoji_provenance()
```

Details

None of these are cosmetic. A glyph released after your **emoji** package was built is not detected at all; a lexicon covers a few hundred of the thousands of emoji that exist; and "we analysed emoji sentiment" without a lexicon name is not a reproducible statement. See emoji_lexicons() for the lexicons in detail and emoji_unicode_version() for the Unicode version on its own.

n_emoji counts **rows of the reference table**, which are spellings, not distinct emoji. With **emoji** 16.0.0 it is 5042, and those 5042 rows carry only 3790 distinct code-point keys, because an emoji whose presentation can be selected appears both with and without U+FE0F. A methods section reporting "5042 emoji" therefore overstates the vocabulary by the 1252 duplicate spellings; length(unique(emoji_reference is the count of distinct emoji, and 212 of the 5042 spellings are not detectable in text as written at all (see emoji_sentiment_lexicon for why). No *emoji* is lost to that: every one of the 3790 keys is reachable through at least one detectable spelling. The two lexicon strings count their tables' rows the same way.

Value

A one-row tibble with columns tidyEmoji, emoji_pkg, unicode_emoji, n_emoji (rows of the reference table – see Details), sentiment_lexicon, emotion_lexicon and R.

See Also

emoji_unicode_version(), emoji_unicode_releases(), emoji_lexicons().

Examples

```
emoji_provenance()
```

`emoji_ratio` *What share of the text is emoji, and is it emoji-only?*

Description

`emoji_ratio()` reports, per row, the share of the text's characters that belong to emoji, and whether the text is emoji-only (nothing left after removing emoji and whitespace). "Emoji-only" messages are a studied signal in social-media research and a useful filter in practice.

Usage

```
emoji_ratio(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

The ratio is computed over characters (code points), so a multi-code-point emoji (a ZWJ family, a skin-tone sequence) contributes all of its characters.

Value

`data`, as a tibble, with added columns `.emoji_ratio` (emoji characters / all characters, 0 when there are no emoji) and `.emoji_only` (`TRUE` when the text contains emoji and nothing else but whitespace). "Whitespace" is Unicode's `White_Space` property, written out explicitly rather than left to the C library: it includes the no-break spaces `U+00A0` and `U+202F` and the ideographic space `U+3000`, and excludes the zero-width space `U+200B`, which despite its name Unicode does not classify as whitespace. The same set is used everywhere the package splits or trims text, and it does not vary with the locale. `NA` text gets `NA` in both. Empty text (`""`) has no characters to take a share of, so `.emoji_ratio` is `NA` there too, but `.emoji_only` is `FALSE`: an empty string is not a row of emoji.

See Also

emoji_position(), emoji_density(); emoji_filter() to keep emoji-bearing rows.

Examples

```
df <- data.frame(text = c("\U0001f600\U0001f389", "half \U0001f600", "no"))
emoji_ratio(df, text)
```

---

emoji_risk *Interpretation risk per row*

Description

emoji_risk() scores how likely each row's emoji are to be *misread*, using the annotation-disagreement statistics of emoji_ambiguity(). It is the content-QA counterpart of emoji_sentiment(): a row can carry a confident positive score built entirely out of glyphs its annotators fought over.

Usage

```
emoji_risk(data, text, measure = "entropy", threshold = NULL)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
measure	Ambiguity statistic to use; see emoji_ambiguity().
threshold	Value at or above which a glyph counts as ambiguous. NULL (default) uses the lexicon's upper quartile of measure.

Details

threshold decides what counts as an ambiguous glyph for .emoji_n_ambiguous. The default, NULL, uses the upper quartile of the chosen measure across the whole lexicon, i.e. "in the most-disputed quarter of all emoji". Supply your own number to make the cut-off explicit in your script.

Emoji absent from the lexicon cannot be scored and are excluded from the means; .emoji_n and .emoji_n_scored together show how much of the row was actually measured.

Value

data, as a tibble, with added columns .emoji_n, .emoji_n_scored, .emoji_ambiguity_mean, .emoji_ambiguity_max and .emoji_n_ambiguous. Rows with no emoji get NA throughout. A row that has emoji the lexicon cannot score gets .emoji_n_scored = 0, .emoji_n_ambiguous = 0 and NA for the two averages – there is nothing to average, but the count of ambiguous glyphs found is genuinely zero.

See Also

emoji_ambiguity(), emoji_flag_ambiguous().

Examples

```
df <- data.frame(text = c("thanks \U0001f643", "great \U0001f600", "plain"))
emoji_risk(df, text)
```

---

emoji_sanitize *Apply an explicit emoji policy to a text column*

Description

emoji_sanitize() rewrites a text column under one named policy: keep the emoji, delete them, spell them out as names or shortcodes, or replace them with a placeholder token. The value is not new capability – most of it exists across emoji_to_text() and the extraction verbs – but a single argument that says which choice was made, so that "we replaced emoji with their Unicode names" becomes a reproducibility statement rather than a forgotten line of gsub().

Usage

```
emoji_sanitize(
  data,
  text,
  policy = "keep",
  placeholder = "[emoji]",
  wrap = ":{x}:"
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.

`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`policy`	One of `"keep"` (default), `"strip"`, `"name"`, `"shortcode"` or `"placeholder"`.
`placeholder`	Replacement token for `policy = "placeholder"`. Default `"[emoji]"`. Ignored, and not even validated, under the other four policies.
`wrap`	Template for `policy = "shortcode"`, with `{x}` standing for the shortcode. Default `":{x}:"`. Ignored under the other four policies, as it is in `emoji_to_text()` for `format = "name"`.

Details

The policies:

- `"keep"` returns the text untouched. It is the honest baseline for an A/B comparison, and it means the policy argument can stay in the script even when the answer is "do nothing".
- `"strip"` deletes the emoji. Because deleting a glyph can leave two spaces where there was one, `strip` also collapses runs of spaces and tabs and trims the ends – the only policy that touches anything but the emoji. Removing a span makes its two neighbours adjacent, and on malformed input those two can spell an emoji the original text did not contain (a bare `U+2603` beside an orphan `U+FE0F` becomes the qualified snowman), so `strip` repeats until there is nothing left to remove. It is the one policy whose result is emoji-free whatever you hand it.
- `"name"` and `"shortcode"` substitute the Unicode name ("grinning face") or the GitHub-style alias (":grinning:"), exactly as `emoji_to_text()` does – including its rule that a glyph with no known name or alias is **left in place unchanged**. A ZWJ sequence too new for the installed catalogue is detected but cannot be named, so these two policies can return a column that still holds emoji. `"strip"` never does. Neither does `"placeholder"`, unless the token you supply is itself emoji-forming: one that *is* an emoji survives by definition, and a lone combining character such as `U+FE0F` binds to whatever the removed glyph was standing next to. `name` is also the accessibility answer: it is what a screen reader announces.
- `"placeholder"` substitutes a fixed token, which keeps the *position* of an emoji as a feature while removing its identity. An empty `placeholder` is a deletion rather than a substitution, so it gets `"strip"`'s repeat pass (but not its whitespace tidying).

Replacements go exactly where the glyph was, with no padding, so a grinning face glued to the end of a word yields `"wordgrinning face"`. If your tokeniser needs whitespace around them, use `"placeholder"` with a padded placeholder such as `" [emoji] "`.

`placeholder` and `wrap` belong to one policy each and are ignored by the rest, silently and without being validated. That is deliberate rather than an oversight: the point of this verb is that `policy` can be a variable, so a script sweeping all five with one `placeholder =` set would otherwise be warned at four fifths of its calls. Nothing is lost either way, since an ignored argument cannot change the answer. Contrast `emoji_incongruity()`'s `threshold`, which *does* warn when `method = "sign_flip"` makes it inert: that one is a number the caller chose in order to change a result, not a companion belonging to a branch.

Value

`data`, as a tibble, with the text column rewritten in place (same column name). `NA` entries stay `NA`.

Which policies can be undone

The five policies are not five parallel options: they are a ladder of information loss, and how far down it you step is invisible until you try to put the emoji back after the model call.

policy	`"great <U+1F600> work"` becomes	Restorable with `text_to_emoji()`?	What is lost
`"keep"`	`great <U+1F600> work`	yes	nothing
`"shortcode"`	`great :grinning: work`	**yes**	nothing
`"name"`	`great grinning face work`	no	the delimiters; the name is no
`"placeholder"`	`great [emoji] work`	no	*which* emoji – the position su
`"strip"`	`great work`	no	that there was an emoji at all

So if the pipeline has to restore emoji downstream, `"shortcode"` is the only policy that permits it, and it holds up on the awkward cases: skin-tone modifiers, flags, ZWJ sequences and keycaps all come back. Measured against the whole reference table of **emoji** 16.0.0: for all 3790 emoji in their canonical (fully qualified) spelling – the spelling a keyboard emits and text normally holds – **the round trip returns the original text byte for byte, 100% of the time**.

That row of the table assumes the default `wrap`. Restoring the text means `text_to_emoji()` can find the token, and it reads exactly `:shortcode:` – a colon, the alias, a colon. So `wrap` is part of the reversibility contract, not a cosmetic choice:

- `wrap = ":{x}:"` (the default) restores the original text exactly.
- A `wrap` that decorates the token – `"[:{x}:]"`, `":{x}:!"` – brings the emoji back but leaves the decoration behind, so the emoji is restored and the text is not.
- A `wrap` with no `:token:` at all – `"{x}"`, `"<{x}>"`, `"@{x}@"`, `":{x}"` – restores nothing: the shortcode stays in the text as an ordinary word. This is the case to watch, because it fails silently.
- `wrap = "::{x}::"` is worse than either: the inner `:grinning:` matches, so the emoji comes back wrapped in the leftover colons (`:<U+1F600>:`) and a further round trip keeps adding to them.

Change `wrap` for readability by all means, but not on a column you intend to restore.

Unicode also lists shorter spellings of the same emoji, with the `U+FE0F` presentation selectors omitted. Feed one of those in and the round trip returns the *canonical* spelling instead: `U+270C` comes back as `U+270C U+FE0F`. Across all 4853 catalogued spellings that is 79.5% byte-identical, and the remaining 20.5% differ by `U+FE0F` alone – never by more. The emoji is always the same emoji, and every tidyEmoji lookup treats the two spellings as one, so this matters only if you are diffing raw bytes on text that had its selectors stripped upstream.

`"placeholder"` keeps *where* but not *which*, which is enough to use "an emoji was here" as a model feature and not enough to reconstruct the text. `"name"` is the accessibility answer rather than the reversible one – it is what a screen reader announces.

See Also

emoji_token_cost() for what the emoji are costing you; emoji_to_text() for the name/shortcode rewrite on its own.

Examples

```
df <- data.frame(text = c("ship it \U0001f680", "no emoji"))
emoji_sanitize(df, text, policy = "strip")
emoji_sanitize(df, text, policy = "name")
emoji_sanitize(df, text, policy = "placeholder")
```

---

emoji_score *Score emoji in a text column against any lexicon*

Description

emoji_score() is the generic scorer that the friendly verbs (emoji_sentiment(), emoji_emotion()) sit on top of. It joins each row's emoji to lexicon through emoji_key() and returns the per-row mean of the score column, plus the number of emoji scored. Bring your own lexicon, or name a bundled / registered one.

Usage

```
emoji_score(data, text, lexicon = "novak2015", by = "emoji", score = NULL)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
lexicon	Either a string naming a bundled or registered lexicon, or a data frame. For data frames, by names the glyph column and score the score column. Defaults to "novak2015", matching emoji_sentiment(). Two requirements on a data frame, both refused rather than worked around. The score column must be numeric or logical: as text every score comes back NA while the emoji still counts as scored, which contradicts .emoji_n_scored below. And no two rows may give one emoji *different* scores – spellings differing

only by a variation selector share a single code-point key, so a table listing both U+2764 and U+2764 U+FE0F has one emoji twice. Identical scores are fine and collapse silently; when they differ, the row order would be choosing the answer.

A third value is neither refused nor used: an infinite score warns and is treated as missing, so the emoji carrying it counts as unscored. It is the same rule emoji_incongruity() applies to a non-finite text_score, and the same one NA and NaN already got. Left alone, one Inf makes every row that meets it infinite while .emoji_n_scored still reports the row as scored. This applies to an emotion lexicon's dimensions too, and those must be numeric for the reason the score column must be.

by — Glyph column name when lexicon is a data frame, as a single string. Default "emoji". Ignored when lexicon names a bundled or registered lexicon, which carries its own key.

score — Score column name when lexicon is a data frame. If NULL, "sentiment_score" then "score" are tried. Ignored, like by, when lexicon is a name rather than a table.

Value

data, as a tibble, with .emoji_n (total emoji), .emoji_n_scored (emoji found in the lexicon) and .emoji_score (per-row mean) added, in that order – the same order emoji_sentiment(), emoji_emotion(), emoji_risk(), emoji_incongruity() and emoji_faceness() use.

For the multi-dimensional "emotag1200" lexicon the score is the mean over its eight emotion dimensions; use emoji_emotion() for the per-emotion profile. **Note the scale changes with the lexicon.** "novak2015" is a signed valence on [-1, 1], where the sign is the direction of sentiment. The "emotag1200" mean is an *intensity* on [0, 1]: its eight dimensions are each non-negative and four of them (anger, disgust, fear, sadness) are negatively valenced, so a maximally angry emoji and a maximally joyful one score alike and neither is negative. The two are not comparable and must not be pooled.

That averaging is specific to the bundled lexicon. A *registered* or inline lexicon carrying emotion columns has no score column, so emoji_score() cannot collapse it and says so: pass it to emoji_emotion() instead, or name one dimension with score = "joy" to score on that alone.

.emoji_n_scored distinguishes the two ways a score can be missing, as in emoji_sentiment(): 0 means the row had emoji that the lexicon could not score, NA that it had no emoji to score. .emoji_n counts every emoji either way.

See Also

emoji_lexicons(), register_emoji_lexicon().

Examples

```
df <- data.frame(text = c("love \U0001f60d", "angry \U0001f621", "meh"))
emoji_score(df, text, lexicon = "novak2015")

# a bring-your-own lexicon
own <- data.frame(emoji = c("\U0001f600", "\U0001f621"),
                  score = c(0.9, -0.8))
```

```
emoji_score(df, text, lexicon = own)
```

`emoji_search` *Search emoji by keyword, name or shortcode*

Description

`emoji_search()` finds emoji whose Unicode keywords, name or shortcodes match a query (case-insensitive, substring match). It returns a tidy tibble of matches with the glyph, name, shortcode, category and the matching keywords, ready for further inspection or piping into other verbs.

Usage

```
emoji_search(query)
```

Arguments

`query`	A search string, matched as a case-insensitive substring against keywords, name and shortcodes.

Details

`shortcode` is the *matched row's* first alias, and it is `NA` when the matched spelling has none: 189 of the catalogue's 5042 rows carry no GitHub-style alias at all, so a search that hits one (7 of the 198 rows `emoji_search("face")` returns, for instance) has nothing to put in that column. Use the `emoji` column for those, or `as_emoji_shortcode()`, which is keyed on the emoji rather than on the row and so can borrow the alias of the glyph's other spelling. For the same reason the two can disagree even when both answer – see `as_emoji_shortcode()`.

Every non-`NA` `shortcode` is a token `text_to_emoji()` reads, and it recovers the matched row's emoji exactly. `as_emoji()` resolves a bare string by Unicode name first, so for the 17 strings that name one emoji and alias another it returns the emoji of that *name* rather than the row you searched. They are `calendar`, `camel`, `cat`, `cow`, `dog`, `horse`, `kiss`, `mouse`, `pig`, `rabbit`, `satellite`, `snowman`, `sunglasses`, `tiger`, `train`, `umbrella` and `whale`; that is the complete set, not a sample of it. See `as_emoji()` for why.

Value

A tibble with columns `emoji`, `name`, `shortcode`, `group` and `keyword` (the keywords of the emoji that contained the match, collapsed with `, `). `keyword` is the empty string, not `NA`, when the query matched the name or a shortcode rather than a keyword. `shortcode` *is* `NA` when the matched spelling has no alias; see Details.

See Also

`text_to_emoji()` to turn `shortcode` back into a glyph; `as_emoji_name()` for the name of a glyph.

Examples

```
emoji_search("happy")
emoji_search("heart")
```

`emoji_seasonality` *Cyclical patterns in emoji use*

Description

`emoji_seasonality()` aggregates emoji use by month of year, day of week or hour of day. Emoji use is strongly seasonal and strongly diurnal, and both are confounders worth seeing before any trend is interpreted.

Usage

```
emoji_seasonality(data, text, time, period = c("month", "weekday", "hour"))
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`time`	Unquoted column of dates or date-times (`Date`, `POSIXct`, or character in `"YYYY-MM-DD"` form). A date-time is bucketed by the calendar day it *displays* as in its own timezone, not by its UTC day: an emoji posted at 23:30 New York time belongs to that day, not to the next one. "Its own timezone" means the column's `tzone` attribute. A `POSIXct` created without one – which is what `as.POSIXct("2024-01-01 23:30")` and most CSV readers give you – has no timezone of its own, so R displays it in the session's, and the buckets follow. The same column then gives hour 23 on one machine and hour 4 on another. Tag the column (`as.POSIXct(x, tz = "UTC")`, or `lubridate::force_tz()`) if the result has to be reproducible; a `Date` column is immune either way. A character column must lead with a four-digit year: `"2024-01-01"` or `"2024/01/01"`, with one- or two-digit month and day, and any trailing time ignored. Values that do not parse warn and are dropped – but a column in which *nothing* reads as a

date is an error rather than a column of NA, since there would be no time axis left. Note that "01/02/2024" is in the second group: convert a column written that way with as.Date() and its own format first.

period — "month" (default), "weekday" or "hour". "hour" needs a POSIXct/POSIXlt time column.

Details

Every level of the cycle is returned, including the empty ones, so a bar chart has no invisible gaps. Labels are fixed English abbreviations rather than locale-dependent ones, so the output of a script does not change with the machine that runs it. Weeks start on Monday.

Rows whose time is missing or unparseable contribute nothing, as in emoji_trend(). That is worth knowing here in particular, because this table is complete whether or not the data is: every count in it is over the *dated* rows, so sum(n_texts) is the number of rows carrying a readable time rather than nrow(data), and an emoji sitting in an undated row reaches neither n_emoji nor share. emoji_summary() counts the corpus itself if that is what you want to compare against.

Value

A tibble with one row per level of the cycle: .period (integer: 1-12, 1-7 with Monday first, or 0-23), .period_label, n_texts, n_with_emoji, n_emoji, emoji_per_text and share (this level's share of all emoji tokens).

emoji_per_text is n_emoji / n_texts, so the average is over **every** text in the level and not only over the ones carrying an emoji. The distinction is large on a mixed corpus: four emoji spread over three texts, one of which carries all four, gives 1.33 here and would give 4 the other way. Both denominators are in the table, so divide by n_with_emoji yourself for the intensity among users of emoji, and read emoji_per_text as a rate over the whole level. It is NA, not 0, where the level holds no text at all.

See Also

emoji_trend() for the calendar-time view.

Examples

```
df <- data.frame(
  when = as.Date(c("2024-01-05", "2024-01-20", "2024-07-03")),
  text = c("\U0001f600", "\U0001f600\U0001f602", "plain")
)
emoji_seasonality(df, text, when)
```

`emoji_sentiment` *Score the sentiment of the emoji in each row*

Description

`emoji_sentiment()` adds the mean emoji sentiment of each row, based on the Emoji Sentiment Ranking lexicon (see emoji_sentiment_lexicon). Scores range from -1 (negative) through 0 (neutral) to +1 (positive). Rows that contain no emoji, or whose emoji are absent from the lexicon, receive `NA`.

Usage

```
emoji_sentiment(data, text, lexicon = "novak2015", se = FALSE)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`lexicon`	Lexicon to use. The default, `"novak2015"`, uses the bundled emoji_sentiment_lexicon. A registered lexicon (see `register_emoji_lexicon()`) or a data frame can also be supplied; see `emoji_score()` for the generic scorer.
`se`	If `TRUE`, also return `.emoji_sentiment_se`, the standard error of the row's mean sentiment. Requires the bundled `"novak2015"` lexicon. Default `FALSE`.

Details

The lexicon covers about 19% of the distinct emoji tidyEmoji can detect, and nothing added to Unicode after 2015, so `NA` is a common and meaningful answer. `.emoji_n_scored` reports the shortfall per row: `0` means the row had emoji that the lexicon could not score, `NA` that it had none at all. See emoji_sentiment_lexicon for the figure and its denominator.

Detection is grapheme-aware. Some lexicon entries are stored as unqualified, text-presentation code points (notably the bare heart, `U+2764`, without the `U+FE0F` variation selector); those are not treated as emoji in your text, so they are neither counted nor scored. Supply the emoji-presentation (qualified) form and it resolves normally. See emoji_sentiment_lexicon for the full picture.

Value

data, as a tibble, with added columns .emoji_n (the number of emoji in the row), .emoji_n_scored (the number of emoji that actually appear in the lexicon), and .emoji_sentiment (the mean sentiment of the scored emoji). With se = TRUE, also .emoji_sentiment_se.

Uncertainty

A glyph annotated eight times should not carry the same authority as one annotated eight thousand times, and the bundled lexicon keeps the annotation counts that say which is which. With se = TRUE the result gains .emoji_sentiment_se, the standard error of the row's mean: each glyph's score has a binomial-style standard error computed from its own counts, and those are propagated to the mean assuming independent annotations (sqrt(sum(se^2)) / n_scored). It needs the annotation counts, so it is available for the bundled "novak2015" lexicon only. See emoji_ambiguity() for the same counts read as disagreement.

References

Kralj Novak P, Smailovic J, Sluban B, Mozetic I (2015) Sentiment of Emojis. PLoS ONE 10(12): e0144296. doi:10.1371/journal.pone.0144296

See Also

emoji_sentiment_lexicon for the underlying scores; emoji_score() for scoring against any lexicon; emoji_emotion() for discrete emotions; emoji_ambiguity() for annotator disagreement.

Examples

```
df <- data.frame(text = c("love it \U0001f60d", "awful \U0001f621", "meh"))
emoji_sentiment(df, text)
emoji_sentiment(df, text, se = TRUE)
```

---

emoji_sentiment_lexicon

Emoji Sentiment Ranking lexicon

Description

Sentiment scores for emoji, from the *Emoji Sentiment Ranking 1.0*, computed from ~70,000 tweets in 13 European languages annotated for sentiment. The sentiment_score is (positive - negative) / occurrences, ranging from -1 (negative) to +1 (positive); sentiment_label is derived from its sign.

Usage

```
emoji_sentiment_lexicon
```

Format

A data frame with one row per emoji and the columns:

emoji The emoji glyph.

occurrences Number of times the emoji was observed.

position Mean position of the emoji within its text (0-1).

negative, neutral, positive Annotation counts for each class.

sentiment_score Sentiment score from -1 to 1.

sentiment_label "negative", "neutral" or "positive".

unicode_name The official Unicode character name.

unicode_block The Unicode block.

How much of the catalogue this covers

969 rows, of which 736 resolve to an emoji tidyEmoji can detect – about 19% of the 3790 distinct codepoint keys in the reference table of emoji 16.0.0 (see `emoji_provenance()` for the version you have). Two caveats on that figure, both consequences of the lexicon being built from 2015 tweets: the other 233 rows are not in the reference table at all – see *Detection limitations* below – and nothing added to Unicode after 2015 is in here. `emoji_sentiment()`'s `.emoji_n_scored` reports the shortfall per row.

Detection limitations

Many of the glyphs in this lexicon are stored in their *unqualified*, text-presentation form: a single code point with no `U+FE0F` emoji-presentation variation selector. The best-known is the bare heart, `U+2764`; others include the white smiling face (`U+263A`), the heavy check mark (`U+2714`) and the black rightwards arrow (`U+27A1`). The lexicon also contains characters that are not emoji at all (box-drawing characters, the copyright and registered signs, the replacement character), inherited from the tweets it was built from.

The grapheme-aware detection used throughout the package does not treat these text-presentation code points as emoji, so a row whose only "emoji" is one of them is not counted or scored – it behaves as if it contained no emoji. This affects detection only, never the join: supply the qualified form (the red heart `U+2764 U+FE0F`, say) and it resolves to the same lexicon entry, because every lookup goes through a codepoint key that ignores `U+FE0F`.

Source

Kralj Novak P, Smailovic J, Sluban B, Mozetic I (2015) Sentiment of Emojis. PLoS ONE 10(12): e0144296. doi:10.1371/journal.pone.0144296. Data from `https://hdl.handle.net/11356/1048`, released under the Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) licence. Processed by `data-raw/emoji_sentiment_lexicon.R`.

`emoji_summary` *Summarise emoji presence in a text column*

Description

`emoji_summary()` reports how many entries in a text column contain at least one emoji, alongside the total number of entries. An entry is counted once regardless of how many emoji it holds.

Usage

```
emoji_summary(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

A one-row tibble with columns `n_with_emoji` (entries containing at least one emoji) and `n_total` (all entries).

See Also

`emoji_filter()` to keep the emoji-bearing rows themselves.

Examples

```
df <- data.frame(text = c("I love R \U0001f600",
                          "no emoji here",
                          "flags \U0001f3c1\U0001f600"))
emoji_summary(df, text)
```

`emoji_tokens` *Tidy emoji tokens, one row per occurrence with metadata*

Description

`emoji_tokens()` expands `data` to one row per emoji occurrence (in reading order), keeping the original columns and adding the glyph together with its name, category and sentiment score. This mirrors the one-token-per-row shape familiar from tidy text mining and is convenient for counting, joining and plotting.

Usage

```
emoji_tokens(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Value

A tibble with the original columns plus `.emoji`, `.emoji_name`, `.emoji_category` and `.emoji_sentiment`. Rows without emoji are dropped. Your columns are kept, so a grouped input stays grouped, as it does through `emoji_extract_nest()`.

See Also

`emoji_frequency()` for corpus-level counts and `emoji_sentiment()` for per-row sentiment.

Examples

```
df <- data.frame(id = 1:2, text = c("great \U0001f600", "bad \U0001f621"))
emoji_tokens(df, text)
```

`emoji_token_cost` *What are the emoji in this text costing a tokeniser?*

Description

`emoji_token_cost()` measures the size of the emoji in each row: bytes, code points, grapheme clusters, and an estimate of the tokens they will cost a byte-level tokeniser. Emoji are several times more expensive than their visual weight suggests – a single ZWJ family emoji can run to well over a dozen tokens – which makes them a real line item in a prompt budget.

Usage

```
emoji_token_cost(data, text, tokenizer = NULL)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
`tokenizer`	Optional function taking a character vector and returning either token counts (a numeric vector of the same length) or a list of token vectors. It is called on the row's emoji, concatenated. `NULL` (default) uses the byte heuristic. What it returns is checked, because a wrong answer here is silent otherwise. A count is rounded up, and it has to be finite, not negative and within integer range; `NA` is accepted and passed through, for a tokeniser that cannot answer for a row. A data frame is refused rather than read, since `lengths()` on one counts its columns rather than its tokens.

Details

Bytes, code points and graphemes are exact and tidyEmoji can be authoritative about them. The token count cannot be: it depends on the tokeniser. Without `tokenizer`, `.emoji_token_estimate` is a deliberately crude heuristic of roughly two UTF-8 bytes per token, which is in the right range for byte-level BPE vocabularies but is an estimate and should never be quoted as a bill. Pass your real tokeniser through `tokenizer` when the number matters.

`.emoji_graphemes` is the number of emoji occurrences, since the package's detection is grapheme-aware: a skin-toned family emoji is one grapheme and many code points, which is precisely the gap that makes emoji expensive.

Value

`data`, as a tibble, with added columns `.emoji_n`, `.emoji_bytes`, `.emoji_codepoints`, `.emoji_graphemes` and `.emoji_token_estimate`.

See Also

`emoji_sanitize()` for acting on the answer; `emoji_ratio()` for the share of the text that is emoji.

Examples

```
family <- paste0("\U0001F468\u200d\U0001F469\u200d",
                 "\U0001F467\u200d\U0001F466")
df <- data.frame(text = c("hi \U0001f600", family, "plain"))
emoji_token_cost(df, text)
```

`emoji_to_text` *Replace emoji in a text column with words (demojize)*

Description

`emoji_to_text()` returns a copy of `data` with its text column rewritten so that every emoji is replaced by its name or shortcode. This is useful for accessibility (screen readers) and as an NLP normalisation step before tokenising. Detection is grapheme-aware and joins go through `emoji_key()`, so emoji carrying the `U+FE0F` variation selector still resolve.

Usage

```
emoji_to_text(data, text, format = c("name", "shortcode"), wrap = ":{x}:")
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

format — Output form: "name" (the Unicode name, e.g. "grinning face") or "shortcode" (the canonical GitHub-style alias, e.g. "grinning", wrapped as ":grinning:"). Default "name".

wrap — When format = "shortcode", the wrapper applied to each shortcode, written as a template with {x} standing for the shortcode. Default ":{x}:". Must contain {x}, or every emoji would be replaced by the same literal string. Ignored for format = "name".

Only the default is reversible by text_to_emoji(), which looks for exactly :shortcode:; a wrap that does not produce that token leaves the shortcode in the text as an ordinary word, silently. See emoji_sanitize() for the cases.

Value

data, as a tibble, with the text column rewritten in place (same column name). NA entries stay NA, and emoji with no known name are left in place unchanged.

See Also

text_to_emoji() for the inverse (emojize); as_emoji_name(), as_emoji_shortcode(), as_emoji() for vector helpers.

Examples

```
df <- data.frame(text = "great \U0001f600 love \u2764\ufe0f")
emoji_to_text(df, text, format = "name")
emoji_to_text(df, text, format = "shortcode")
```

---

emoji_trend *Emoji frequency over time*

Description

emoji_trend() counts emoji per time period and returns the long table that plots directly: one row per (period, emoji) over the periods it returns, including the ones in which a given emoji is absent, so a trend line does not silently skip its zeros.

Usage

```
emoji_trend(
  data,
  text,
  time,
  by = "month",
  top_n = 20,
  measure = c("n", "share")
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
time	Unquoted column of dates or date-times (`Date`, `POSIXct`, or character in `"YYYY-MM-DD"` form). A date-time is bucketed by the calendar day it *displays* as in its own timezone, not by its UTC day: an emoji posted at 23:30 New York time belongs to that day, not to the next one. "Its own timezone" means the column's `tzone` attribute. A `POSIXct` created without one – which is what `as.POSIXct("2024-01-01 23:30")` and most CSV readers give you – has no timezone of its own, so R displays it in the session's, and the buckets follow. The same column then gives hour 23 on one machine and hour 4 on another. Tag the column (`as.POSIXct(x, tz = "UTC")`, or `lubridate::force_tz()`) if the result has to be reproducible; a `Date` column is immune either way. A character column must lead with a four-digit year: `"2024-01-01"` or `"2024/01/01"`, with one- or two-digit month and day, and any trailing time ignored. Values that do not parse warn and are dropped – but a column in which *nothing* reads as a date is an error rather than a column of `NA`, since there would be no time axis left. Note that `"01/02/2024"` is in the second group: convert a column written that way with `as.Date()` and its own `format` first.
by	Period length: `"day"`, `"week"` (starting Monday), `"month"` (default), `"quarter"` or `"year"`.
top_n	Number of emoji to follow, ranked by `measure` over the whole corpus. `NULL` keeps every emoji. Default `20`. When a tie straddles the cut the glyph decides which emoji fall inside it, in the C locale, as in `top_n_emojis()`; a corpus with fewer emoji than this returns every one of them rather than padding, and `0` returns no rows at all.
measure	Statistic used to rank emoji for `top_n` and to order the rows within a period: `"n"` (default) or `"share"`.

Details

Which periods appear. The grid is complete over the *observed* periods, and "observed" means a period holding at least one emoji. A period whose rows carry no emoji at all does not appear, and neither does a gap in the calendar: `emoji_trend()` never invents a period. So the zeros it fills

in are the ones *within* the periods it returns, not a continuous time axis. Pass the result through `tidyr::complete()` against a calendar sequence if you need the empty periods too.

The three time verbs answer this differently, on purpose, and it is worth knowing which you are getting before joining two of them on `.period`:

- `emoji_trend()` – periods containing at least one emoji.
- `emoji_turnover()` – every period containing at least one dated row, including emoji-free ones, which report `n_types = 0`.
- `emoji_seasonality()` – every level of the cycle unconditionally, whether or not the data reaches it. `share` is the emoji's count divided by all emoji tokens in the same period, which is what makes periods with different volumes comparable. `top_n` selects the emoji to follow, ranked over the whole corpus by `measure`, and the selected set is the same in every period.

Rows whose time is missing or unparseable contribute nothing. Glyphs are canonicalised through the package's codepoint key, so qualified and unqualified forms share one series.

Value

A tibble with columns `.period` (a `Date`, the start of the period), `emoji`, `name`, `n` and `share`, sorted by `.period`, then by `measure` descending, then by the glyph, so the order is fully determined. That last key matters: within a period the zeros this verb fills in all tie on both of the others.

See Also

`emoji_turnover()` for vocabulary churn, `emoji_seasonality()` for cyclical patterns.

Examples

```
df <- data.frame(
  when = as.Date(c("2024-01-05", "2024-01-20", "2024-02-03")),
  text = c("\U0001f600 hi", "\U0001f600\U0001f602", "\U0001f602 yes")
)
emoji_trend(df, text, when)
```

`emoji_turnover` *Emoji vocabulary churn between consecutive periods*

Description

`emoji_turnover()` compares the set of distinct emoji used in each period with the set used in the one before: how much of the vocabulary is shared, how much is new, how much was dropped.

Usage

```
emoji_turnover(
  data,
  text,
  time,
  by = "month",
  measure = c("jaccard", "new", "lost", "core")
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.
time	Unquoted column of dates or date-times (`Date`, `POSIXct`, or character in `"YYYY-MM-DD"` form). A date-time is bucketed by the calendar day it *displays* as in its own timezone, not by its UTC day: an emoji posted at 23:30 New York time belongs to that day, not to the next one. "Its own timezone" means the column's `tzone` attribute. A `POSIXct` created without one – which is what `as.POSIXct("2024-01-01 23:30")` and most CSV readers give you – has no timezone of its own, so R displays it in the session's, and the buckets follow. The same column then gives hour 23 on one machine and hour 4 on another. Tag the column (`as.POSIXct(x, tz = "UTC")`, or `lubridate::force_tz()`) if the result has to be reproducible; a `Date` column is immune either way. A character column must lead with a four-digit year: `"2024-01-01"` or `"2024/01/01"`, with one- or two-digit month and day, and any trailing time ignored. Values that do not parse warn and are dropped – but a column in which *nothing* reads as a date is an error rather than a column of `NA`, since there would be no time axis left. Note that `"01/02/2024"` is in the second group: convert a column written that way with `as.Date()` and its own `format` first.
by	Period length: `"day"`, `"week"` (starting Monday), `"month"` (default), `"quarter"` or `"year"`.
measure	Which statistics to return: any of `"jaccard"`, `"new"`, `"lost"` and `"core"`. All four by default. Abbreviations work (`"jac"`), duplicates are ignored, and the columns come back in the order above whatever order you ask in. A value that matches none of the four is an error rather than being dropped.

Details

A period's vocabulary is its set of distinct canonicalised glyphs, so an emoji used a thousand times and one used once count the same – turnover is about repertoire, not volume. `jaccard` is the size of the intersection over the size of the union, and is `NA` when both periods are empty.

Rows whose time is missing or unparseable contribute nothing, as in `emoji_trend()`, so a period appears here only if at least one *dated* row falls in it.

Value

A tibble with one row per consecutive pair of periods: `.period`, `.period_prev`, `n_types_prev`, `n_types`, and then the requested `jaccard`, `n_new`, `n_lost` and `n_core` columns. Fewer than two periods yields no rows.

See Also

`emoji_trend()`, `emoji_version_profile()`.

Examples

```
df <- data.frame(
  when = as.Date(c("2024-01-05", "2024-02-03", "2024-02-20")),
  text = c("\U0001f600\U0001f602", "\U0001f600", "\U0001f389")
)
emoji_turnover(df, text, when)
```

---

emoji_type *Which functional types of emoji does each row use?*

Description

`emoji_type()` adds `.emoji_type`, the distinct functional types present in each row (see `as_emoji_type()`), separated by `|` when a row spans more than one. The face-versus-object contrast it exposes is the key variable in the consumer-behaviour literature on emoji in reviews and marketing copy.

Usage

```
emoji_type(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.

`text` The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

`.emoji_type` is `NA` for two different reasons, and this column cannot tell you which: a row with no emoji at all, and a row whose every emoji is one the recode cannot type. The second is rare but not impossible – the recode maps the ten Unicode groups the catalogue currently uses, so a glyph in a group added to Unicode after your **emoji** package was built has no type – and it is the same conflation `emoji_categorize()` describes for `.emoji_category`. `emoji_faceness()` separates them: `.emoji_n_typed` is `NA` when the row had no emoji and `0` when it had emoji that could not be typed. `emoji_provenance()` reports which catalogue you are matching against.

Value

`data`, as a tibble, with an added `.emoji_type` column. Unlike `emoji_categorize()`, no rows are dropped: a row with no emoji gets `NA`, as does a row whose emoji cannot be typed – see Details.

See Also

`as_emoji_type()`, `emoji_faceness()`, `emoji_categorize()`.

Examples

```
df <- data.frame(text = c("yum \U0001f355 \U0001f600", "\U0001f44d", "none"))
emoji_type(df, text)
```

---

emoji_unicode_crosswalk

Emoji name, unicode and category crosswalk

Description

A table with one row per (name, glyph) *pair*, not one row per name and not one row per glyph. 5761 rows cover 4698 distinct names and 4853 distinct glyphs, because the mapping is many-to-many in both directions:

Usage

```
emoji_unicode_crosswalk
```

Format

A data frame with four columns:

emoji_name The emoji name / shortcode (e.g. "grinning").

unicode The emoji glyph.

emoji_category The Unicode category the emoji belongs to.

key Codepoint-normalised key (U+FE0F stripped) for robust joining.

Details

- a glyph appears once for every GitHub-style name it is known by (the grinning face is both "grinning" and "grinning_face"), and
- a name appears once for every *spelling* of the emoji it names – 973 do, because the qualified and unqualified forms of an emoji are separate rows that share one alias (`A_button_blood_type_` names both `U+1F170 U+FE0F` and the bare `U+1F170`).

So a join by `emoji_name` duplicates rows for those 973 names. Join on `key` – which collapses the spellings – or `dplyr::distinct()` the columns you need first.

Source

Derived from the `emojis` table of the **emoji** package; rebuilt by `data-raw/crosswalks.R`.

`emoji_unicode_releases`

Unicode and Unicode Emoji release dates

Description

`emoji_unicode_releases()` returns the publication date of each Unicode Emoji (UTS #51) data-file release, plus the earlier Unicode versions that introduced emoji before the emoji series was numbered separately. It is the lookup that turns the `version` carried by the emoji reference table into a date, and hence into a time axis.

Usage

```
emoji_unicode_releases()
```

Details

Two numbering series exist and both turn up in emoji reference data. The Unicode Emoji series (`series = "emoji"`) runs 1.0, 2.0, ... 5.0 and then jumps to 11.0 to line up with the Unicode version; the Unicode series (`series = "unicode"`) covers the 6.0-10.0 releases that added emoji before the alignment. The two do not collide, so `version` is a unique key.

The table is kept in code rather than as a bundled `.rda`: it is a few dozen rows, it changes only when Unicode ships, and keeping it beside the verbs that use it means it can never drift out of sync with them.

Value

A tibble with columns `version` (character, the normalised label with any leading `E` removed), `version_num` (the same parsed as a number, for ordering), `series` (`"emoji"` or `"unicode"`) and `release_date` (a `Date`).

See Also

`emoji_version_profile()` and `emoji_adoption_lag()`, which join to this table; `emoji_unicode_version()` for the version this build reflects.

Examples

```
emoji_unicode_releases()
```

---

emoji_unicode_version *Which Unicode emoji version does this build reflect?*

Description

`emoji_unicode_version()` reports the highest emoji version present in the reference table tidyEmoji detects against, i.e. how current your installed **emoji** package is. Anything newer than this simply will not be recognised as an emoji.

Usage

```
emoji_unicode_version()
```

Value

A single string, `"16.0"` with **emoji** 16.0.0, or `NA` if the reference table carries no usable version information. It reports the catalogue you have installed rather than anything about tidyEmoji, so it moves when you upgrade that package and not when you upgrade this one.

See Also

`emoji_provenance()` for the full provenance row; `emoji_unicode_releases()` for release dates.

Examples

```
emoji_unicode_version()
```

`emoji_version_profile` *How new is this corpus's emoji repertoire?*

Description

`emoji_version_profile()` breaks a corpus down by the Unicode emoji version that introduced each glyph. A corpus written entirely in emoji from 2015 and one full of 2023 additions look identical to a frequency table and quite different here.

Usage

```
emoji_version_profile(data, text)
```

Arguments

`data`	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as `dplyr::mutate()` and `dplyr::filter()` do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
`text`	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a `factor` works and a numeric, `Date` or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry `U+FE0F`.

Details

The version comes from the reference table tidyEmoji detects against, so it is capped by your installed **emoji** package (see `emoji_unicode_version()`). Glyphs whose version is unknown – including any the reference table does not carry – are reported in a row with `version = NA` rather than dropped. That row is rare in practice: the upstream table records the introducing version on only one spelling of a variation pair, and tidyEmoji fills it across every spelling sharing a codepoint key, so a fully-qualified glyph such as `U+2764 U+FE0F` reports the same version as its unqualified form.

The corpus's average vintage is a weighted mean over this table, for example `with(profile, weighted.mean(version_num, n_tokens, na.rm = TRUE))`.

Value

A tibble with one row per version, oldest first: `version`, `version_num`, `release_date`, `n_types` (distinct emoji), `n_tokens` (occurrences), `share_types` and `share_tokens`. Emoji whose version the reference table does not give are pooled into one last row with `version = NA`, after every known version rather than before them.

See Also

emoji_adoption_lag() for how quickly new emoji were picked up; emoji_unicode_releases() for the date lookup.

Examples

```
df <- data.frame(text = c("\U0001f600 hello", "\U0001f97a nice"))
emoji_version_profile(df, text)
```

---

register_emoji_lexicon

Register a custom emoji lexicon

Description

register_emoji_lexicon() adds a user-supplied lexicon to the in-session registry so it can be referenced by name in emoji_score(), emoji_sentiment() or emoji_emotion(). The lexicon is normalised through the package's codepoint key (U+FE0F stripped), so a lexicon keyed on unqualified glyphs still matches qualified text.

Usage

```
register_emoji_lexicon(name, tbl, by = "emoji")
```

Arguments

name	Name to register the lexicon under.
tbl	A data frame. Must contain a glyph column named by (default "emoji") and at least one score column, and every score column present must be numeric or logical – a text column is rejected here rather than returning NA for every score at first use. See emoji_score() for the one-row-per-emoji requirement, which is checked when the lexicon is used.
by	Name of the column holding the emoji glyph, as a single string. Default "emoji".

Details

Registration lasts for the session; it is not written to disk. Registering a name that is already taken replaces the table under it, without warning and without a way to get the old one back: there is no public counterpart that removes a registration, so re-registering is how a lexicon is changed. emoji_lexicons() shows what is currently registered.

Value

Invisibly, the registered lexicon, with a key column holding the code-point key of each glyph. A key column already in tbl is replaced rather than trusted: the registry looks a lexicon up by that column, so one holding anything else would resolve every row to nothing.

See Also

emoji_lexicons() to list lexicons; emoji_score() to use one.

Examples

```
my_lex <- data.frame(
  emoji = c("\U0001f600", "\U0001f621"),
  score = c(0.9, -0.8)
)
register_emoji_lexicon("mine", my_lex)
emoji_lexicons()
emoji_score(data.frame(text = "great \U0001f600"), text, lexicon = "mine")
```

---

text_to_emoji *Replace shortcodes with emoji (emojize)*

Description

text_to_emoji() returns a copy of data with its text column rewritten so that every :shortcode: token is replaced by the corresponding emoji glyph (the inverse of emoji_to_text() with format = "shortcode" *and its default* wrap, up to the presentation selector – see Details). Shortcodes that do not match a known emoji are left unchanged.

Usage

```
text_to_emoji(data, text)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.

Details

A shortcode token is a colon, one or more of A-Z, a-z, 0-9, _, + or -, and a closing colon. Restricting the token this way means colons used for other purposes – clock times, URLs, ratios, ordinary punctuation – cannot swallow a following shortcode: "meet at 10:30 :wave:" still emojizes the wave.

The round trip recovers the emoji, not necessarily the same bytes. Like the vector helpers, both directions resolve through emoji_key(), which ignores U+FE0F, so an unqualified glyph and its fully-qualified form share one shortcode and only one of the two spellings can come back. Feeding the whole emoji catalogue through emoji_to_text(format = "shortcode") and back returns an identical code-point key for all 5042 entries and identical bytes for 79% of them. The other 1040 differ **by** U+FE0F **alone, never by more**: they come back as the spelling this verb's shortcode table carries. A second round trip changes nothing, so the result is stable either way – but compare with emoji_key(), never with string equality.

Which 79% is *not* the same question as which were already fully qualified, and the two sets genuinely differ in both directions. The bare heart (U+2764) survives unchanged, because that unqualified spelling is the one :heart: maps to; the already-qualified man detective (U+1F575 U+FE0F U+200D U+2642) does not, because it comes back with a second selector on the gender sign. If your text holds the *canonical* spelling of each emoji – what a keyboard emits – the round trip is byte-exact for all 3790 of them; see emoji_sanitize(), which tabulates both denominators.

Value

data, as a tibble, with the text column rewritten in place. NA entries stay NA.

See Also

emoji_to_text(); as_emoji() for the vector helper, which resolves a bare string by Unicode name first and so differs from this verb on 17 strings that name one emoji and alias another.

Examples

```
df <- data.frame(text = "hi :grinning: bye :waving_hand:")
text_to_emoji(df, text)

# colons elsewhere in the text do not interfere
text_to_emoji(data.frame(text = "https://example.org at 10:30 :grinning:"),
              text)
```

---

top_n_emojis *The most frequent emoji in a text column*

Description

top_n_emojis() returns the n most frequent emoji. By default each emoji (unicode) appears on a single row; set duplicated = TRUE to list every name an emoji is known by, so glyphs that share several names occupy several rows.

Usage

```
top_n_emojis(
  data,
  text,
  n = 20,
  duplicated = FALSE,
  duplicated_unicode = lifecycle::deprecated()
)
```

Arguments

data	A data frame or tibble containing a text column. Grouped data frames are accepted. The verbs that work a row at a time (adding columns, or keeping and expanding rows) carry the grouping through to their result, as dplyr::mutate() and dplyr::filter() do. The verbs that pool across rows – the counts, the co-occurrence edge lists, the time series – warn that they ignore the grouping and return one corpus-wide answer.
text	The text column to scan, supplied unquoted. Any atomic column is accepted and read as character, so a factor works and a numeric, Date or logical one simply contains no emoji. A list column – or a data-frame column – is refused rather than coerced, because coercing one deparses it and the emoji found would be in the code rather than in your data. What counts as an emoji is the same in every verb; see the *Detection* section of tidyEmoji for the one case that surprises people, code points that are emoji only when they carry U+FE0F.
n	Number of emoji to return. Default 20.
duplicated	If TRUE, emoji with several names occupy several rows. Default FALSE.
duplicated_unicode	**[Deprecated]** Use duplicated instead. Accepts only the values it ever meant – TRUE, FALSE, "yes" or "no" – and errors on anything else rather than reading it as FALSE, which is what "TRUE" and 1 used to get.

Value

A tibble with columns emoji_name, unicode, emoji_category and n, sorted by descending n with ties broken by the glyph so the order is deterministic – the same rule emoji_frequency() uses. When a tie straddles position n the glyph order decides which side of the cut each emoji falls on, and a corpus with fewer than n distinct emoji returns every one of them rather than padding to n.

duplicated = TRUE leaves several rows sharing both n and unicode, which that rule does not settle. Within one glyph the rows come out in emoji_unicode_crosswalk's order for the glyph's codepoint key, and the first of them carries the same emoji_name that duplicated = FALSE reports. Because the join is on the key rather than on the spelling, a glyph collects the aliases of *every* spelling of itself: the unqualified U+26F9 U+200D U+2640 is listed under woman_bouncing_ball and under the two aliases the fully-qualified spelling carries. That is what "every name an emoji is known by" means here.

See Also

emoji_frequency() for the full distribution.

Examples

```
df <- data.frame(text = c("\U0001f600\U0001f600\U0001f3c1", "\U0001f621"))
top_n_emojis(df, text, n = 2)
```

Index

∗ **datasets**
  category_unicode_crosswalk, 6
  emoji_emotion_lexicon, 23
  emoji_sentiment_lexicon, 51
  emoji_unicode_crosswalk, 62

as_emoji *(*as_emoji_name*)*, 3
as_emoji(), *47*, *57*, *68*
as_emoji_name, 3
as_emoji_name(), *47*, *57*
as_emoji_shortcode *(*as_emoji_name*)*, 3
as_emoji_shortcode(), *47*, *57*
as_emoji_type, 5
as_emoji_type(), *26*, *61*, *62*

category_unicode_crosswalk, 6, *11*

dplyr::filter(), *6*, *10*, *12*, *13*, *15–17*, *19*, *20*, *22*, *24–30*, *33*, *35*, *36*, *38*, *40–42*, *45*, *48*, *50*, *53–56*, *58*, *60*, *61*, *65*, *67*, *69*
dplyr::mutate(), *6*, *10*, *12*, *13*, *15–17*, *19*, *20*, *22*, *24–30*, *33*, *35*, *36*, *38*, *40–42*, *45*, *48*, *50*, *53–56*, *58*, *60*, *61*, *65*, *67*, *69*

emoji_adoption_lag, 6
emoji_adoption_lag(), *64*, *66*
emoji_ambiguity, 8
emoji_ambiguity(), *28*, *41*, *42*, *51*
emoji_categorize, 10
emoji_categorize(), *5*, *62*
emoji_collocations, 11
emoji_collocations(), *16*
emoji_congruence, 13
emoji_congruence(), *32*
emoji_context, 14
emoji_context(), *12*
emoji_cooccurrence, 16
emoji_cooccurrence(), *37*
emoji_density, 17
emoji_density(), *15*, *39*, *41*
emoji_dfm, 18
emoji_emotion, 20
emoji_emotion(), *21–23*, *45*, *46*, *51*, *66*
emoji_emotion_label, 21
emoji_emotion_label(), *21*
emoji_emotion_lexicon, *21*, *22*, 23
emoji_extract_nest, 24
emoji_extract_nest(), *25*, *54*
emoji_extract_unnest, 25
emoji_extract_unnest(), *19*, *24*, *27*, *36*
emoji_faceness, 26
emoji_faceness(), *5*, *11*, *46*, *62*
emoji_filter, 27
emoji_filter(), *41*, *53*
emoji_flag_ambiguous, 28
emoji_flag_ambiguous(), *9*, *42*
emoji_frequency, 29
emoji_frequency(), *11*, *19*, *25*, *54*, *69*
emoji_incongruity, 30
emoji_incongruity(), *13*, *14*, *32*, *34*, *43*, *46*
emoji_incongruity_profile, 32
emoji_incongruity_profile(), *32*
emoji_lexicons, 34
emoji_lexicons(), *39*, *46*, *66*, *67*
emoji_ngrams, 35
emoji_ngrams(), *15*, *17*, *37*
emoji_pairs, 36
emoji_pairs(), *16*, *17*, *36*
emoji_position, 37
emoji_position(), *16*, *18*, *41*
emoji_provenance, 39
emoji_provenance(), *11*, *23*, *34*, *52*, *62*, *64*
emoji_ratio, 40
emoji_ratio(), *15*, *18*, *27*, *38*, *39*, *56*
emoji_risk, 41
emoji_risk(), *9*, *28*, *46*
emoji_sanitize, 42

emoji_sanitize(), *56, 57*, *68*
emoji_score, 45
emoji_score(), *35*, *50, 51*, *66, 67*
emoji_search, 47
emoji_search(), *4*
emoji_seasonality, 48
emoji_seasonality(), *59*
emoji_sentiment, 50
emoji_sentiment(), *9*, *21, 22*, *26*, *31, 32*, *41*, *45, 46*, *52*, *54*, *66*
emoji_sentiment_lexicon, *8, 9*, *39*, *50, 51*, 51
emoji_summary, 53
emoji_summary(), *27*, *49*
emoji_to_text, 56
emoji_to_text(), *4*, *42, 43*, *45*, *67, 68*
emoji_token_cost, 55
emoji_token_cost(), *45*
emoji_tokens, 54
emoji_tokens(), *19*, *24, 25*, *27*
emoji_trend, 57
emoji_trend(), *49*, *59*, *61*
emoji_turnover, 59
emoji_turnover(), *59*
emoji_tweets *(*emoji_filter*)*, 27
emoji_type, 61
emoji_type(), *5*, *11*, *26*
emoji_unicode_crosswalk, 62, *69*
emoji_unicode_releases, 63
emoji_unicode_releases(), *7*, *39*, *64*, *66*
emoji_unicode_version, 64
emoji_unicode_version(), *39*, *64, 65*
emoji_version_profile, 65
emoji_version_profile(), *7*, *61*, *64*

nchar(), *15*, *18*

register_emoji_lexicon, 66
register_emoji_lexicon(), *20*, *35*, *46*, *50*

substr(), *15*, *38*

text_to_emoji, 67
text_to_emoji(), *4*, *44*, *47*, *57*
tidyEmoji, *7*, *10–13*, *15–20*, *22*, *24–30*, *33–36*, *38*, *40, 41*, *43*, *45*, *48*, *50*, *53–56*, *58*, *60*, *62*, *65*, *67*, *69*
tidyr::complete(), *59*
top_n_emojis, 68
top_n_emojis(), *28, 29*, *58*