Package ‘qdaR’

September 5, 2026

Title Read and Analyse Qualitative Coding Exported from Zotero

Version 0.1.0

Description Reads the versioned exchange files written by the Zotero plugins 'zotQDA' and 'qdaZ' -- coded fragments, code systems, coding histories and team-consensus results -- validates them against the shipped contract, and reproduces the plugin's graphics with 'ggplot2'. Adds what those plugins deliberately leave out: six agreement coefficients with bootstrap confidence intervals, the reliability of the segmentation itself, chi-squared tests of code by group tables with effect sizes, correspondence analysis, multidimensional scaling and hierarchical clustering of codes. Projects from other programs can be read through the 'REFI-QDA' interchange standard <https://www.qdasoftware.org/>, which makes those analyses available to users of established software that does not offer them; the subset a '.qdpx' supports is reported on import. Reference files are included, so every function can be tried without a Zotero installation.

License AGPL-3

Encoding UTF-8

Depends R (>= 4.1)

Imports ggplot2, jsonlite, stats, utils, MASS

Suggests testthat (>= 3.0.0), knitr, rmarkdown, vegawidget, withr, xml2

Config/testthat/edition 3

VignetteBuilder knitr

URL https://zotqda.org, https://qdar.zotqda.org/, https://github.com/fre-ms/qdaR

BugReports https://github.com/fre-ms/qdaR/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author fre.ms [aut, cre]

Maintainer fre.ms <fre.ms@fre.ms>

Repository CRAN

Date/Publication 2026-09-05 13:50:08 UTC

Contents

`qda_ac1` *Gwet's AC1*

Description

Chance-corrected agreement that stays stable when one category dominates, the situation in which kappa collapses although the coders plainly agree. Worth reporting beside kappa rather than instead of it: where the two diverge, the marginals are the story.

Usage

```
qda_ac1(units)
```

Arguments

`units`	A unit-by-coder matrix from [qda_units()].

Details

Gwet (2008) doi:10.1348/000711006X126600.

Value

A number, or 'NaN' when AC1 is undefined.

Examples

```
u <- cbind(ann = c("A", "A", "A", "B"), bob = c("A", "A", "A", "A"))
qda_ac1(u)
```

`qda_agreement` *All agreement measures at once*

Description

Reports the measures side by side, because no single coefficient settles the question: they disagree exactly where the marginals are skewed, and seeing them disagree is the finding.

Usage

```
qda_agreement(units)
```

Arguments

`units`	A unit-by-coder matrix from [qda_units()].

Value

A one-row data frame with the number of comparable units, the number of categories, and the measures. Cohen's kappa is 'NA' for more than two coders.

Examples

```
u <- cbind(ann = c("A", "B", "A", "B"), bob = c("A", "B", "B", "B"))
qda_agreement(u)
```

`qda_agreement_by_code` *Agreement per code*

Description

A single pooled coefficient hides which codes the coders actually argued about. This asks the question once per code, as a yes/no judgement, which is also the only honest way to treat material where segments legitimately carry several codes.

Usage

```
qda_agreement_by_code(
  fragments,
  min_n = 3,
  unit = "annotationKey",
  coder = "codedBy",
  value = "code"
)
```

Arguments

`fragments`	A fragments data frame from [qda_read_fragments()].
`min_n`	Skip codes used fewer than this many times; with two or three uses every coefficient is noise.
`unit`	Column identifying the unit of analysis.
`coder`	Column identifying the coder.
`value`	Column holding the category; ‘"code"‘ is the readable path, ‘"codeId"‘ the identity that survives renaming.

Value

A data frame with one row per code: ‘n‘ uses, ‘units‘ compared, ‘percent‘, ‘cohen‘, ‘ac1‘, and ‘prevalence‘ with its Wilson interval.

Examples

```
frag <- data.frame(
  annotationKey = rep(paste0("s", 1:6), each = 2),
  codedBy = rep(c("ann", "bob"), 6),
  code = c("A", "A", "A", "B", "B", "B", "A", "A", "B", "B", "A", "A")
)
qda_agreement_by_code(frag, min_n = 1)
```

`qda_alpha` *Krippendorff's alpha*

Description

Chance-corrected agreement for any number of coders that tolerates missing values, computed from the coincidence matrix. Nominal data only here, which is what codes are.

Usage

```
qda_alpha(units)
```

Arguments

`units`	A unit-by-coder matrix from [qda_units()].

Details

Hayes and Krippendorff (2007) doi:10.1080/19312450709336664.

Value

A number, or ‘NaN‘ when alpha is undefined.

Examples

```
u <- cbind(ann = c("A", "B", "A", NA), bob = c("A", "B", "B", "A"))
qda_alpha(u)
```

`qda_apply_mapping` *Apply a consensus mapping to coded fragments*

Description

Adds a ‘consensusCode‘ column without touching ‘code‘: the original coding stays visible next to its consensus interpretation.

Usage

```
qda_apply_mapping(fragments, mapping, coder_col = "codedBy")
```

Arguments

`fragments`	A fragments data frame.
`mapping`	A consensus-mapping data frame.
`coder_col`	Column holding the coder in ‘fragments‘; defaults to ‘"codedBy"‘.

Value

‘fragments‘ with an added ‘consensusCode‘ column.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
map <- qda_read_mapping(qda_example("zotqda-konsens-abbildung.csv"))
out <- qda_apply_mapping(frag, map)
names(out)[ncol(out)]
```

`qda_bootstrap_ci` *Bootstrap confidence interval for an agreement coefficient*

Description

A coefficient without an interval invites over-reading, which is the complaint Sim and Wright (2005) doi:10.1093/ptj/85.3.257 and Zapf et al. (2016) doi:10.1186/s1287401602009 both make. Qualitative studies work with few units, so the interval is usually wide – and that is the point.

Usage

```
qda_bootstrap_ci(
  units,
  fn = qda_fleiss,
  resamples = 1000,
  seed = 42,
  level = 0.95
)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].
fn	The coefficient to bootstrap, e.g. [qda_fleiss()].
resamples	Number of bootstrap samples.
seed	Seed, so a published interval can be reproduced. The plugin and the Python twin use the same generator and the same default, so all three report the same interval for the same data.
level	Confidence level.

Details

Units are resampled, not ratings: the unit of analysis is the segment, and resampling ratings would treat two judgements of one segment as independent observations.

Value

A list with ‘estimate‘, ‘lo‘, ‘hi‘ and ‘used‘ (how many resamples produced a finite value), or ‘NULL‘ when fewer than 20 did – a wide interval is informative, an interval computed from nothing is not.

Examples

```
u <- cbind(ann = rep(c("A", "B"), 20), bob = rep(c("A", "B", "B", "A"), 10))
qda_bootstrap_ci(u, qda_kappa, resamples = 200)
```

---

qda_brennan *Brennan and Prediger's kappa*

Description

Like Cohen's, but chance is the uniform ‘1/q‘ over the categories the scheme offers rather than the coders' marginals. This is the figure MAXQDA reports, so it is the one to use when a result has to line up with a MAXQDA output.

Usage

```
qda_brennan(units, q = NULL)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].
q	Number of categories the scheme offers. Defaults to the categories present anywhere in ‘units‘ – pass the size of the code system when coders could have chosen codes they never used, because that is the number the coefficient is actually about.

Details

Brennan and Prediger (1981) doi:10.1177/001316448104100307.

Value

A number, or ‘NaN‘ when nothing is comparable.

Examples

```
u <- cbind(ann = c("A", "B", "A", "B"), bob = c("A", "B", "B", "B"))
qda_brennan(u)
```

---

qda_ca *Correspondence analysis of the code by document table*

Description

Shows which codes and which documents attract each other. Unlike the plugin's descriptive matrix, this decomposes the table and reports how much of its inertia the first dimensions explain – the honest answer to "how much of the picture am I actually seeing".

Usage

```
qda_ca(fragments, doc_col = "citekey", n_dims = 2)
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
doc_col	Column identifying the document.
n_dims	Number of dimensions to keep.

Value

A list with the ‘correspondence‘ object from [MASS::corresp()], the row and column ‘scores‘, the ‘inertia‘ of the kept dimensions, the ‘total_inertia‘ of the whole table and ‘inertia_share‘, the fraction of that total each kept dimension carries. The share is deliberately relative to the *total*: shares that are normalised to the dimensions you happened to keep always add up to 100 percent and so answer a question nobody asked.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
if (nrow(unique(frag["code"])) > 1) {
  ca <- try(qda_ca(frag), silent = TRUE)
}
```

---

qda_chisq *Association between codes and a grouping variable*

Description

The plugins report descriptive agreement and co-occurrence but no inferential tests – deliberately, because a test invites a claim the design often does not support. Where the design *does* support it, this function performs the usual chi-squared test of independence, reports Cramer's V as an effect size, and says whether the approximation was appropriate at all. When expected counts fall below five it reports an exact test instead: Fisher's for a two-by-two table, and a Monte Carlo p-value with the margins fixed for anything larger.

Usage

```
qda_chisq(
  fragments,
  group = "citekey",
  codes = NULL,
  resamples = 2000,
  seed = 42
)
```

Arguments

fragments	A fragments data frame.
group	A column of 'fragments' to test the codes against, e.g. '"citekey"', or a vector of the same length.
codes	Restrict to these codes; 'NULL' uses all.
resamples	Number of Monte Carlo replicates for the simulated p-value.
seed	Seed for those replicates, so a reported p-value can be reproduced exactly.

Details

Note the unit of this test: one coded fragment. Fragments from the same document are not independent observations, so a significant result across documents is weaker evidence than the p-value suggests.

Value

A list with the contingency ‘table‘, the ‘test‘ object, the chi-squared ‘statistic‘ and ‘expected‘ counts it was computed from, the effect size ‘cramers_v‘, and ‘expected_ok‘ telling you whether the chi-squared approximation was appropriate. The statistic is reported even when the exact test is used, because Cramer’s V is derived from it and an effect size nobody can recompute is not worth reporting.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
res <- qda_chisq(frag, group = "citekey")
res$cramers_v
```

`qda_cluster` *Hierarchical clustering of codes*

Description

Clusters codes by the segments they share. The cophenetic correlation is reported alongside, because a dendrogram always looks convincing even when it represents the distances poorly – values well below about 0.7 mean the picture should not be over-read.

Usage

```
qda_cluster(fragments, unit = "annotationKey", min_n = 3, method = "average")
```

Arguments

`fragments`	A fragments data frame.
`unit`	Column identifying the segment; defaults to ‘"annotationKey"‘.
`min_n`	Ignore codes used fewer than ‘min_n‘ times. Rarely used codes make every coefficient unstable.
`method`	Linkage passed to [stats::hclust()].

Value

A list with the ‘hclust‘ object, the ‘distance‘ and the ‘cophenetic‘ correlation. With fewer than three codes the correlation is undefined and reported as ‘NA‘ rather than as a number that means nothing.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_cluster(frag, min_n = 1)$cophenetic
```

qda_code_counts *Counts per code*

Description

Counts per code

Usage

```
qda_code_counts(fragments, top = NULL)
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
top	Show only the ‘top‘ most frequent codes; ‘NULL‘ shows all.

Value

A data frame with ‘code‘ and ‘n‘, most frequent first.

Examples

```
qda_code_counts(qda_read_fragments(qda_example("zotqda-fragments.csv")))
```

qda_code_distance *Jaccard distances between codes*

Description

Two codes are close when they are assigned to the same segments. This is the distance the multi-dimensional scaling and the clustering below work on, and the same coefficient the plugin uses to propose code matches.

Usage

```
qda_code_distance(fragments, unit = "annotationKey", min_n = 3)
```

Arguments

fragments	A fragments data frame.
unit	Column identifying the segment; defaults to ‘"annotationKey"‘.
min_n	Ignore codes used fewer than ‘min_n‘ times. Rarely used codes make every coefficient unstable.

Value

An object of class ‘dist‘.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_code_distance(frag, min_n = 1)
```

---

qda_code_drift *Did a coder's behaviour shift while the project ran?*

Description

The coding log records who coded what and when, which is unusual: most tools keep no such trail, so this question normally cannot be asked at all. What it supports is the distribution of codes a coder used in successive windows, reported as the total variation distance to that coder's first window. Nought means they are coding as they started, one that the two windows share no code.

Usage

```
qda_code_drift(history, windows = 4)
```

Arguments

history	A history data frame from [qda_read_history()].
windows	Number of equal-count windows per coder. Equal counts rather than equal time, because a coder who worked in bursts would otherwise get empty windows.

Details

It is a description, not a test. A large distance can mean the coder drifted, or simply that the later material was about something else. Read it next to what was coded, not on its own.

Value

A data frame with one row per coder and window: ‘coder‘, ‘window‘, ‘n‘, ‘codes‘, ‘from‘, ‘to‘ and ‘distance‘.

Examples

```
h <- data.frame(
  ts = sprintf("2026-01-%02dT09:00:00Z", 1:8), user = "ann",
  action = "add", code = c(rep("A", 4), rep("B", 4)), citekey = "d1"
)
qda_code_drift(h, windows = 2)
```

qda_code_matrix *Code by document counts*

Description

Code by document counts

Usage

```
qda_code_matrix(fragments, doc_col = "citekey", long = TRUE)
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
doc_col	Column identifying the document.
long	Return a long data frame (‘TRUE‘) or a matrix (‘FALSE‘).

Value

A data frame or a matrix of counts.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_code_matrix(frag, long = FALSE)
```

qda_codings *Reconstruct the current per-coder coding state from the history*

Description

The fragments export is "last state": one row per annotation and code, with a single ‘codedBy‘ – when two coders coded the same segment, the last write wins and the other coder is gone. That makes fragments the wrong table for intercoder reliability, because the very disagreement reliability measures is what it collapses.

Usage

```
qda_codings(
  history,
  unit = "annotationKey",
  coder = "user",
  value = "code",
  time = "ts",
  action = "action"
)
```

Arguments

history	A history data frame from [qda_read_history()].
unit	Column identifying the unit of analysis.
coder	Column identifying the coder.
value	Column holding the category (the code path).
time	Column holding the event timestamp, sorted oldest first.
action	Column holding ‘"add"‘ or ‘"remove"‘.

Details

The history keeps every coding *event* instead: one row per ‘add‘ or ‘remove‘, per coder. Replaying it recovers who coded what – both coders on the same segment survive, which is exactly what an agreement figure needs. This is how the plugin itself computes reliability.

For each ‘(unit, coder, value)‘ the events are applied oldest first and the pair is kept when its last event is an ‘add‘; an ‘add‘ later withdrawn by a ‘remove‘ drops out. The result is one row per surviving coder–code pairing, ready for [qda_units()] with ‘coder = "user"‘.

Value

A data frame with one row per surviving pairing, columns ‘unit‘, ‘coder‘ and ‘value‘ (by their given names), and ‘citekey‘ and ‘title‘ carried through when present.

Examples

```
hist <- qda_read_history(qda_example("zotqda-history-demo.csv"))
codings <- qda_codings(hist)
u <- qda_units(codings, coder = "user")
qda_agreement(u)$alpha
```

---

qda_confusion *Where two coders disagreed*

Description

The confusion table is what turns a disappointing kappa into something actionable: usually a handful of category pairs account for most of it, and those pairs are the ones whose definitions need work.

Usage

```
qda_confusion(units, only_disagreements = FALSE)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].
only_disagreements	Drop the diagonal.

Value

A data frame with the two coders' categories and the count, most frequent first.

Examples

```
u <- cbind(ann = c("A", "B", "A", "B"), bob = c("A", "B", "B", "B"))
qda_confusion(u)
```

`qda_contract` *The zotQDA exchange contract*

Description

zotQDA and qdaZ write versioned files described by a machine-readable contract. Every file states its kind and version in its first column, so a reader can recognise what it is holding without relying on the file name and can refuse a version it does not understand instead of quietly computing something wrong.

Usage

```
qda_contract(path = NULL)
```

Arguments

`path`	Optional path to an 'exchange-v*.json'. Defaults to the copy shipped with this package.

Value

A list with 'contract', 'version', 'csv' and 'formats'.

Examples

```
ct <- qda_contract()
ct$version
names(ct$formats)
```

qda_coreq *A COREQ checklist, pre-filled with what the data can answer*

Description

COREQ (Tong, Sainsbury & Craig 2007, doi:10.1093/intqhc/mzm042) is a submission requirement at many journals: 32 items across research team, study design, and analysis and findings. Most of them only the researcher can answer. Six of them the exports already know, and filling those in saves the tedious part while making the rest visible as gaps.

Usage

```
qda_coreq(fragments = NULL, history = NULL, codebook = NULL, software = NULL)
```

Arguments

`fragments`	A fragments data frame, or 'NULL'.
`history`	A history data frame, or 'NULL'; enables the saturation item.
`codebook`	A codebook data frame, or 'NULL'; enables the coding-tree item.
`software`	Free text for item 27. The default names the tools in use.

Details

The point is not automation. It is that the numbers a reviewer will ask for – how many documents, how many coders, how large the code system, was saturation discussed – come out of the data rather than out of memory, and therefore match what the analysis actually did.

Value

A data frame with one row per item: 'item', 'domain', 'section', 'name', 'question', 'answer' and 'filled' ('TRUE' where the data supplied it).

What COREQ does not ask

There is no item for intercoder agreement. If you computed it, it belongs in your answer to item 24 or 25; the checklist will not prompt you.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
cq <- qda_coreq(frag)
cq[cq$filled, c("item", "name", "answer")]
```

qda_coreq_markdown *The checklist as Markdown, ready to paste into a submission*

Description

The checklist as Markdown, ready to paste into a submission

Usage

```
qda_coreq_markdown(coreq, title = "COREQ checklist", file = NULL)
```

Arguments

coreq	A data frame from [qda_coreq()].
title	Heading for the document.
file	Optional path to write to.

Value

A character vector of Markdown lines.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
head(qda_coreq_markdown(qda_coreq(frag)), 8)
```

qda_documents_for *How many more documents for a given saturation?*

Description

The practical follow-up to [qda_saturation_index()]: the fitted curve is solved for the number of documents at which the index would reach ‘target‘.

Usage

```
qda_documents_for(fit, target = 95, max_n = 1000)
```

Arguments

fit	A fit from [qda_saturation_index()].
target	Desired saturation, in per cent.
max_n	Largest number of documents to consider.

Value

The number of documents, or ‘NA‘ when the model does not reach the target within ‘max_n‘ – which is itself worth reporting.

Examples

```
fit <- qda_saturation_index(c(8, 13, 16, 18, 19, 20, 20, 21))
qda_documents_for(fit, 95)
```

`qda_example` *Path to a bundled reference file*

Description

The reference files from the contract are installed with this package, so every example and test runs without a Zotero installation. They contain the awkward cases on purpose: quotes, the delimiter and a line break inside a field.

Usage

```
qda_example(file = NULL)
```

Arguments

`file`	File name, e.g. ‘"zotqda-fragments.csv"‘. Call without arguments to list what is available.

Value

A file path, or the available names when called with no argument.

Examples

```
qda_example()
qda_example("zotqda-fragments.csv")
```

qda_flatten_path *Shorten a code path to a number of levels*

Description

‘"Belastung/beruflich/akut"‘ at level 2 becomes ‘"Belastung/beruflich"‘.

Usage

```
qda_flatten_path(path, level = NULL)
```

Arguments

path	Code paths.
level	Number of levels to keep; ‘NULL‘ or ‘0‘ keeps everything.

Value

A character vector.

Examples

```
qda_flatten_path("Belastung/beruflich/akut", 2)
```

qda_fleiss *Fleiss' kappa*

Description

Chance-corrected agreement for any number of coders. Units rated by fewer than two coders carry no agreement information and are skipped.

Usage

```
qda_fleiss(units)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].

Details

Because it works on one category per unit, this is the figure that makes the case for coding a segment once: a scheme where segments routinely carry several codes has no single value to compare, and the overall figure is then computed on whatever remains unambiguous. [qda_units()] counts what it set aside, and the count is worth reporting next to the kappa.

Fleiss (1971) doi:10.1037/h0031619.

Value

A number, or ‘NaN‘ when kappa is undefined.

Examples

```
u <- cbind(ann = c("A", "B", "A"), bob = c("A", "B", "B"),
           cat = c("A", "B", "A"))
qda_fleiss(u)
```

`qda_formats` *Supported exchange formats*

Description

Supported exchange formats

Usage

```
qda_formats(path = NULL)
```

Arguments

`path`	Optional path to an ‘exchange-v*.json‘. Defaults to the copy shipped with this package.

Value

A data frame with one row per format: ‘format‘, ‘id‘, ‘file‘, ‘grain‘ and the number of columns.

Examples

```
qda_formats()
```

`qda_gamma` *Agreement measured with respect to the best alignment*

Description

Every other coefficient in this package fixes the alignment first and then measures agreement on it. Gamma (Mathet, Widlocher & Metivier 2015, doi:10.1162/coli_a_00227) refuses that separation: unitizing and categorisation are judged together, and the measure reports the alignment it found alongside the number.

Usage

```
qda_gamma(
  by_coder,
  dist_cat = NULL,
  alpha = 1,
  beta = 1,
  samples = 30,
  seed = 42,
  max_nodes = 2e+05
)
```

Arguments

by_coder	A list, one entry per annotator, of lists of units.
dist_cat	Optional category distance.
alpha, beta	Weights for position and category.
samples	Number of random continua for the expected disorder.
seed	Seed for the shifts.
max_nodes	Search budget.

Details

The practical difference from [qda_unitizing_alpha()] is that gamma can pair two units that do not overlap at all, when the surrounding configuration says they refer to the same phenomenon. Alpha cannot express that.

Chance correction is by sampling: the annotations are randomly shifted around the continuum, which preserves every unit's length and category and destroys only the alignment. The generator is the plugin's, seeded, so all three implementations report the same expected value.

Value

A list with 'gamma', the 'observed' and 'expected' disorder, the 'alignment', 'samples', and 'recommended_samples' – the number the observed variability suggests for two per cent precision (the paper's sampling rule). 'gamma' is 'NaN' with a 'reason' when the search was cut short.

Examples

```
u <- function(s, e, v) list(start = s, end = e, value = v)
same <- list(u(0, 10, "A"), u(20, 30, "B"))
qda_gamma(list(same, same), samples = 10)$gamma      # 1
```

`qda_gamma_best_alignment` *The best alignment, and the disorder of an annotation set*

Description

Gamma does not fix the alignment before measuring: it searches for the pairing of units across annotators that minimises the combined positional and categorical disorder, and reports agreement with respect to that. Finding it is a set-partitioning problem, NP-hard for three or more annotators.

Usage

```
qda_gamma_best_alignment(
  by_coder,
  dist_cat = NULL,
  alpha = 1,
  beta = 1,
  max_nodes = 2e+05
)
```

Arguments

`by_coder`	A list, one entry per annotator, of lists of units.
`dist_cat`	Optional category distance.
`alpha, beta`	Weights for position and category.
`max_nodes`	Search budget.

Details

This is an exact branch and bound using the paper's pruning theorem (equation 9) plus an admissible bound. When the search would exceed ‘max_nodes‘ it **refuses**: a gamma produced by a heuristic is not gamma, and reporting one would be worse than reporting nothing.

Value

A list with ‘disorder‘, the ‘alignment‘ as unit identifiers, the number of ‘candidates‘ after pruning, and ‘exhausted‘.

Examples

```
u <- function(s, e, v) list(start = s, end = e, value = v)
qda_gamma_best_alignment(list(list(u(0, 10, "A")), list(u(0, 10, "A"))))$disorder
```

qda_gamma_dissimilarity *Dissimilarity between two units, as gamma defines it*

Description

Equation (5) of Mathet, Widlocher and Metivier (2015) doi:10.1162/coli_a_00227, with both weights at one: position and category are added, so a unit in the right place with the wrong code costs the same as one with the right code in a badly wrong place.

Usage

```
qda_gamma_dissimilarity(u, v, dist_cat = NULL, alpha = 1, beta = 1)
```

Arguments

u, v	Units, each a list with ‘start‘, ‘end‘ and ‘value‘. ‘NULL‘ stands for the empty unit, which costs ‘Delta_empty‘ against anything.
dist_cat	Optional category distance in ‘[0, 1]‘; nominal by default.
alpha, beta	Weights for position and category.

Value

A number.

Examples

```
u <- list(start = 0, end = 10, value = "A")
qda_gamma_dissimilarity(u, list(start = 2, end = 12, value = "A"))  # 0.04
qda_gamma_dissimilarity(u, list(start = 0, end = 10, value = "B"))  # 1
```

qda_kappa *Cohen's kappa*

Description

Chance-corrected agreement for exactly two coders, on the units both rated. Chance is estimated from the coders' own marginals, which is what makes kappa fall when one category dominates – the paradox that keeps being mistaken for a defect of the coding.

Usage

```
qda_kappa(units)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].

Details

Cohen (1960) doi:10.1177/001316446002000104.

Value

A number, or ‘NaN‘ when kappa is undefined.

Examples

```
u <- cbind(ann = c("A", "B", "A", "B"), bob = c("A", "B", "B", "B"))
qda_kappa(u)
```

---

qda_kappa_lower *What lower bound can this much material reach?*

Description

The question the other way round, which is the one you face when the number of segments is already fixed by the budget: given ‘n‘ segments, how far down does the one-sided interval for kappa reach? Donner and Rotondi’s Table 3 answers it; this reproduces the calculation.

Usage

```
qda_kappa_lower(
  n,
  kappa0,
  prevalence,
  raters = 2,
  alpha = 0.05,
  critical = NULL
)
```

Arguments

n	Number of segments available.
kappa0	The kappa you anticipate.
prevalence	Share of segments carrying the code, between 0 and 1. The requirement is symmetric about 0.5, so a conservative planner takes the value further from it.
raters	Number of coders; two or more.
alpha	One minus the confidence level of the one-sided interval.
critical	The chi-squared critical value. The default is the exact quantile. The published tables were computed with it rounded to 2.71, which makes eight of their forty-eight cells one larger; pass ‘critical = 2.71‘ to reproduce them cell for cell.

Value

The expected lower bound, or ‘NA‘ when even a kappa of nought cannot be excluded with this much material.

Examples

```
qda_kappa_lower(100, kappa0 = 0.7, prevalence = 0.3, raters = 4)
```

`qda_level_agreement` *Agreement by level of the code system*

Description

A hierarchical code system can be read at several resolutions, and coders who disagree about ‘Belastung/beruflich‘ against ‘Belastung/privat‘ still agree that the segment is about ‘Belastung‘. Flattening paths level by level and recomputing shows where in the hierarchy the agreement is lost – which is a statement about the code system, not about the coders.

Usage

```
qda_level_agreement(units, max_level = NULL)
```

Arguments

`units`	A unit-by-coder matrix from [qda_units()], values being code paths.
`max_level`	Deepest level to report; defaults to the deepest path.

Value

A data frame with one row per level, holding the categories and comparable units at that level and all measures.

Examples

```
u <- cbind(ann = c("A/x", "A/y", "B/x"), bob = c("A/y", "A/y", "B/x"))
qda_level_agreement(u)
```

qda_mds *Multidimensional scaling of codes*

Description

Places codes in two dimensions so that codes applied to the same segments end up close together. A map of this kind says nothing about significance; it is a way of looking at a distance matrix.

Usage

```
qda_mds(fragments, unit = "annotationKey", min_n = 3)
```

Arguments

fragments	A fragments data frame.
unit	Column identifying the segment; defaults to ‘"annotationKey"‘.
min_n	Ignore codes used fewer than ‘min_n‘ times. Rarely used codes make every coefficient unstable.

Value

A list with the ‘points‘ data frame and the ‘goodness‘ of fit reported by [stats::cmdscale()].

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_mds(frag, min_n = 1)$points
```

qda_new_codes *New codes per document, in coding order*

Description

The input the saturation measures work on: how many codes appeared for the first time in each document, with documents ordered by when they were first coded.

Usage

```
qda_new_codes(history, doc_col = "citekey")
```

Arguments

history	A history data frame from [qda_read_history()].
doc_col	Column identifying the document.

Value

A data frame with ‘position‘, ‘document‘, ‘new_codes‘ and ‘cumulative‘.

Examples

```
h <- data.frame(
  ts = sprintf("2026-01-%02dT09:00:00Z", 1:6),
  user = "ann", action = "add",
  code = c("A", "B", "A", "C", "A", "B"),
  citekey = c("d1", "d1", "d2", "d2", "d3", "d3")
)
qda_new_codes(h)
```

---

qda_paradox *Why a kappa is disappointing*

Description

Feinstein and Cicchetti (1990) doi:10.1016/08954356(90)90158L named the two reasons a kappa can collapse while observed agreement is high: a skewed marginal distribution, and a systematic difference between the coders. These indices measure exactly those two, and PABAK is kappa recomputed with chance fixed at one half, which removes the prevalence effect.

Usage

```
qda_paradox(units)
```

Arguments

units	A unit-by-coder matrix with exactly two coders and two categories; anything else returns ‘NULL‘ rather than a number that does not mean what it appears to.

Details

Reporting kappa alone tells a reader that agreement is poor. These three numbers tell them why, which is the difference between a result and something they can act on.

Value

A list with ‘prevalence_index‘, ‘bias_index‘, ‘pabak‘, ‘percent‘, the two ‘categories‘, ‘n‘ and the two-by-two ‘table‘; or ‘NULL‘.

Examples

```
u <- cbind(ann = c("A", "A", "A", "B"), bob = c("A", "A", "A", "A"))
qda_kappa(u)             # 0, which looks like failure
qda_paradox(u)$pabak     # 0.5, and the prevalence index says why
```

qda_percent_agreement *Observed agreement between coders*

Description

The share of agreeing coder pairs, over all units and all pairs where both coders rated. Easy to read and, on its own, easy to over-read: with one dominant category a high value says little.

Usage

```
qda_percent_agreement(units)
```

Arguments

units	A unit-by-coder matrix from [qda_units()].

Value

A number between 0 and 1, or ‘NaN‘ when nothing is comparable.

Examples

```
u <- cbind(ann = c("A", "B", "A"), bob = c("A", "B", "B"))
qda_percent_agreement(u)
```

qda_plan_kappa *How many segments must be double-coded?*

Description

Interobserver studies are routinely run at whatever size was convenient and then reported with a confidence interval far too wide to support the claim made from it. Donner and Rotondi (2010) doi:10.2202/15574679.1275 give the sample size that makes the *lower* bound of a one-sided interval for kappa reach a value you name in advance – which is the quantity a reader actually cares about, since nobody argues that agreement was too good.

Usage

```
qda_plan_kappa(
  kappa0,
  kappa_lower,
  prevalence,
  raters = 2,
  alpha = 0.05,
  critical = NULL
)
```

Arguments

kappa0	The kappa you anticipate.
kappa_lower	The minimum you want the interval's lower bound to reach.
prevalence	Share of segments carrying the code, between 0 and 1. The requirement is symmetric about 0.5, so a conservative planner takes the value further from it.
raters	Number of coders; two or more.
alpha	One minus the confidence level of the one-sided interval.
critical	The chi-squared critical value. The default is the exact quantile. The published tables were computed with it rounded to 2.71, which makes eight of their forty-eight cells one larger; pass 'critical = 2.71' to reproduce them cell for cell.

Details

You supply three numbers: the kappa you expect ('kappa0', from a pilot or the literature), the smallest kappa you would still be willing to defend ('kappa_lower'), and the prevalence of the code ('prevalence'). Prevalence matters more than people expect: a code applied to a tenth of the segments needs several times the material of one applied to a third.

Value

The number of segments, rounded up. 'Inf' when 'kappa_lower' is not below 'kappa0' – no sample size makes an interval reach a bound at or above the point estimate it is centred on.

Examples

```
# Donner and Rotondi's own Table 2: kappa0 = 0.8, lower bound 0.6,
# prevalence 0.1, two raters
qda_plan_kappa(0.8, 0.6, 0.1, raters = 2)     # 116
qda_plan_kappa(0.8, 0.6, 0.1, raters = 4)     # 62 -- more coders, less material
```

---

qda_plan_themes *How many documents to be reasonably sure of meeting a theme?*

Description

Fugard and Potts (2015) doi:10.1080/13645579.2015.1005453 ask the planning question thematic analysis usually answers with a rule of thumb: if a theme is present in a known share of the population, how many interviews does it take to be, say, 80 percent sure of meeting it at least 'instances' times? The waiting time is negative binomial, which is the same as requiring the binomial tail 'P(X >= instances)' to reach the desired power.

Usage

```
qda_plan_themes(prevalence, instances = 1, power = 0.8, max_n = 10000)
```

Arguments

prevalence	Share of the population in which the theme is present.
instances	How many separate occurrences you want to see.
power	Desired probability of seeing them.
max_n	Upper bound for the search.

Value

The number of documents, or ‘NA‘ when ‘max_n‘ is not enough.

What it assumes, and who disputes it

Themes are treated as present or absent, independent of one another, and certain to surface once present. Braun and Clarke (2016) doi:10.1080/13645579.2016.1195588 reject the premise for reflexive thematic analysis, where themes are constructed rather than discovered and a population prevalence is not a meaningful quantity. The number is a planning aid for work that accepts those assumptions, not a sample size requirement for qualitative research at large.

Examples

```
# Fugard and Potts' Table 1: a theme in 5 % of the population, one
# instance wanted, 80 % power
qda_plan_themes(0.05, instances = 1)     # 32
qda_plan_themes(0.10, instances = 2)     # 29
```

---

qda_plot_code_matrix *Code by document matrix*

Description

The plugin’s code-by-document heat map. Documents are identified by ‘citekey‘ when present, otherwise by title.

Usage

```
qda_plot_code_matrix(fragments, doc_col = "citekey")
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
doc_col	Column identifying the document.

Value

A ’ggplot2’ object.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_plot_code_matrix(frag)
```

`qda_plot_frequencies` *Code frequencies*

Description

The plugin's overview chart, drawn with 'ggplot2': how often each code was assigned.

Usage

```
qda_plot_frequencies(fragments, top = 25, fill = "#4c78a8")
```

Arguments

`fragments`	A fragments data frame from [qda_read_fragments()].
`top`	Show only the 'top' most frequent codes; 'NULL' shows all.
`fill`	Bar colour. When the export carries a 'color' column and 'fill' is 'NULL', the code colours from the code system are used.

Value

A 'ggplot2' object.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_plot_frequencies(frag)
```

`qda_plot_level_agreement`
Plot agreement across the levels of the code system

Description

Plot agreement across the levels of the code system

Usage

```
qda_plot_level_agreement(
  units,
  max_level = NULL,
  measures = c("percent", "fleiss", "alpha")
)
```

Arguments

units	A unit-by-coder matrix from [qda_units()], values being code paths.
max_level	Deepest level to report; defaults to the deepest path.
measures	Which measures to draw.

Value

A 'ggplot2' object.

Examples

```
u <- cbind(ann = c("A/x", "A/y", "B/x"), bob = c("A/y", "A/y", "B/x"))
qda_plot_level_agreement(u)
```

---

qda_plot_mds *Plot the code map*

Description

Plot the code map

Usage

```
qda_plot_mds(fragments, unit = "annotationKey", min_n = 3)
```

Arguments

fragments	A fragments data frame.
unit	Column identifying the segment; defaults to ‘"annotationKey"‘.
min_n	Ignore codes used fewer than ‘min_n‘ times. Rarely used codes make every coefficient unstable.

Value

A 'ggplot2' object.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
qda_plot_mds(frag, min_n = 1)
```

qda_plot_saturation *Saturation curve*

Description

How many *new* codes each successive coding introduced – the curve flattens when a code system stops growing.

Usage

```
qda_plot_saturation(history)
```

Arguments

history	A history data frame from [qda_read_history()].

Value

A 'ggplot2' object.

Examples

```
qda_plot_saturation(qda_read_history(qda_example("zotqda-history.csv")))
```

qda_plot_timeline *Coding progress over time*

Description

The plugin's process view: how the number of codings grew, per coder.

Usage

```
qda_plot_timeline(history)
```

Arguments

history	A history data frame from [qda_read_history()].

Value

A 'ggplot2' object.

Examples

```
h <- qda_read_history(qda_example("zotqda-history.csv"))
qda_plot_timeline(h)
```

qda_read *Read a zotQDA exchange file*

Description

Reads any of the files zotQDA writes and checks it against the contract: the file must declare a known kind, a version this package understands, and the columns the contract promises. A file from a newer major version is refused rather than guessed at.

Usage

```
qda_read(path, format = NULL, strict = TRUE)
```

Arguments

path	Path to the CSV.
format	Optional expected format, e.g. ‘"fragments"‘. When given, a file of a different kind is an error – useful in scripts that must not silently accept the wrong export.
strict	When ‘TRUE‘ (the default), a missing contract column is an error: every column the contract declares is part of it, so an export without one is broken rather than merely different. Extra columns are always allowed – readers address columns by name and ignore what they do not know.

Details

Files are UTF-8 with a byte-order mark and may use ‘,‘ or ‘;‘ as the delimiter, depending on the setting in the plugin; both are detected.

Value

A data frame with the attributes ‘qda_format‘ (e.g. ‘"fragments"‘), ‘qda_version‘ and ‘qda_grain‘.

Examples

```
frag <- qda_read(qda_example("zotqda-fragments.csv"))
attr(frag, "qda_format")
names(frag)[1:4]
```

qda_read_codebook *Read the code system*

Description

Read the code system

Usage

```
qda_read_codebook(path)
```

Arguments

path	Path to the CSV.

Value

A data frame; see [qda_read()].

Examples

```
cb <- qda_read_codebook(qda_example("zotqda-codebook.csv"))
cb$code
```

qda_read_fragments *Read the coded fragments*

Description

One row per annotation and code. This is the table most analyses start from.

Usage

```
qda_read_fragments(path)
```

Arguments

path	Path to the CSV.

Value

A data frame; see [qda_read()].

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
nrow(frag)
```

qda_read_history *Read the coding history*

Description

Every logged coding event, ‘add‘ and ‘remove‘, oldest first.

Usage

```
qda_read_history(path)
```

Arguments

path	Path to the CSV.

Value

A data frame; see [qda_read()].

Examples

```
h <- qda_read_history(qda_example("zotqda-history.csv"))
table(h$action)
```

qda_read_mapping *Read the consensus mapping*

Description

Which coder code corresponds to which consensus code. This is what lets phase-2 codings be analysed in terms of the consensus code system without anyone rewriting a coding – derived codings would inflate every agreement figure by construction.

Usage

```
qda_read_mapping(path)
```

Arguments

path	Path to the CSV.

Value

A data frame; see [qda_read()].

Examples

```
m <- qda_read_mapping(qda_example("zotqda-konsens-abbildung.csv"))
m$consensusCode
```

qda_read_qdpx *Read a REFI-QDA project file*

Description

Reads a ‘.qdpx‘ archive – the interchange format MAXQDA, ATLAS.ti, NVivo, QDA Miner and Dedoose all write – into the same tables ‘qda_read()‘ produces from the zotQDA CSV exports. Everything downstream then works unchanged.

Usage

```
qda_read_qdpx(path, warn = TRUE)
```

Arguments

path	Path to a ‘.qdpx‘ archive.
warn	Emit a warning listing what the file cannot support. Leave it on until you have read the list once.

Details

A ‘.qdpx‘ supports a **subset** of these analyses, and the difference is not a detail. The exchange CSVs were designed for this package; ‘.qdpx‘ was designed to move a project between programs. What is missing is listed in the returned object’s ‘limitations‘ and, unless ‘warn = FALSE‘, printed as a warning. Nothing is guessed: a column that cannot be filled is empty, and an analysis needing it fails rather than returning a flattering number.

What does survive is more than one might expect: the code tree with its GUIDs as stable identities, the coders, the timestamps, and the character positions of text selections – so the unitizing measures work too.

Needs the suggested package **xml2**.

Value

A list of class ‘qda_qdpx‘ with the elements ‘fragments‘, ‘codebook‘, ‘history‘, ‘uncoded‘, ‘multi_coded‘, ‘coders‘, ‘sources‘, ‘skipped‘ and ‘limitations‘.

Examples

```
if (requireNamespace("xml2", quietly = TRUE)) {
  p <- qda_read_qdpx(qda_example("sample.qdpx"), warn = FALSE)
  nrow(p$fragments)
  p$coders
  p$limitations[1]
}
```

qda_saturation *Distinct codes over the course of coding*

Description

Distinct codes over the course of coding

Usage

```
qda_saturation(history)
```

Arguments

history	A history data frame from [qda_read_history()].

Value

A data frame with ‘step‘ and ‘codes‘.

Examples

```
qda_saturation(qda_read_history(qda_example("zotqda-history.csv")))
```

qda_saturation_index *How saturated is this material, and how much more would it take?*

Description

A saturation curve that is still climbing tells you nothing about how far from the top it is. Lowe, Norris, Farris and Babbage (2018) doi:10.1177/1525822X17749386 fit the accumulation of themes to a growth model, which estimates the number of themes that exist to be found (‘A‘) and thereby turns "still climbing" into a percentage.

Usage

```
qda_saturation_index(cumulative, model = c("IW", "IS", "SW"))
```

Arguments

cumulative	Distinct themes after each document, in coding order – the ‘cumulative‘ column of [qda_new_codes()], or a plain vector.
model	‘"IS"‘, ‘"IW"‘ or ‘"SW"‘.

Details

Their saturation index is the share of the estimable themes you already have, ‘100 * T_N / floor(A)‘. Because it comes from a fitted ‘A‘, it also answers the question a project actually asks halfway through: how many more documents for another ten points.

Value

A list with the fitted ‘A‘ and ‘b‘, the ‘index‘ (per cent), the ‘fitted‘ curve, the residual ‘rmse‘, and ‘model‘.

Which model

‘IS‘ assumes observations are independent, ‘IW‘ that overlap grows with what is already known, ‘SW‘ that it grows with the number of observations. They differ mainly in how fast the curve flattens, and Lowe et al. found no single winner – fit all three and look at which describes your data, rather than picking one in advance.

Examples

```
# a curve that is clearly flattening
qda_saturation_index(c(8, 13, 16, 18, 19, 20, 20, 21))$index
```

---

qda_saturation_ratio *Saturation as a number you can report*

Description

A saturation curve shows a trend; it does not answer "how many documents were enough". Guest, Namey and Chen (2020) doi:10.1371/journal.pone.0232076 operationalised the question with three parameters and one ratio: a base of documents whose codes count as what is already known, a run of consecutive later documents inspected for new codes, and the share of new information that still counts as saturated.

Usage

```
qda_saturation_ratio(
  new_codes,
  base_size = 4,
  run_length = 2,
  threshold = 0.05
)
```

Arguments

new_codes	New codes per document, in order – either the data frame from [qda_new_codes()] or a plain numeric vector.
base_size	Documents forming the base; Guest et al. recommend 4 and found the choice barely mattered.
run_length	Consecutive documents per run, successive runs overlapping by one.
threshold	Share of new information still counting as saturated.

Details

The result is reported as ‘"6+2"‘: saturation declared at document 6, confirmed over a run of 2.

Value

A list with ‘notation‘ (the string for the paper, or ‘NULL‘), ‘saturated_at‘, ‘base_codes‘, the table of ‘runs‘, and ‘reason‘ when the question could not be answered.

What this is not

This is *code* saturation, and only that. Hennink, Kaiser and Marconi (2017) doi:10.1177/1049732316665344 distinguish it from meaning saturation, which no algorithm can see, and Braun and Clarke (2019) doi:10.1080/2159676X.2019.1704846 reject saturation altogether as a criterion for reflexive thematic analysis. If you report the number, report which conception it belongs to.

Examples

```
qda_saturation_ratio(c(4, 3, 2, 1, 1, 0, 0, 0))$notation
qda_saturation_ratio(c(4, 4, 4, 4, 4, 4))$reason
```

---

qda_segments *Segments from a fragments export*

Description

Turns the position columns into the segments the unitizing measures work on. Only ‘positionKind == "text"‘ gives a continuum to measure boundaries on; PDF rectangles do not, and are dropped with a warning rather than quietly approximated.

Usage

```
qda_segments(fragments, coder = "codedBy", value = "code")
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
coder	Column identifying the coder.
value	Column holding the category; ‘"code"‘ is the readable path, ‘"codeId"‘ the identity that survives renaming.

Details

The position columns arrived with a later version of the plugin. An older export simply does not have them, and this function says so instead of returning an empty result that looks like disagreement.

Value

A list of data frames, one per coder, each with ‘start‘, ‘end‘ and ‘value‘.

Examples

```
frag <- data.frame(
  codedBy = c("ann", "bob"), code = c("A", "A"),
  positionKind = c("text", "text"),
  positionStart = c(0, 5), positionEnd = c(20, 22)
)
qda_segments(frag)
```

`qda_spec_read` *Read a chart specification written by qdaZ*

Description

Every chart in qdaZ can be saved as a "data + spec" pair: the data as CSV and the chart as a Vega-Lite specification. The specification carries its provenance in ‘usermeta‘, so a reader can tell which analysis produced it.

Usage

```
qda_spec_read(path)
```

Arguments

`path`	Path to the ‘.json‘ specification.

Value

The parsed specification, with the attributes ‘qdaz_analysis‘ and ‘qdaz_version‘ when the file states them.

Examples

```
f <- tempfile(fileext = ".json")
writeLines('{"mark":"bar","usermeta":{"contract":"zotqda-exchange",
  "version":1,"analysis":"demo"}}', f)
spec <- qda_spec_read(f)
attr(spec, "qdaz_analysis")
unlink(f)
```

qda_spec_render *Render a qdaZ chart specification*

Description

Renders the original Vega-Lite chart, so a figure looks exactly as it did in the plugin. Requires the 'vegawidget' package; use the 'qda_plot_*' functions for 'ggplot2' versions that need no extra dependency.

Usage

```
qda_spec_render(spec, data = NULL)
```

Arguments

spec	A specification from [qda_spec_read()], or a path to one.
data	Optional data frame to inline into the specification, e.g. the CSV saved next to it.

Value

A 'vegawidget' object.

Examples

```
if (requireNamespace("vegawidget", quietly = TRUE)) {
  spec <- list(`$schema` = "https://vega.github.io/schema/vega-lite/v5.json",
               mark = "point")
  qda_spec_render(spec)
}
```

qda_srqr *An SRQR checklist, pre-filled with what the data can answer*

Description

SRQR (O'Brien et al. 2014, doi:10.1097/ACM.0000000000000388) is the other reporting standard journals ask for, and it is the broader of the two: 21 standards covering the whole report rather than COREQ's focus on interviews and focus groups. Use it when your material is not interview transcripts, or when the journal names it.

Usage

```
qda_srqr(fragments = NULL, history = NULL, codebook = NULL, software = NULL)
```

Arguments

fragments	A fragments data frame, or 'NULL'.
history	A history data frame, or 'NULL'; enables the saturation item.
codebook	A codebook data frame, or 'NULL'; enables the coding-tree item.
software	Free text for item 27. The default names the tools in use.

Value

A data frame with one row per standard: 'item', 'section', 'name', 'description', 'answer' and 'filled'.

Where the agreement figure belongs

Unlike COREQ, SRQR has a home for it. Standard S15, techniques to enhance trustworthiness, names the audit trail explicitly – and the coding log is one. If you computed intercoder agreement, that is the item it answers. [qda_coreq()] has to say the opposite, because COREQ never asks.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
sq <- qda_srqr(frag)
sq[sq$filled, c("item", "name")]
```

---

qda_srqr_markdown *The SRQR checklist as Markdown*

Description

The SRQR checklist as Markdown

Usage

```
qda_srqr_markdown(srqr, title = "SRQR checklist", file = NULL)
```

Arguments

srqr	A data frame from [qda_srqr()].
title	Heading for the document.
file	Optional path to write to.

Value

A character vector of Markdown lines.

Examples

```
frag <- qda_read_fragments(qda_example("zotqda-fragments.csv"))
head(qda_srqr_markdown(qda_srqr(frag)), 6)
```

qda_theme_power *The same question as power*

Description

Given the documents you have, how likely are you to meet a theme of this prevalence the desired number of times?

Usage

```
qda_theme_power(n, prevalence, instances = 1)
```

Arguments

n	Number of documents.
prevalence	Share of the population in which the theme is present.
instances	How many separate occurrences you want to see.

Value

A probability.

Examples

```
qda_theme_power(32, 0.05)          # about 0.8, the flip side of the table
```

qda_timeline *Cumulative codings per coder*

Description

Cumulative codings per coder

Usage

```
qda_timeline(history)
```

Arguments

history	A history data frame from [qda_read_history()].

Value

A data frame with ‘time‘, ‘user‘ and ‘cumulative‘.

Examples

```
qda_timeline(qda_read_history(qda_example("zotqda-history.csv")))
```

`qda_unitizing_alpha` *Krippendorff's alpha for unitizing*

Description

Every other coefficient in this package assumes the segments already line up and only asks whether the categories agree. That assumption does a lot of work. This one asks the prior question: did the coders mark the same stretches of text at all?

Usage

```
qda_unitizing_alpha(segments, metric = NULL)
```

Arguments

`segments`	A list of coders' segments, from [qda_segments()].
`metric`	Squared difference between two values; the default is nominal (0 when equal, 1 otherwise). Pass 'function(a, b) 0' to measure identification alone and ignore the codes.

Details

Krippendorff (1995) doi:10.2307/271061, in the form given in the replacement of section 12.4 of *Content Analysis* (3rd ed.), equations 16 to 19. Gaps are not compared with each other – two coders agreeing that a stretch is irrelevant is not evidence of reliable unitizing.

Established QDA software settles this with a single overlap threshold, yes or no. What that discards is precisely the information about how the boundaries differ.

Value

A list with 'alpha', the observed and expected disagreement 'Do' and 'De', the number of 'intersections' behind 'Do' and the number of 'units'; or 'NA' when fewer than two coders contributed.

Comparing the two

Ignoring the categories lowers the observed disagreement, but it lowers the *expected* disagreement too, because randomly paired units no longer differ by category either. Which effect wins depends on whether the coders actually disagreed about categories: where they did, alpha rises; where they agreed throughout, alpha can fall. Compare the two 'Do' values, not the two alphas.

Examples

```
ann <- data.frame(start = c(0, 40), end = c(20, 60), value = c("A", "B"))
bob <- data.frame(start = c(0, 40), end = c(20, 60), value = c("A", "B"))
qda_unitizing_alpha(list(ann, bob))$alpha        # 1
```

qda_units *Build the unit-by-coder matrix*

Description

Intercoder measures need one row per unit of analysis and one column per coder. The fragments export is longer than that – one row per annotation and code – so it has to be reshaped, and two decisions have to be made explicitly rather than by accident.

Usage

```
qda_units(
  fragments,
  uncoded = NULL,
  unit = "annotationKey",
  coder = "codedBy",
  value = "code",
  no_code = "(no code)",
  level = NULL
)
```

Arguments

fragments	A fragments data frame from [qda_read_fragments()].
uncoded	Optionally the matching uncoded export, so segments no coder coded become their own category.
unit	Column identifying the unit of analysis.
coder	Column identifying the coder.
value	Column holding the category; ‘"code"‘ is the readable path, ‘"codeId"‘ the identity that survives renaming.
no_code	Label for a segment a coder left uncoded.
level	Flatten paths to this many levels first; see [qda_level_agreement()].

Details

Segments nobody coded are a category, not a gap. Agreement about what is *not* relevant is agreement. Pass the ‘uncoded‘ export and those segments enter as their own category; leave it out and the figures only describe the segments at least one coder marked, which is a different and usually more flattering question.

A segment two coders coded twice is set aside. Where a coder gave one segment several codes there is no single value to compare, so the cell becomes missing and is counted in the ‘multi‘ attribute. The honest way to include those segments is the per-code binary view, [qda_units_binary()]. Reporting an overall figure that quietly dropped a tenth of the material is not.

Value

A character matrix, units in rows, coders in columns, ‘NA‘ where a coder did not rate a unit, with the attribute ‘multi‘ giving the number of cells set aside because of multiple coding.

Examples

```
frag <- data.frame(
  annotationKey = c("s1", "s1", "s2", "s2"),
  codedBy = c("ann", "bob", "ann", "bob"),
  code = c("A", "A", "B", "A")
)
qda_units(frag)
```

`qda_units_binary` *The per-code binary view*

Description

Turns one code into a yes/no judgement per unit, which is how a multiply-coded body of material can still be assessed: every code is asked about separately, and a segment carrying three codes contributes to all three questions instead of being dropped.

Usage

```
qda_units_binary(
  fragments,
  code,
  unit = "annotationKey",
  coder = "codedBy",
  value = "code",
  uncoded = NULL
)
```

Arguments

`fragments`	A fragments data frame from [qda_read_fragments()].
`code`	The code to ask about.
`unit`	Column identifying the unit of analysis.
`coder`	Column identifying the coder.
`value`	Column holding the category; ‘"code"‘ is the readable path, ‘"codeId"‘ the identity that survives renaming.
`uncoded`	Optionally the matching uncoded export, so segments no coder coded become their own category.

Value

A character matrix as in [qda_units()], with values ‘"yes"‘ and ‘"no"‘.

Examples

```
frag <- data.frame(
  annotationKey = c("s1", "s1", "s2", "s2"),
  codedBy = c("ann", "bob", "ann", "bob"),
  code = c("A", "A", "B", "A")
)
qda_units_binary(frag, "A")
```

---

qda_wilson *A proportion with an interval that behaves at the edges*

Description

Wilson (1927) doi:10.1080/01621459.1927.10502953 rather than the textbook normal approximation, which Brown, Cai and DasGupta (2001) doi:10.1214/ss/1009213286 show to be erratic for small samples and degenerate at nought or one. Code prevalences live exactly there: a code used in two of forty segments must not get an interval reaching below zero.

Usage

```
qda_wilson(successes, total, level = 0.95)
```

Arguments

successes	Count.
total	Sample size.
level	Confidence level.

Value

A list with ‘estimate‘, ‘lo‘, ‘hi‘ and ‘n‘.

Examples

```
qda_wilson(2, 40)
qda_wilson(0, 10)   # upper bound only, and it stays inside [0, 1]
```

qda_window_diff *WindowDiff and Pk: how far apart are two segmentations?*

Description

Two error rates from text segmentation, reported together because they disagree in an informative way. Both slide a window across the continuum; ‘qda_window_diff()‘ compares how many boundaries each segmentation puts in it, ‘qda_pk()‘ only whether there is one at all. A spurious extra boundary therefore costs something in the first and nothing in the second.

Usage

```
qda_window_diff(reference, hypothesis, length_, k = NULL)

qda_pk(reference, hypothesis, length_, k = NULL)
```

Arguments

reference, hypothesis	Segment data frames, as from [qda_segments()].
length_	Length of the continuum in characters.
k	Window width; the default is half the average reference segment.

Details

Neither is chance-corrected – for that, use [qda_unitizing_alpha()]. What they offer instead is comparability with the segmentation literature, and a number that behaves sensibly for near misses: a boundary two characters off is nearly right, and both measures say so.

Pevzner and Hearst (2002) doi:10.1162/089120102317341756; Beeferman, Berger and Lafferty (1999) doi:10.1023/A:1007506220214.

Value

A number between 0 and 1; 0 means the boundaries coincide.

Examples

```
ref <- data.frame(start = c(0, 20, 40), end = c(20, 40, 60))
hyp <- data.frame(start = c(0, 22, 40), end = c(22, 40, 60))
qda_window_diff(ref, ref, 60)   # 0
qda_window_diff(ref, hyp, 60)
qda_pk(ref, hyp, 60)
```

Index