Package 'rmoriedata'

November 18, 2026

Type Package

Title Disclosure Control for Aggregate Releases with Open Datasets

Version 0.4.3

Description Disclosure-control tools for releasing aggregate statistics, with open datasets to use them on. Differential privacy: an exact discrete Laplace sampler for counts and the analytic Gaussian mechanism for means, drawing on the operating system's random source, with privacy-budget accounting. Checks of k-anonymity and l-diversity, and cell suppression audited for exact recovery from the margins. The datasets are snapshots of publicly available tables from open-data portals built on the Comprehensive Knowledge Archive Network ('CKAN', <https://ckan.org/>), 'Socrata' (<https://dev.socrata.com/>), and 'Opendatasoft' (<https://www.huwise.com/>) (Chicago, New York City, Toronto, Vancouver, and others), Statistics Canada Canadian Centre for Justice and Community Safety Statistics ('CCJS') tables, a multi-agent-reviewed corpus of Ontario Special Investigations Unit ('SIU', <https://www.siu.on.ca/>) director's reports, and synthetic fixtures, with functions that read and verify them.

License AGPL (>= 3)

Encoding UTF-8

Language en-GB

Depends R (>= 4.3.0)

Imports stats, utils, rmoriebricklayer (>= 1.1.0)

LinkingTo rmoriebricklayer

Suggests withr, testthat (>= 3.0.0), nanoparquet, tibble, curl, jsonlite, knitr, rmarkdown

VignetteBuilder knitr

LazyData true

LazyDataCompression xz

Config/testthat/edition 3

URL https://rootcoder007.github.io/rmoriedata/, https://github.com/rootcoder007/rmoriedata

BugReports https://github.com/rootcoder007/rmoriedata/issues

biocViews Software, ExperimentData

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Vansh Singh Ruhela [aut, cre] (ORCID:
<https://orcid.org/0009-0004-1750-3592>)

Maintainer Vansh Singh Ruhela <vsruhela@proton.me>

Repository CRAN

Date/Publication 2026-11-18 09:50:14 UTC

Contents

arrest_sample . . . 3
cellsupp . . . 4
chiclr . . . 5
chiload . . . 6
cihiget . . . 8
cihitbls . . . 9
complaint_sample . . . 10
dpbudget . . . 12
dpgmean . . . 13
dplapcnt . . . 14
dplaphist . . . 15
dpspent . . . 16
hostcat . . . 17
kanonvrf . . . 18
ldivvrf . . . 20
mdask . . . 21
mdcat . . . 22
mddict . . . 23
mdllmconf . . . 24
mdload . . . 24
mdpath . . . 25
mdsums . . . 26
mdverify . . . 27
morie_core . . . 28
rmoriedata-victoria . . . 29
siuload . . . 30

Index **33**

`arrest_sample` *Chicago arrests sample*

Description

A CRAN-safe slice of the City of Chicago "Arrests" dataset. For the full ~1.5M-row dataset use `chiload("arrests", full = TRUE)`.

Usage

```
arrest_sample
```

Format

A base `data.frame` with up to 25,000 rows and 8 columns:

case_number Chicago PD records-division number (character).

date_iso Arrest timestamp as an ISO-8601 string with offset.

date Arrest timestamp as `POSIXct` (America/Chicago).

race Recorded race of the arrestee (character).

charge_type Charge type of the primary charge (F/M/etc.).

charge_class Charge class of the primary charge.

charge_desc Description of the primary charge.

charge_statute Statute of the primary charge.

Source

City of Chicago Open Data Portal, "Arrests" (dataset `dpt3-jri9`). https://data.cityofchicago.org/

See Also

`chiload`

Examples

```
data(arrest_sample)
dim(arrest_sample)
str(arrest_sample)

# Recorded race distribution.
sort(table(arrest_sample$race), decreasing = TRUE)

# Charge severity (F = felony, M = misdemeanour, ...).
sort(table(arrest_sample$charge_type), decreasing = TRUE)

# Most frequent primary charges.
```

```
head(sort(table(arrest_sample$charge_desc), decreasing = TRUE), 5)

# Cross-tab race x charge type.
with(arrest_sample, table(race, charge_type))
```

`cellsupp` *Cell suppression with complementary suppression and a recoverability audit*

Description

Suppresses (sets to `NA`) every positive count below `threshold`, then, when `retcomp = TRUE`, hides further cells until no suppressed value can be solved exactly from the table's row and column totals, which a published table normally carries.

Usage

```
cellsupp(tbl, threshold = 5, retcomp = TRUE)
```

Arguments

`tbl`	A numeric matrix or 2-D table of counts. Will be coerced to matrix; row/column names are preserved.
`threshold`	Minimum count to remain unsuppressed. Default 5.
`retcomp`	Logical; if `TRUE` (default), add complementary suppressions until no hidden cell is recoverable from the row and column totals.

Details

Complements are added to a fixed point. A row or column holding exactly one hidden cell gives that cell away by subtraction, so every line with a hidden cell gets a second one; a complement placed for a row is a hidden cell in its column too, so the passes repeat until no line holds a lone hidden cell. That condition is necessary but not sufficient, so the result is then audited exactly: a hidden cell is recoverable when the linear system of row and column totals determines it (its indicator lies in the row space of the constraints over the hidden cells). Any recoverable cell receives another complement in its row, and the audit repeats. Complements are taken from positive cells, preferring one in a line that already holds a hidden cell (which creates no new lone cell), then the largest such cell, which leaves the widest range for the hidden values. A zero cell is never used as a complement: it would bound the hidden values from below at nothing.

The audit covers exact recovery from the margins. It does not compute the feasible interval of each hidden cell, which bounds from non-negativity can narrow; a table whose hidden values must stay inside a wide interval needs an interval audit (linear programming) on top.

Cells must be non-negative whole numbers or `NA`; infinite, fractional and negative cells are refused. `NA` cells in the input pass through unchanged and are treated as unpublished: they count as hidden in the audit (their values are not known to a reader) and receive complements like any hidden cell, but they are not counted as primary or complementary suppressions. A zero cell is published as it is: the threshold rule protects the few contributors of a small cell, and a zero has none.

Value

A list with class "cellsupp":

suppressed numeric matrix, suppressed cells set to NA.

primary_mask logical matrix, TRUE for primary suppressions.

complementary_mask logical matrix, TRUE for complementary suppressions (all FALSE when retcomp = FALSE).

recoverable_mask logical matrix, TRUE for a hidden cell whose value the row and column totals determine exactly.

protected logical, TRUE when no hidden cell is recoverable. Test this before publishing.

n_primary integer.

n_complementary integer.

n_recoverable integer.

threshold the threshold used.

Examples

```
tbl <- matrix(c(120, 3, 47, 88, 2, 99, 14, 51, 60),
  nrow = 3,
  dimnames = list(c("A", "B", "C"), c("X", "Y", "Z"))
)

res <- cellsupp(tbl, threshold = 5)
res$suppressed # NA where suppressed
res$protected # no hidden cell can be solved from the totals
res$n_primary # cells below threshold
res$n_complementary # extra cells hidden to protect them

# Without complements the small cells are hidden but recoverable.
bare <- cellsupp(tbl, threshold = 5, retcomp = FALSE)
bare$protected
bare$recoverable_mask

# Works on a 2-D table too; NA cells pass through untouched.
t2 <- as.table(matrix(c(2, 40, 30, 1), 2,
  dimnames = list(c("a", "b"), c("c", "d"))
))
cellsupp(t2, threshold = 5)$suppressed
```

---

chiclr *Clear the cache of full Chicago datasets*

Description

Deletes the Parquet files that chiload(full = TRUE) wrote to the cache directory (tempdir() by default, or the directory named in options(rmoriedata.cache_dir =)): the arrests_full.parquet and complaints_full.parquet files, their .sha256 checksum files and any unfinished write (write-*.parquet and its .sha256). Nothing else in that directory is touched.

Usage

```
chiclr()
```

Value

The cache directory, invisibly.

Examples

```
chiclr()
```

---

chiload *Load Chicago crime or arrest data*

Description

Returns the bundled sample by default, or fetches the full dataset from the City of Chicago SODA API when `full = TRUE`. A complete fetch is cached as Parquet under `tempdir()` for the session; to keep it across sessions name a directory first, for example `options(rmoriedata.cache_dir = tools::R_user_dir("rmoriedata", "cache"))`, and drop it with `chiclr()`. Nothing is written outside `tempdir()` unless you set that option. The result can be returned as a base data frame, a tibble, or written to a Parquet file whose path is returned – the last being the recommended bridge for Python (`pandas.read_parquet`).

Usage

```
chiload(
  type = c("arrests", "complaints"),
  as = c("data.frame", "tibble", "parquet_path"),
  full = FALSE,
  mirror = getOption("rmoriedata.mirror", NULL),
  limit = NULL,
  fraction = NULL,
  refresh = FALSE
)
```

Arguments

`type`	One of `"arrests"` or `"complaints"`.
`as`	Return format: `"data.frame"` (default), `"tibble"`, or `"parquet_path"` (writes a Parquet file under `tempdir()` and returns its path; nothing is written to the user's home). Parquet keeps no time zone, so `date` reads back as the same instant in UTC; `date_iso` keeps the local time and offset.

`full` If `TRUE`, fetch the complete dataset from Socrata (network, large) instead of the bundled sample; a complete fetch is cached as Parquet (see Description). The result has the bundled sample's columns, in its order and with its types, derived from Socrata's export as the sample was. Text in Socrata's export is read as decimal numbers only (hexadecimal, `Inf` and numbers past the double range are not numbers here); a value that does not fit its column becomes `NA`, with one warning per column that counts such values and shows the first. A mirror's or the cache's file in the sample's own columns must fit exactly, and is refused otherwise with an error naming the column and the file.

`mirror` Optional base URL of an r-universe/drat mirror, or a local directory, to try before Socrata (offline-friendly fallback). Defaults to `getOption("rmoriedata.mirror")`. A URL is downloaded over https and a local directory read in place; either way the file is checked against `<type>_full.parquet.sha256` when the mirror has one beside it, and a file that does not match is refused. A local file with no `.sha256` beside it is read as it is, unchecked. If the mirror fails, Socrata is used. The mirror's `<type>_full.parquet` must use snappy or no compression and data pages v1: the package's own reader supports nothing else.

`limit` Optional row cap. For a `full = TRUE` fetch it is passed to the Socrata `$limit` parameter (a bounded fetch skips the mirror and is never written to the full-dataset cache); otherwise it keeps the first `limit` rows of the bundled sample. Default `NULL` fetches everything.

`fraction` Optional share of the dataset, in `(0, 1]`, for a `full = TRUE` fetch: the live row count is looked up and `limit` is set to `ceiling(total * fraction)` (of the bundled sample's rows otherwise). Give either `fraction` or `limit`, not both.

`refresh` If `TRUE`, ignore the cache of the complete dataset and fetch it again (the cache is rewritten). A cache whose bytes do not match its `.sha256`, or that is not a Parquet file, is deleted and fetched again regardless, and one without a usable `.sha256` is fetched again and rewritten. A cache that cannot be read for any other reason (memory that runs short, a read that fails, an option such as `rmoriedata.parquet_max_rows`) is kept: the call stops and says why. The cache is opened once and read a page at a time, each part checked against the digest taken as it was opened; it is never held whole.

Details

Parquet I/O uses this package's own native codec (R/aaa_parquet.R), so no **arrow** install is required.

Value

A `data.frame`/`tibble`, or a length-1 character Parquet path when `as = "parquet_path"`.

Examples

```
# `type` selects the dataset; the bundled sample is returned by default.
comp <- chiload("complaints") # reported incidents
arr <- chiload("arrests") # arrests
nrow(comp)
nrow(arr)
```

```
head(sort(table(comp$primary_type), decreasing = TRUE), 5)

# `as = "tibble"` returns a tibble when the package is installed.
if (requireNamespace("tibble", quietly = TRUE)) {
  tb <- chiload("complaints", as = "tibble")
  class(tb)
}


# `as = "parquet_path"` writes a Parquet file and returns its path --
# the recommended bridge to Python (pandas.read_parquet). Offline: the
# bundled sample is written, no network.
pq <- chiload("arrests", as = "parquet_path")
file.exists(pq)



# `full = TRUE` fetches from the live Chicago SODA API; `limit` bounds
# the request (seconds, not minutes) and try() keeps the example
# graceful when the service is unreachable. Omit `limit` for the
# complete multi-million-row dataset (cached, see Description); `mirror`
# tries an offline-friendly Parquet mirror first when set.
# `fraction = 0.001` takes a share of the dataset (0.1% of all rows)
# instead of a row count; the live total is looked up first.
big <- try(chiload("complaints", full = TRUE, limit = 200))
if (!inherits(big, "try-error")) nrow(big)
```

`cihiget` *Download a CIHI data table (live, with Wayback fallback)*

Description

Resolves a CIHI table from the bundled catalogue and downloads it to `dest`, falling back to the Internet Archive snapshot if the live CIHI URL has rotated or been removed. The download + fallback runs through **rmoriebricklayer**'s shared C++/libcurl foundation (`blfetch()`).

Usage

```
cihiget(which, dest = NULL, timeout = 120L)
```

Arguments

`which`	A row index into `cihitbls()`, or a string matched (case-insensitively, as a substring) against table titles. Must resolve to exactly one table.
`dest`	Destination file path. Default: a tempfile with the table's own extension.
`timeout`	Per-request timeout in seconds, from 1 to 86400 (a day); a fraction is rounded up to the next whole second.

Value

The `dest` path, invisibly. Errors if both the live URL and its Wayback fallback fail.

Examples

```
# Offline: inspect the catalogue to choose a `which` argument.
cat <- cihitbls()
head(cat$title, 3)


# Downloads a table from the live CIHI web service; try() keeps the
# example graceful when the service is unreachable.
# `which` by title substring (case-insensitive; must match exactly one).
f1 <- try(cihiget("Hospital Beds, 2024")) # -> tempfile path

# `which` by row index into cihitbls(); `dest` chooses the
# output path and `timeout` bounds each request (seconds).
f3 <- try(cihiget(1,
  dest = tempfile(fileext = ".xlsx"),
  timeout = 60
))

# An ambiguous substring errors and lists the candidates:
try(cihiget("data"))
```

`cihitbls` *Catalogue of CIHI open data-table workbooks (with Wayback fallbacks)*

Description

Returns the bundled catalogue of the public data-table `.xlsx` workbooks published on the Canadian Institute for Health Information (CIHI) "Access data and reports > Data tables" page (https://www.cihi.ca/en/access-data-and-reports/data-tables). Each row carries the table title, its direct `url`, and a `wayback_url` snapshot on the Internet Archive so the table stays retrievable even if CIHI rotates or removes the live file.

Usage

```
cihitbls(archonly = FALSE)
```

Arguments

`archonly`	If `TRUE`, drop rows with no Wayback snapshot. Default `FALSE` (return the full catalogue).

Details

Download any row with `cihiget()`, which tries `url` first and falls back to `wayback_url`. The Wayback snapshots were resolved with `rmoriebricklayer::wbsnpurl()`.

Value

A `data.frame` with columns `title`, `url`, `wayback_url`.

Source

Canadian Institute for Health Information, Data tables (https://www.cihi.ca/en/access-data-and-reports/data-tables). Snapshotted to the Internet Archive (https://web.archive.org). Catalogue current as of 2026-07.

Examples

```
# Full catalogue: title, live url, Wayback snapshot url.
cat <- cihitbls()
nrow(cat)
names(cat)
head(cat$title, 3)

# `archonly = TRUE` keeps only rows that have a Wayback snapshot,
# i.e. tables still retrievable if CIHI rotates the live file.
arch <- cihitbls(archonly = TRUE)
nrow(arch) # <= nrow(cat)
all(nzchar(arch$wayback_url)) # TRUE

# Find a table by keyword before fetching it.
cat$title[grepl("hospital", cat$title, ignore.case = TRUE)][1:3]
```

---

complaint_sample *Chicago reported-crime sample ("complaints")*

Description

A CRAN-safe slice of the City of Chicago "Crimes – 2001 to present" dataset (reported incidents), filtered to geocoded rows from 2020 onward. For the full dataset use `chiload("complaints", full = TRUE)`.

Usage

```
complaint_sample
```

Format

A base `data.frame` with up to 25,000 rows and 16 columns:

case_number Chicago PD records-division number (character).

date_iso Incident timestamp as an ISO-8601 string with offset (lossless across R/Python).

date Incident timestamp as `POSIXct` (America/Chicago).

iucr Illinois Uniform Crime Reporting code (character).

primary_type Primary FBI crime classification.

description Secondary description of the offense.

arrest Whether an arrest was made (logical).

domestic Whether domestic-violence related (logical).

beat,district,ward,community_area Geographic area codes (integer).

fbi_code FBI crime code (character).

year Year of the incident (integer).

latitude,longitude WGS84 coordinates (numeric).

Source

City of Chicago Open Data Portal, "Crimes - 2001 to present" (dataset `ijzp-q8t2`). https://data.cityofchicago.org/

See Also

`chiload`

Examples

```
data(complaint_sample)
dim(complaint_sample)
str(complaint_sample)

# Most common offense types.
head(sort(table(complaint_sample$primary_type), decreasing = TRUE), 5)

# Arrest rate among reported incidents.
mean(complaint_sample$arrest)

# Incidents per year (the sample spans 2020+).
table(complaint_sample$year)

# Domestic-violence-flagged incidents by type.
head(sort(table(complaint_sample$primary_type[complaint_sample$domestic]),
  decreasing = TRUE
), 3)
```

dpbudget *A privacy budget that DP releases are charged against*

Description

Creates a budget of `epsilon` (and `delta`) that the mechanisms in this package draw down when passed as `budget =`. Releases compose by basic composition: the epsilons and deltas of the releases charged to one budget add up, and a release that would exceed the budget is refused before any noise is drawn. A hundred releases at `epsilon = 1` spend `epsilon = 100`; a budget makes that visible and enforceable.

Usage

```
dpbudget(epsilon, delta = 0)
```

Arguments

`epsilon`	Total epsilon available (positive).
`delta`	Total delta available (0 for pure DP only).

Details

A budget is an in-memory ledger held by the R process that created it. A forked worker (for example under `parallel::mclapply()` or `parallel::mcparallel()`) gets its own copy, so its spending would never reach the parent and every worker could spend the whole budget again. A release charged to a budget from any process other than the one that created it is therefore refused with an error. Spend a budget in the process that created it; to release from workers, give each worker its own budget and account for the total yourself by composition (the epsilons and deltas of all workers add up). A budget saved and read back in another R session is refused in the same way, even one whose process has the same PID (in a container whose entrypoint is `Rscript` every session is PID 1): each budget carries a random token drawn from the operating system's CSPRNG when the package is loaded, and a release is charged only in the process, and the load of the package, that made the budget.

A copy of a budget made in the same session is refused too: one written with `saveRDS()` and read back with `readRDS()`, passed through `serialize()` and `unserialize()`, or rebuilt from its fields. The copy is a second ledger that would start from what the budget had spent when it was copied, so the two together could spend the budget twice; every budget is registered in the package when it is made, and only that object can be charged. Assigning a budget to another name (`b2 <- b`) makes no copy: both names are the one ledger. What is not prevented: code that sets a budget's fields by hand, or that creates several budgets for one dataset. A budget is an accounting aid for honest code, not a defence against code that means to overspend.

A budget of `delta = 0` pays only for pure-DP releases: a release with any `delta > 0` is refused. The totals are compared with a slack for floating-point rounding only, so that ten releases at `epsilon = 0.1` fit a budget of 1: the slack is `(n + 2)` units in the last place of the larger of the budget and the total, `n` being the number of releases (below a millionth of a millionth of the budget for 4,000 releases), and a release more than twice what remains is refused whatever the slack.

A release is charged before its noise is drawn. If it is interrupted or fails after that, the budget stays charged though nothing was released: the safe side, since a release that went out uncharged would spend privacy the ledger does not show.

Value

An object of class `"dpbudget"` (an environment, so charges persist across calls).

See Also

`dpspent()`

Examples

```
b <- dpbudget(epsilon = 2)
dplapcnt(42, epsilon = 1, budget = b)
dpspent(b)
# a second release at epsilon = 1.5 would exceed the remaining 1
try(dplapcnt(42, epsilon = 1.5, budget = b))
```

`dpgmean` *Differentially-private mean via the analytic Gaussian mechanism*

Description

Releases an (ϵ, δ)-DP mean of a bounded numeric vector, as a noised sum over a noised count, each with half the budget. Under the add-or-remove-one relation every mechanism here uses, one record moves the clipped sum by at most `max(abs(lower), abs(upper))` and the count by 1, so neither the sum nor `length(x)` is treated as public.

Usage

```
dpgmean(x, lower, upper, epsilon, delta = 1e-06, budget = NULL)
```

Arguments

`x`	Numeric vector (no NAs).
`lower`, `upper`	Hard bounds on `x`. Values outside are clipped (with a warning); the bounds must be chosen without looking at the data.
`epsilon`, `delta`	Privacy parameters for the whole release. `delta` should be well below `1 / length(x)`.
`budget`	Optional `dpbudget()` to charge against.

Details

Each half is calibrated with the analytic Gaussian mechanism (Balle and Wang 2018), which holds for every $\epsilon > 0$; the classical bound $\sigma = \Delta\sqrt{2\ln(1.25/\delta)}/\epsilon$ holds only for $\epsilon \leq 1$ and is not used. Noise comes from the operating system's CSPRNG; `set.seed()` has no effect.

Value

A noised mean (single numeric).

References

Balle and Wang (2018), Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising, ICML.

Examples

```
x <- runif(1000, 0, 1)
dpgmean(x, lower = 0, upper = 1, epsilon = 1.0)
mean(x) # the true mean, for comparison

# Wider bounds raise sensitivity, so the same epsilon adds more noise.
dpgmean(x, lower = -5, upper = 5, epsilon = 1.0)
```

---

dplapcnt *Differentially-private count via the discrete Laplace mechanism*

Description

Releases a count with discrete Laplace noise, $P(y) \propto e^{-\epsilon|y|}$, so the release is a whole number. Use it for the number of records matching a predicate (for example incidents in a division-year). Adding or removing one record changes the count by at most 1, and the release is pure ϵ-DP under that add-or-remove-one relation, the one every mechanism in this package uses.

Usage

```
dplapcnt(truecnt, epsilon, budget = NULL)
```

Arguments

`truecnt`	Non-negative integer, at most 2^{52}; the true count. Up to that bound the count plus its noise is exact in a double unless the noise exceeds 2^{52} in magnitude, which has probability below $e^{-2^{32}}$ at the smallest `epsilon`.
`epsilon`	Privacy loss (smaller means more noise, stronger privacy).
`budget`	Optional `dpbudget()` to charge `epsilon` against.

Details

The noise is drawn from the operating system's CSPRNG with the exact sampler of Canonne, Kamath and Steinke (2020), in integer arithmetic, so no floating-point artefact separates neighbouring datasets and `set.seed()` has no effect. `epsilon` is used as a rational with denominator 2^{20}, rounded down.

Value

The noised count, a whole number (it can be negative; clip with `pmax(0, x)` for display, which is post-processing and costs no privacy).

References

Canonne, Kamath and Steinke (2020), The Discrete Gaussian for Differential Privacy, NeurIPS. Mironov (2012), On Significance of the Least Significant Bits for Differential Privacy, CCS.

Examples

```
# A single noised release of a true count of 42 (different every call).
dplapcnt(truecnt = 42, epsilon = 1.0)

# Smaller epsilon = stronger privacy = more noise.
dplapcnt(42, epsilon = 0.1)
dplapcnt(42, epsilon = 5.0)

# The mechanism is unbiased: averaging many releases returns about the truth
# (for illustration; releasing many answers spends many epsilons).
mean(replicate(500, dplapcnt(42, epsilon = 1.0)))
```

---

dplaphist *Differentially-private histogram via the discrete Laplace mechanism*

Description

Adds independent discrete Laplace noise, $P(y) \propto e^{-\epsilon|y|}$, to each bin count. Adding or removing one record changes exactly one bin by 1, so the whole histogram is pure ϵ-DP under the add-or-remove-one relation. Noise and sampler as in `dplapcnt()`.

Usage

```
dplaphist(counts, epsilon, budget = NULL)
```

Arguments

`counts`	Vector of non-negative whole-number bin counts, each at most 2^{52} (see `dplapcnt()`).
`epsilon`	Privacy loss for the whole histogram (positive scalar).
`budget`	Optional `dpbudget()` to charge `epsilon` against.

Value

A numeric vector of whole numbers, the same length as `counts` (values may be negative; clip for display).

Examples

```
true <- c(120, 45, 8, 230, 17)
dplaphist(true, epsilon = 0.5)

# Post-process for display: clip negatives (costs no privacy).
pmax(0, dplaphist(true, epsilon = 1.0))

# Release a private histogram straight from tabulated data.
counts <- as.integer(table(complaint_sample$year))
dplaphist(counts, epsilon = 1.0)
```

---

dpspent *What a privacy budget has spent*

Description

What a privacy budget has spent

Usage

```
dpspent(budget)
```

Arguments

budget	A budget from dpbudget().

Value

A list: epsilon, delta (the totals), spent_epsilon, spent_delta, remaining_epsilon, remaining_delta, releases.

Examples

```
b <- dpbudget(1)
invisible(dplaphist(c(10, 20), epsilon = 0.25, budget = b))
dpspent(b)$remaining_epsilon
```

hostcat *Curated datasets at data.rmorie.com*

Description

Beyond the open data this package ships, the MORIE project keeps databases materialised from Google BigQuery public datasets (Chicago crime, EPA air quality, US census, FEC, FDA, NOAA, NHTSA, Hacker News, Ethereum, World Bank, ...) and serves their tables from the edge. They need no sign-in and no key; a key that an earlier version of this or a related package stored is not read and never sent. `hostcat()` returns every table with its key, description, rows and BigQuery source (the manifest is kept for a day); tables whose source terms do not allow redistribution (ICPSR studies, among them the NCVS 38604 extract, and any table the manifest marks restricted) are not served: they are left out, with a message counting them, and `hostload()` refuses them before any request. A table is served only when its `access` mark, if any, is `"open"` or `"public"` and its `restricted` mark, if any, is `false`; a mark of any other value or shape counts as restricted. `hostload()` checks each table against the SHA-256 the manifest lists for it (`sha256_gz`, `sha256` or `sha`) and refuses a file that does not match; a table listed without any digest is used with a warning, once per session. `hostload("db/table")` returns one table, cached under the package cache directory (`tempdir()` unless `options(rmoriedata.cache_dir = )` names a persistent one).

Usage

```
hostcat(refresh = FALSE)

hostload(key, refresh = FALSE)
```

Arguments

`refresh`	Fetch again even when a day-old copy is cached.
`key`	A `db/table` key from the catalog: segments of ASCII letters, digits, `_`, `-` and `.`, none starting with a dot, joined by single slashes. Any other key is refused before anything is requested, and a manifest entry whose key is not of this form is not served.

Details

`hostload()` returns the table as the file holds it, as the bundled tables are read: the column names as written (as the manifest lists them); an empty field is missing in a logical or number column and `""` in a text column (the text `"NA"` is a value). A column is logical, integer or double only when every value in it is written the way R writes such a value: `TRUE` or `FALSE`; whole numbers without a leading zero or a plus sign; for double, numbers without a leading zero, whole ones of at most 15 digits (which a double holds exactly) and decimal ones of at most 17 significant digits (zeros after the decimal point are formatting, and `"6.0"` is 6), or in R's own exponent form (`1e-05`). Any other column is text, so a code such as `"046"` keeps its leading zero, and `"+3"`, `".5"`, `"0x1A"`, `"Inf"` or an identifier of 16 or more digits keep every character. A column with no value at all is logical `NA`. Text is marked UTF-8, whatever the session's locale; bytes that are not valid UTF-8 are marked `"bytes"`, and a NUL byte, which no R string holds, is dropped. Quoted fields may hold commas,

quotes and line breaks (a Windows line end inside a field reads as `"\n"`); Windows line ends and a byte-order mark are allowed. A record with more or fewer fields than the header is a damaged download: the copy is deleted and the call fails, so the next call fetches it again. A table's file is its gzip members, each one read: NUL bytes after the last member are ignored, as `gzip -d` ignores them, and any other bytes after it are refused, never read as rows. A cached copy is deleted only when it does not match the manifest's digest or is damaged itself; a table that cannot be read for another reason (memory that runs short, bytes after its gzip data, a session option) stops with the copy left in place. Each table is cached in a file of its own under `hosted/tables` in the cache folder, named from its key so that no two keys share one (two that differ only in case included); tables that versions before 0.4.3 cached under `hosted/` are not read, and are fetched again once.

The manifest is fetched again once the cached copy is a day old. When that fails (offline, or the service is down) the cached copy is used, with a message, so a table already cached in a persistent `rmoriedata.cache_dir` still loads offline; every table is still checked against the digest the manifest lists, and one that does not match is not used. With `refresh = TRUE` the manifest must be fetched: when it cannot be, `hostcat()` and `hostload()` stop with an error instead of using the cached copy.

R sessions loading the same table take turns: while one downloads it, or checks its cached copy, the others wait. A session whose wait sees the download not grow for `options(rmoriedata.lockwait = )` seconds (600 by default) stops with an error saying the other session may be stalled.

Value

`hostcat()`: a data frame with `key`, `name`, `rows` (`NA` where the manifest gives no whole count) and `source`; `hostload()`: the table as a data frame.

Examples

```
## Not run:
head(hostcat())
df <- hostload("chicago_crime/incidents")

## End(Not run)
```

---

kanonvrf *k-anonymity verification*

Description

Checks whether a data.frame satisfies k-anonymity over the supplied quasi-identifier columns. A dataset is k-anonymous if every combination of quasi-identifier values appears in at least k rows. A data frame with no rows is refused: every class of an empty table meets any threshold vacuously, which says nothing about a release (it was reported as "VIOLATED ... 0/0 classes below threshold").

Usage

```
kanonvrf(data, qids, k = 5)
```

Arguments

data	data.frame.
qids	Character vector of column names.
k	Minimum equivalence-class size. Default 5 (a common public-health / open-data threshold).

Details

A missing value (NA) in a quasi-identifier is treated as a value of its own: rows with an unrecorded ward, age band or race form their own classes and are counted, never dropped. Unrecorded values are often the rarest records, which makes them the most re-identifiable, so they must meet the threshold like any other class. Every row is accounted for: the class sizes sum to nrow(data). NaN counts as missing, in the same class as NA.

Values are compared exactly. Two numbers are in one class only when they are the same number in storage, whatever the column's class (plain double, Date, POSIXct, difftime, I() and other classed numbers alike), so 0.1 + 0.2 and 0.3, which differ in the last bit, are two classes, and so are two times a microsecond apart; -0 and 0 are one. A complex number is compared by both parts. Integer, logical, character and raw columns are compared by value, a factor by its level label and a POSIXlt by the instant it names. A factor level that is itself NA is missing, and the string "NA" is a value, not a missing one. A column that cannot be compared exactly is refused with an error naming it: a bit64::integer64 column (convert it with as.character()), a list column and a matrix column.

Value

A list with class "morie_k_anon" containing:

satisfies logical, whether the dataset is k-anonymous.

k the threshold used.

min_class_size integer, size of the smallest class.

n_classes integer, total number of equivalence classes.

n_violations integer, number of classes below the threshold.

violating_classes data.frame of class keys plus their .n sizes (empty data.frame when none).

summary human-readable one-line summary.

Examples

```
df <- data.frame(
  age = c(25, 25, 25, 32, 32, 40),
  sex = c("F", "F", "F", "M", "M", "M")
)

# k = 2: the class {age=40, sex=M} has only 1 row -> VIOLATED.
res <- kanonvrf(df, c("age", "sex"), k = 2)
res$summary
res$satisfies
res$violating_classes # the offending quasi-identifier combos
```

```
# Loosening to k = 1 always holds; the default k = 5 is stricter.
kanonvrf(df, c("age", "sex"), k = 1)$satisfies
kanonvrf(df, c("age", "sex"))$satisfies # k = 5

# A single quasi-identifier is fine too.
kanonvrf(df, "sex", k = 3)$min_class_size

# On real bundled data: are (year, arrest) cells 5-anonymous?
kanonvrf(complaint_sample,
  c("year", "arrest"),
  k = 5
)$summary
```

`ldivvrf` *l-diversity verification*

Description

Checks whether a data.frame satisfies l-diversity: within each equivalence class defined by the quasi-identifiers, the sensitive attribute must take at least `l` distinct values. A data frame with no rows is refused, as by `kanonvrf()`.

Usage

```
ldivvrf(data, qids, sensitive, l = 3)
```

Arguments

`data`	data.frame.
`qids`	Character vector of QI column names.
`sensitive`	Name of the sensitive-attribute column.
`l`	Minimum number of distinct sensitive values per class. Default 3.

Details

Only known (non-`NA`) sensitive values count towards diversity: a class whose rows read `"HIV"`, `NA`, `NA` tells an adversary the attribute is HIV or withheld, which is diversity 1. Classes are formed as in `kanonvrf()`, with `NA` in a quasi-identifier as a level of its own, and sensitive values are compared exactly in the same way (a factor level that is itself `NA` is withheld, not a value).

Value

A list with class `"morie_l_div"` containing:

`satisfies` logical.

`l` the threshold used.

`min_diversity` integer, lowest per-class distinct count.

`n_classes` integer.

`n_violations` integer, classes below the threshold.

`violating_classes` data.frame of class keys plus their `.diversity` count.

`summary` human-readable.

Examples

```
df <- data.frame(
  age = c(25, 25, 25, 25, 32, 32, 32),
  sex = c("F", "F", "F", "F", "M", "M", "M"),
  dx  = c("A", "B", "C", "A", "X", "Y", "Z")
)

# Class {25,F} has 3 distinct dx (A,B,C); {32,M} has 3 (X,Y,Z) -> l=3 holds.
res <- ldivvrf(df, c("age", "sex"), "dx", l = 3)
res$summary
res$satisfies
res$min_diversity

# Demanding l = 4 fails: no class has 4 distinct sensitive values.
bad <- ldivvrf(df, c("age", "sex"), "dx", l = 4)
bad$satisfies
bad$violating_classes

# k-anonymity and l-diversity are complementary: check both.
kanonvrf(df, c("age", "sex"), k = 3)$satisfies
```

`mdask` *Ask a language model about the bundled datasets*

Description

Sends a question through `rmoriebricklayer::llmask()`, to a language model you bring: an OpenAI-compatible endpoint of your own, a provider whose key is set (OpenAI, Anthropic, Google Gemini, OpenRouter, Groq, Mistral and more: `rmoriebricklayer::llmprvd()`), or an Ollama server. Nothing is sent anywhere you did not set up. An Ollama server is asked only when you chose one: `route = "ollama"`, an `OLLAMA_HOST`, `OLLAMA_BASE_URL`, `OLLAMA_MODEL` or `OLLAMA_API_KEY` environment variable, or an `ollama.*` setting or `route = "ollama"` saved with `mdllmconf()`. Without any of these, an Ollama server that happens to run on this machine (at `localhost:11434`) is neither probed nor sent the question. The model is given every function of rmoriedata and rmoriebricklayer and tools to read their help, and this package's catalogue of datasets, so it can name the tables and write the code that loads them. `rmoriebricklayer::llmstat()` shows which route would answer; `mdllmconf()` (or `rmbl config setup` from the shell) saves a provider's key, the route and the model.

Usage

```
mdask(question, model = NULL, route = NULL)
```

Arguments

question	Character scalar.
model	Optional model name; the route's model when NULL.
route	NULL or "auto" (the first route that is set up), "own", "ollama" or a provider id such as "anthropic" (that route only).

Value

Character scalar: the answer, or a sentence saying what went wrong or what to set up (never an error, so a script keeps going).

Examples

```
## Not run:
# These need a language model: a provider's key, Ollama, or your own server.
Sys.setenv(ANTHROPIC_API_KEY = "...")
mdask("which bundled datasets cover Toronto police use-of-force?")
mdask("summarise the SIU director's-report corpus", route = "ollama",
    model = "qwen3:8b")

## End(Not run)

# With no route set up the call returns a setup hint, not an error, and
# reaches no network:
if ("llmstat" %in% getNamespaceExports("rmoriebricklayer")) {
  old <- Sys.getenv(c("OLLAMA_HOST", "MORIE_LLM_BASE_URL"), unset = NA)
  Sys.setenv(OLLAMA_HOST = "off", MORIE_LLM_BASE_URL = "off")
  if (!nrow(subset(rmoriebricklayer::llmstat(),
                   status %in% c("key set", "configured")))) {
    cat(mdask("hello"))
  }
  for (v in names(old)) {
    if (is.na(old[[v]])) Sys.unsetenv(v) else do.call(Sys.setenv, as.list(old[v]))
  }
}
```

---

mdcat *Catalog of the bundled datasets*

Description

One row per bundled table or dictionary: slug, source_path (relative to the package's extdata directory), kind, and for tables n_rows, n_cols and parquet_path (the same table as a Parquet file, see mdpath()).

Usage

```
mdcat()
```

Value

A data frame: `slug`, `source_path`, `kind` (table, dictionary, ...), `n_rows`, `n_cols`, `parquet_path`, and `synthetic`, TRUE for the tables that are generated stand-ins with the published schema rather than published data (the ARSAU use-of-force tables and the `*_synthetic` ones).

Examples

```
cat <- mdcat()
cat$slug[cat$synthetic]
tbls <- cat[cat$kind == "table", c("slug", "n_rows", "n_cols")]
head(tbls[order(-tbls$n_rows), ])
```

`mddict` *Data dictionary for a bundled dataset*

Description

Data dictionary for a bundled dataset

Usage

```
mddict(slug)
```

Arguments

`slug`	Dictionary slug; rows with `kind == "dictionary"` in `mdcat()` list them.

Value

The dictionary as JSON text (a length-one character vector), or `NULL` invisibly with a message when none is bundled.

Examples

```
d <- mddict("arsau_2023_dictionary")
substr(d, 1, 60)
```

mdllmconf *Show or save the language-model settings*

Description

The route `mdask()` takes and the address, key and model of each route (your own OpenAI-compatible server, each provider such as Anthropic or OpenAI, a local or LAN Ollama server), shared with **rmoriebricklayer** and the packages built on it. A thin front to `rmoriebricklayer::llmcfg()`, which saves them in `~/.config/morie/llm.json` only when you pass a setting; from the shell, `rmbl config` and `rmbl config setup` do the same.

Usage

```
mdllmconf(...)
```

Arguments

`...`	Settings as `key = value`, e.g. `route = "anthropic"`, `anthropic.key = "..."`, `anthropic.model = "claude-opus-5-5"`, `ollama.url = "http://192.168.1.20:11434"`, `own.url = "http://localhost:1234/v1"`; `NULL` removes one. With no arguments nothing is written.

Value

A data frame with one row per setting (`key`, `value`, `source`, `env`, `help`).

Examples

```
if ("llmcfg" %in% getNamespaceExports("rmoriebricklayer")) {
  mdllmconf()   # reading writes nothing
}
## Not run:
mdllmconf(route = "anthropic", anthropic.model = "claude-opus-5-5")
mdllmconf(route = NULL)   # back to the automatic order

## End(Not run)
```

mdload *Load a bundled dataset by slug*

Description

Reads the table's CSV and applies the column names and classes from the bundled schema, so the result is the same typed data frame on every platform regardless of how `read.csv()` would have guessed. Every table also ships as a Parquet file holding the same typed frame; `format = "parquet"` reads that copy instead. The first load of a table is cached for the session; later calls return the cached copy unless `refresh = TRUE`. The cached copy is what the file held when it was checked; when `mdverify()` finds the file changed since, the cached copy is dropped, and the next load reads the file again and refuses it.

Usage

```
mdload(slug, refresh = FALSE, format = c("csv", "parquet"))
```

Arguments

`slug`	Dataset slug; see the `slug` column of `mdcat()`.
`refresh`	Re-read the file, and check it against the signed manifest again, even if a cached copy exists.
`format`	`"csv"` (default) or `"parquet"`: which shipped copy to read. Both give the same data frame, value for value: decimal text in the CSV is read as the nearest double, the same on every platform.

Value

A data frame.

Examples

```
d <- mdload("arsau_2023_uof_main_records")
str(d[, 1:4])
```

`mdpath` *Path of a bundled table's shipped file*

Description

The absolute path of the CSV or Parquet copy of a table, verified against the signed manifest first. The Parquet path is the bridge to Python: `pandas.read_parquet(path)` gives the same typed table `mdload()` returns.

Usage

```
mdpath(slug, format = c("parquet", "csv"))
```

Arguments

slug	Dataset slug; see the slug column of mdcat().
format	"parquet" (default) or "csv". A dictionary or other non-table slug has one file, which is returned either way.

Details

Every call checks the file again: its SHA-256 is computed and compared with the manifest's (the store's files are small; the largest takes a few tens of milliseconds), so a file changed since an earlier call is refused, whatever else about it stayed the same. What is not covered: a change made after the path is returned, before it is read. mdload() has no such gap: it reads a file's bytes once, checks them and reads the table from them.

Value

A length-1 character path.

Examples

```
p <- mdpath("arsau_2023_uof_main_records")
file.exists(p)
basename(mdpath("arsau_2023_uof_main_records", "csv"))
```

---

mdsums *SHA256 checksums of bundled rmoriedata files*

Description

Computes the SHA256 digest of every file rmoriedata bundles in inst/extdata, each file read once and only when it is a regular file (NA for any other path, and for a file larger than 1 GiB), and compares each with the digest the signed manifest records for it (in_manifest). For the full check, signature included, use mdverify(); this function is the per-file record an analysis can pin and compare later.

Usage

```
mdsums()
```

Value

A data frame with one row per bundled file, in the byte order of path (the same in every locale), and columns path (relative to the extdata root, forward slashes, unique), file (the bare basename), bytes, sha256, and in_manifest (TRUE when the signed manifest lists the file with this digest; NA for the manifest's own files). Use path to locate a file; several basenames recur in more than one directory.

Examples

```
# One row per bundled file: name, size in bytes, SHA256 digest.
ck <- mdsums()
str(ck)
head(ck)

# Total bundled payload and the largest few files.
sum(ck$bytes)
head(ck[order(-ck$bytes), c("file", "bytes")], 3)

# Provenance workflow: pin the digest of a file you depend on, then
# assert it hasn't changed under you in a later session / reinstall.
if (nrow(ck)) {
  pinned <- ck$sha256[1]
  again <- mdsums()
  stopifnot(again$sha256[again$path == ck$path[1]] == pinned)
}
```

`mdverify` *Verify the bundled data store against its signed manifest*

Description

Every file rmoriedata ships is listed with its SHA-256 in a manifest, and the manifest is signed with an XMSS (RFC 8391, SHA-256) key whose public half ships with the package. Loading a table checks its file against the manifest; this function checks all of them at once. What the session has cached from a file found changed or missing (a table loaded by `mdload()`, the catalog, the schema) is dropped, so the next load reads the file again and refuses it.

Usage

```
mdverify()
```

Details

A file present in the installed store but absent from the manifest is reported too, hidden (dot) files included, as a row with `listed` = `FALSE` and `ok` = `FALSE`: a file added to the store is as much a change as a file altered in it.

What is covered: every file under the package's `extdata` directory (or the directory named by `options(rmoriedata.store = )`), which holds every table, dictionary, catalog and Parquet copy that `mdload()`, `mdpath()`, `mddict()`, `siuload()` and `cihitbls()` read. What is not covered: the package's R code, its compiled code, and the two lazy-data objects complaint_sample and arrest_sample (stored in the package's `data` database, not in `extdata`); these are installed files of the package itself, which `tools::checkMD5sums("rmoriedata")` checks against the `MD5` file of a CRAN install.

A listed path that holds anything but the file (it is missing, a directory, a FIFO, a device, a link that loops or leaves the store, or cannot be read) is a row with `ok` = `FALSE`; nothing at such a path is waited on.

Value

A data frame with one row per manifest entry, then one per unlisted file (in the byte order of their paths, the same in every locale): `path`, `bytes`, `sha256` (the manifest digest, or the file's own for an unlisted file: `NA` when it is not a regular file or is larger than 1 GiB), `listed`, `ok` (the file on disk is listed and matches). The attribute `"signature"` is `TRUE` when the manifest's signature verified, and the function errors if it did not.

Examples

```
v <- mdverify()
all(v$ok)
attr(v, "signature")
```

`morie_core` *Shared C-core helpers*

Description

Thin access to the compiled core that ships in **rmoriebricklayer**. rmoriedata links that core via `LinkingTo: rmoriebricklayer`, so these functions call the exact same kernels that package exports – no duplicated C code. They back fast data-integrity hashing and summaries for the bundled datasets.

Usage

```
mdsha(x)

mdmean(x)
```

Arguments

`x`	For `mdsha()`, a length-1 character vector or a raw vector. For `mdmean()`, a numeric vector (coerced with `as.numeric()`); NA/NaN propagate.

Value

`mdsha()` returns a 64-character lowercase hex digest. `mdmean()` returns a length-1 numeric.

Examples

```
## ---- mdsha(): 64-char lowercase hex digest --------------
mdsha("abc") # hash a character scalar
mdsha("") # the empty string still hashes
mdsha(charToRaw("abc")) # identical digest from raw bytes

# character input and its raw-byte equivalent agree:
identical(mdsha("abc"), mdsha(charToRaw("abc")))
```

```
# Data-integrity pin: verify a value is byte-for-byte what you expect.
expected <- mdsha("record-42")
stopifnot(mdsha("record-42") == expected)

# Fingerprint a whole object by hashing its serialization.
mdsha(serialize(list(a = 1, b = "x"), NULL))

## ---- mdmean(): fast length-1 mean --------------------------
mdmean(1:10) # 5.5
mdmean(c(2, 4, 6)) # 4
mdmean(c(-1, 0, 1)) # 0
mdmean(c(1, 2, NA)) # NA propagates (no na.rm)
mdmean(complaint_sample$year) # mean over a bundled column
```

`rmoriedata-victoria` *Victorian crime statistics (Crime Statistics Agency Victoria)*

Description

Ten tables from the Crime Statistics Agency's "Latest Victorian crime data" release, bundled in the CSV store and reached by slug through `mdload()`. Each is Table 01 – the headline series – of the corresponding published workbook, for the year ending March 2026.

Details

`vic_criminal_incidents` Criminal incidents by offence division, subdivision and subgroup, with rate per 100,000.

`vic_recorded_offences` Recorded offences on the same offence hierarchy.

`vic_victim_reports` Victim reports by offence.

`vic_alleged_offender_incidents` Alleged offender incidents, including age and sex breakdowns.

`vic_family_incidents` Family incidents by category and outcome.

`vic_lga_criminal_incidents`, `vic_lga_victim_reports`, `vic_lga_family_incidents` The same measures by police region and Local Government Area.

`vic_indigenous_victim_reports`, `vic_indigenous_family_incidents` Aboriginal and/or Torres Strait Islander status breakdowns, as published.

The workbooks are .xlsx. They were read with a native .xlsx reader, with no **readxl** or **openxlsx** dependency. Rebuild with `data-raw/build_vic_tables.R`.

Counts are as published by the CSA and are subject to its own revisions: figures for a given year change between releases as incidents are reclassified, so a table bundled here is a snapshot of the March 2026 release, not a permanent record of that year.

Source

Crime Statistics Agency Victoria, "Latest Victorian crime data". https://www.crimestatistics.vic.gov.au/crime-statistics/latest-victorian-crime-data (the page answers only browsers; archived copy of 7 October 2026: https://web.archive.org/web/20261007154657/https://www.crimestatistics.vic.gov.au/crime-statistics/latest-victorian-crime-data). Released under CC BY 4.0.

See Also

mdcat(), mdload()

Examples

```
# Every bundled Victorian table, by slug.
cat <- mdcat()
cat[grepl("^vic_", cat$slug), c("slug", "n_rows", "n_cols")]

# Headline criminal-incident series.
ci <- mdload("vic_criminal_incidents")
str(ci)

# Incidents by offence division for the most recent year.
latest <- ci[ci$Year == max(ci$Year), ]
tapply(latest$`Incidents Recorded`, latest$`Offence Division`, sum)
```

---

siuload *Load the Ontario SIU director's-report corpus*

Description

Returns the bundled Ontario Special Investigations Unit (SIU) director's-report table: one row per report drid, 66 structured columns (police service, incident / notification / decision dates, investigator and witness / subject-official counts, affected-person demographics, injuries, legislation, charges verdict, director's decision, and news-release linkage), plus a `panel_reviewed` flag.

Usage

```
siuload(
  lang = c("all", "en", "fr"),
  as = c("data.frame", "tibble"),
  format = c("csv", "parquet")
)
```

Arguments

`lang`	One of `"all"` (default), `"en"`, or `"fr"`: filter to the English-only, French-only, or all rows.
`as`	Return format: `"data.frame"` (default) or `"tibble"`.
`format`	`"csv"` (the gzip CSV, default) or `"parquet"` (the same rows and columns, read from the typed store's Parquet copy and returned as text).

Details

For the English reports marked `panel_reviewed == "TRUE"`, the 16 key columns were verified by a multi-agent LLM review panel against the full report text and the parser's guess resolved to the correct value; the subject-official count is filled for every one of them (witness-officer-only investigations are a genuine 0). A French report carries its English report's reviewed case facts (the dates, the team, the counts, the affected person's age and sex, the charges and the director's view) and `panel_reviewed == "FALSE"`; its text fields stay French, and its `police_service` is its English report's service under the name the SIU's French pages use.

`charges_recommended` is `"FALSE"`, `"TRUE"` or empty here (every column is text); in the typed table, `mdload("siu_directors_reports")`, it is logical (`FALSE`, `TRUE`, or `NA` when the report does not say). `corrigenda` holds the SIU's correction notice of a report (a corrigendum is applied to the report it corrects, not counted as a report). `narrative_summary` is the opening of the report's own account of the incident (about 900 characters, at most 1,500, cut at a sentence end): its "Incident Narrative" / "Description de l'incident" section, or, in reports without one (2005-2016 and re-opened files), the "Notification of the SIU" / "Avis a l'UES" section, which holds the account there. It is empty on the 8 reports whose page has neither. `supplemental_materials` lists the legislation and case-law links a report cites.

`police_service` is the service of the subject officials – not the force that notified the SIU, which is often a custody, requesting or neighbouring service. Each correction to it is listed with the reason read from the report in `data-raw/siu_police_service_review.csv` of the source repository.

One row is one published report, keyed by `drid` and `_language`. A case number can appear on more than one row: the SIU publishes a further report when a case is reconsidered or reported again, each with its own `drid`. Some schema columns (for example the raw times of injury and notification, evidence types, and weapons used) are not stated in the published reports and are empty throughout; they are kept so the table's shape matches the parser's schema.

The table is the output of the SIU director's-report parser, the first open-source pipeline for this corpus, created by Vansh Singh Ruhela as part of the MORIE / MRM framework, regenerated over the full public corpus. The parser itself is in the rmorie package (`rmorie::morie_fetch_siu()` rebuilds the table live); nothing here needs it.

This loader returns the corpus as text: every column is character and an empty cell is `""`, which is the form the SIU parser writes and reads back. The same table is also in the typed data store: `mdload("siu_directors_reports")` applies the bundled schema (integer `drid` and counts, `NA` for empty cells), so counts of missing values differ between the two entry points by construction; pick the typed store for analysis and this loader for the parser round trip.

Value

A `data.frame` (or tibble) of SIU director's-report rows.

Source

Ontario Special Investigations Unit director's reports, https://www.siu.on.ca/en/directors_reports.php (post-2018) and the Ontario Government archive (pre-2018). Parsed with the MORIE SIU parser.

Examples

```
# Default: every parsed report, as a base data.frame.
all <- siuload()
nrow(all)
ncol(all)

# `lang` filters the corpus by report language.
en <- siuload(lang = "en") # English director's reports
fr <- siuload(lang = "fr") # French director's reports
nrow(en)
nrow(fr)

# `as = "tibble"` returns a tibble when the tibble package is present.
if (requireNamespace("tibble", quietly = TRUE)) {
  tb <- siuload(lang = "en", as = "tibble")
  class(tb)
}

# The five police services with the most reports.
if (nrow(en)) {
  top <- sort(table(en$police_service), decreasing = TRUE)
  head(top, 5)
}
```

Index

∗ **datasets**
    arrest_sample, 3
    complaint_sample, 10
    rmoriedata-victoria, 29

arrest_sample, 3, *27*
as.numeric(), *28*

cellsupp, 4
chiclr, 5, *6*
chiload, *3*, 6, *11*
cihiget, 8, *10*
cihitbls, 9
cihitbls(), *27*
complaint_sample, 10, *27*

dpbudget, 12
dpbudget(), *13–16*
dpgmean, 13
dplapcnt, 14
dplapcnt(), *15*
dplaphist, 15
dpspent, 16
dpspent(), *13*

hostcat, 17
hostload *(*hostcat*)*, 17

kanonvrf, 18
kanonvrf(), *20*

ldivvrf, 20

mdask, 21, *24*
mdcat, 22
mdcat(), *23*, *25, 26*, *30*
mddict, 23
mddict(), *27*
mdllmconf, *21*, 24
mdload, 24
mdload(), *25–27*, *29, 30*
mdmean *(*morie_core*)*, 28
mdpath, 25
mdpath(), *22*, *27*
mdsha *(*morie_core*)*, 28
mdsums, 26
mdverify, 27
mdverify(), *25, 26*
morie_core, 28

parallel::mclapply(), *12*

readRDS(), *12*
rmoriedata-victoria, 29

saveRDS(), *12*
serialize(), *12*
siuload, 30
siuload(), *27*

unserialize(), *12*