Package 'masque'

November 18, 2026

Title Structurally Faithful Development Surrogates for Tabular Data

Version 0.16.0

Description Turns a confidential tabular dataset - a single table, a folder of files, or a multi-sheet workbook - into a structurally faithful synthetic clone suitable for pipeline development. Experimental-design allocation and the NA pattern are preserved exactly; categorical design, treatment, and covariate vocabularies can be reversibly aliased; outcome and numeric-covariate values are re-simulated via a Gaussian copula that preserves the global covariance structure. Columns shared across tables are aliased consistently so the synthetic tables still join. A private recipe object round-trips a pipeline written against the synthetic clone onto the original data. Not a differential-privacy or anonymisation tool: outputs are development surrogates, not public-release-safe artefacts.

License MIT + file LICENSE

Encoding UTF-8

Language en-AU

Depends R (>= 4.2.0)

Imports cli, data.table (>= 1.15.0), digest, Matrix, S7, stats, utils, withr

Suggests agridat, callr, fst, ggplot2, knitr, maps, readxl, rmarkdown, spelling, testthat (>= 3.0.0), writexl, yaml

VignetteBuilder knitr

Config/testthat/edition 3

URL https://max578.github.io/masque/, https://github.com/max578/masque

BugReports https://github.com/max578/masque/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Max Moldovan [aut, cre] (ORCID: <https://orcid.org/0000-0001-9680-8474>, affiliation: Adelaide University)

Maintainer Max Moldovan <max.moldovan@gmail.com>

Repository CRAN

Date/Publication 2026-11-18 21:20:07 UTC

Contents

apply_recipe 2
audit_mask 4
clean_table 6
conform_table 7
data_card 10
detect_design 12
jitter_coordinates 14
mask 17
mask_set 22
masque 24
plot_design_summary 27
propose_roles 28
read_recipe 31
read_set 32
recipe 33
reveal_maps 34
roles 35
roles_validate 36
role_options 37
save_recipe 38
set_role 39
synthesise_geospatial 40
synthetic 42
unmask 43
write_set 44

Index **46**

`apply_recipe` *Translate a data frame into the synthetic namespace*

Description

Forward translation: takes an `original`-namespace data frame and returns it renamed and relabelled to match the synthetic namespace produced by `mask()`. Use this to run a pipeline (trained on the synthetic) against the original data without modifying pipeline code.

Usage

```
apply_recipe(original, rec, check_integrity = TRUE)
```

Arguments

original	A data frame in the original namespace.
rec	A masque_recipe object (e.g. from recipe(m)).
check_integrity	Logical. When TRUE (default), verifies that the NA mask of original matches the recipe's recorded integrity_fp. Mismatches error with guidance. Pass FALSE to bypass when the missingness has legitimately changed since the recipe was built.

Details

Operations applied (in order):

1. **Re-apply cleaning and conform decisions**: the legalised names and trimmed labels from clean_table(), then the label merges and type changes from conform_table() when the clone was built on a conformance object.
2. **Verify integrity** by comparing the NA mask of original to the SHA-256 fingerprint stored on the recipe (controlled by check_integrity).
3. **Drop** every column whose resolved action was "drop" on the roles table mask() masked with, in either mode. A linked join key of a set is aliased in the clone and is kept.
4. **Subset and reorder** to the columns the recipe knows about.
5. **Re-label factors / characters** for any column with a level map held by the recipe (i.e., treatment and categorical covariates in collaborate mode; or treatment in local mode with opt-in permutation). Unknown non-NA values fail closed.
6. **Rename columns** per recipe@column_name_map when mask() was called with alias_names (otherwise NULL, and names are unchanged).

Numeric columns are passed through unchanged: the synthetic-namespace for numeric columns is the same as the original. NA cells in the input remain NA in the output (no synthesis is performed here).

Value

A data.table in the synthetic namespace, ready for the pipeline.

See Also

unmask(), mask().

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
r$role[r$col == "Species"] <- "covariate"
m <- mask(iris, r, mode = "collaborate", seed = 1)
rec <- recipe(m)
iris_in_synth_space <- apply_recipe(iris, rec)
head(iris_in_synth_space)
```

audit_mask *Audit a masque object for leakage and shareability risks*

Description

Returns a per-column audit data.table, with table-level checks attached, and prints a severity-grouped report. Auto-runs in `mode = "collaborate"` at `mask()` time and stores the result on the masque object (`m@audit`); for local-mode audits or explicit re-audits, pass the original data frame via `original`.

Usage

```
audit_mask(m, original = NULL, print = TRUE)
```

Arguments

`m`	A `masque` object from `mask()`, or a `masque_set` from `mask_set()`.
`original`	Optional. Required when `m@audit` is NULL (typically in local mode). Used to recompute exact-match-pct etc. on demand. The original data frame for a `masque`; for a `masque_set`, a named list holding the original table of every table in the set.
`print`	Logical; if TRUE (default), print a styled report.

Details

Each row of the returned data.table holds:

- `col`: column name in the original.
- `role`: assigned role.
- `kind`: storage kind.
- `leakage_class`: `low`, `medium`, or `high`.
- `n_unique_levels`: distinct non-NA values (categorical only).
- `freq_min`: minimum per-level frequency (categorical only).
- `exact_match_pct`: percentage of synthetic cells equal to the original cell (numeric only; cell-by-cell).
- `na_pct`: percentage of NA cells in the original column.
- `na_pattern_uniqueness`: fraction of rows in the original with a globally unique NA pattern (one number per data frame, repeated on every row).
- `alias_status`: `aliased`, `permuted`, `passthrough`, or `dropped`.
- `notes`: short human summary.
- `same_row_pct`: percentage of rows where the clone value equals the original value in the same row, read back through the level map (categorical, logical, date and two-valued numeric columns).

- `baseline_pct`: the same share expected by chance, under a random permutation of the original column: $100 \sum_k p_k^2$ over its level frequencies (within the conditioning strata for a scrambled numeric column of a conditional clone).
- `excess_pct`: `same_row_pct` - `baseline_pct`.

The table-level checks sit in `attr(audit, "joint")`, a data.table with one row per check (`check`, `cols`, `observed`, `baseline`, `excess`, `sd`, `leakage_class`, `notes`). Each baseline comes from independent permutations of the original content columns (outcomes, covariates, dates, kept text), drawn under a fixed internal seed that leaves the caller's random number stream untouched; design, treatment, id, aliased text and linked columns are held fixed:

- `same_row_joint`: rows whose every content column equals the original row.
- `whole_row`: clone rows equal to some original row on every retained column.
- `rare_cross`: in the cross of visible (not aliased) categorical covariates and outcomes, clone cells holding one row whose combination is also a single row in the original.
- `unit_sizes`: a note, never graded, when design units differ in size, since the multiset of sizes can link a clone to a public trial list.

Classification heuristics:

- Retained PII-pattern column -> `high`.
- Outcome with action `keep` in collaborate -> `high`, or `medium` when `mask()` was called with `allow_kept_outcome = TRUE`, which the recipe records.
- Treatment unaliased in collaborate -> `high`. A numeric treatment (a dose) is aliased like a categorical one.
- Categorical covariate with a frequency-1 level in collaborate -> `high`.
- Outcome with exact-match-pct > 1\
- Numeric covariate with exact-match-pct > 5\ `medium`.
- Two-valued numeric columns skip the two exact-match rules: a 0/1 column matches its own cell about half the time by chance. So do columns synthesised once per design unit (see `grain` in `mask()`) and whole-valued numeric covariates, which repeat real values by construction; all three are graded on `excess_pct` instead.
- Content column whose `excess_pct` is at least 10 points and four binomial standard deviations in collaborate -> `medium`.
- Joint match excess of at least 20 points and four permutation standard deviations -> `high` (5 points -> `medium`); rare-cross excess of at least 3 cells and four standard deviations -> `high` (1 cell and three -> `medium`). In collaborate mode these verdicts raise `masque_high_leakage` and block package-managed writers; in local mode they are reported in the notes and graded `low`.
- Ignore column retained in local -> `low` (informational).

A column constant within a design unit that was synthesised row by row (`grain = FALSE`, or a collaborate grain with fewer than five units) carries a note saying the clone does not keep that grain.

As a reference point, a collaborate-mode clone of `agridat::besag.met` (yield as the outcome, design columns kept) scores `low` on every column, and none of its yields equals the original value.

Value

The audit data.table, returned invisibly. For a `masque_set`, a named list of per-table audit data.tables, printed table by table.

See Also

`mask()`.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
r$role[r$col == "Species"] <- "covariate"
m <- mask(iris, r, mode = "collaborate", seed = 1)
audit_mask(m)
```

---

clean_table *Tidy a dirty table's column names and category labels before masking*

Description

Real custodian tables arrive with column names that are not valid R names (`"Yield (t/ha)"`, `"Site Name"`), leading or trailing whitespace in names and factor / character values, and the occasional near-duplicate label (`"north"` vs `"North"` vs `" north"`). `clean_table()` legalises names and trims whitespace, and reports each fix. Near-duplicate labels are only reported, since merging labels that look alike is a judgement call; `conform_table()` makes it.

Usage

```
clean_table(df, clean = c("auto", "report", "off"), quiet = FALSE)
```

Arguments

`df`	A data frame.
`clean`	One of `"auto"` (default - legalise names, trim whitespace, report near-duplicates), `"report"` (legalise names, report what whitespace / near-duplicate changes *would* be made but apply none), or `"off"` (legalise names only, skip all other hygiene). Column-name legalisation is applied in **every** mode – an invalid name silently rewritten downstream corrupts the clone – and is surfaced as a `masque_name_repaired` warning; only the whitespace and near-duplicate handling is governed by the mode.
`quiet`	Logical. When `FALSE` (default) a `cli` summary of the fixes and advisories is printed. Set `TRUE` to suppress it (the report object is returned either way).

Details

The corrections are returned alongside the cleaned data, so `mask()` records them in the recipe and `apply_recipe()` re-applies the same cleaning to the original.

Two input columns with the same name are refused with a `masque_duplicate_names_refusal` error: the recipe records renamed columns by their original name, so the two could not be told apart on the round trip. Rename them first.

Value

An object of class `masque_cleaning`: a list with

- `data` - the cleaned (or, under `report` / `off`, unchanged) data.table;
- `name_map` - named character `original -> clean` for every column whose name changed (empty if none);
- `level_fixes` - named list, one entry per column whose values were trimmed, each a named character `original -> clean`;
- `near_duplicates` - a data.table of report-only label pairs (`col`, `a`, `b`, `kind`) that look like typos but were left untouched. Labels that differ only inside a run of digits (`"G01"` and `"G02"`, `"Plot12"` and `"Plot13"`) are different codes and are not listed. The printed report shows the first 20 pairs and the count, and a recipe keeps the same;
- `mode` - the `clean` mode applied.

See Also

`mask()`, `propose_roles()`.

Examples

```
df <- data.frame(
  `Site Name` = c("north ", "north", "South"),
  `Yield (t/ha)` = c(3.1, 2.9, 5.0),
  check.names = FALSE
)
cl <- clean_table(df, quiet = TRUE)
names(cl$data)
cl$near_duplicates
```

---

conform_table *Bring a cleaned table to an analysable format, reporting every assumption*

Description

`clean_table()` makes the fixes that are unambiguously safe: it legalises column names and trims whitespace, and it *reports* near-duplicate labels without merging them. `conform_table()` is the next step and makes the two decisions that need judgement: deciding that `"NSW"` and `"Nsw"` are one category, and deciding that a character column holding six labels is a factor.

Usage

```
conform_table(
  df,
  merge_labels = c("report", "auto", "off"),
  types = c("report", "auto", "off"),
  max_levels = 20L,
  clean = c("auto", "report", "off"),
  quiet = FALSE
)
```

Arguments

df	A data frame.
merge_labels	One of "report" (default), "auto" or "off". Governs near-duplicate category labels, and treats the two kinds differently. A pair differing **only by capitalisation** is one category whatever happens, so it is merged onto the more frequent spelling, and on a tie onto whichever appeared first, which is deterministic and recorded. A pair differing **by one edit** might be two real categories, so it is merged only when one spelling is clearly the more common; on a tie it is refused and reported.
types	One of "report" (default), "auto" or "off". Governs storage format: a low-cardinality character column becomes a factor, a fully numeric character column becomes numeric, and a fully ISO-8601 character column becomes a Date.
max_levels	Cardinality at or below which a character column is treated as categorical. Default 20.
clean	Passed to clean_table(), which always runs first. Default "auto".
quiet	Logical. When FALSE (default) a cli summary is printed.

Details

Both are **off by default**: the default "report" mode lists every change it would make, in plain words, and applies none. Set a mode to "auto" to apply them; the assumption behind each one is recorded in the returned object.

Storage is decided **before** categories, so a column of numbers or dates held as text is typed first and "4.2" is never merged into "4.4". Only columns that remain categorical are considered for merging, and a one-character edit between labels shorter than four characters is ignored ("a" and "b" are one edit apart).

Two labels that differ by an edit are merged only when one spelling is clearly more common; a column is converted only when every value parses; numeric columns keep their storage. Anything not changed for these reasons is reported as a not applied row with the reason.

Merges follow chains: if "Nsw" merges onto "nsw" and "nsw" onto "NSW", both end on "NSW". A label whose merges end on two different labels ("colur" beside "color" and "colour" that occur equally often) is not merged; it is reported, and a masque_conform_ambiguous_merge warning is raised under merge_labels = "auto". Codes that differ only inside a run of digits ("G01" and "G02") are never treated as near-duplicates. Which columns hold categories is decided the same way whatever types is, so types = "off" still merges labels and leaves storage as it is.

Pass the result to `mask()` (or as a table of `mask_set()`): the recipe records the merges and type changes, and `apply_recipe()` replays them on the original so a pipeline built on the conformed clone sees the same labels and storage.

Value

An object of class `masque_conformance`: a list with

- `data` - the conformed data.table;
- `cleaning` - the `masque_cleaning` record from `clean_table()`;
- `merges` - one row per near-duplicate pair considered (`col`, `from`, `to`, `n_from`, `n_to`, `applied`, `reason`);
- `merge_map` - the merges as applied to the data (`col`, `from`, `to`), one row per merged-away label, each `to` a label left in the column. This is what `mask()` records and `apply_recipe()` replays;
- `types` - one row per column whose storage was considered (`col`, `from`, `to`, `applied`, `reason`);
- `assumptions` - every decision as a plain sentence (`col`, `assumption`, `applied`), which is the thing to read;
- `modes` - the modes applied;
- `max_levels` - the cardinality threshold used.

See Also

`clean_table()` for the safe fixes, `propose_roles()` for what a column is, `mask()` for the round-trip.

Examples

```
df <- data.frame(
  state = c("NSW", "Nsw", "NSW", "VIC", "VIC"),
  yield = c("3.1", "2.9", "5.0", "4.2", "3.8"),
  sown  = c("2024-05-01", "2024-05-03", "2024-05-01", "2024-05-08",
            "2024-05-08"),
  stringsAsFactors = FALSE
)
# report only: nothing is changed
cf <- conform_table(df, quiet = TRUE)
cf$assumptions

# apply the decisions
cf2 <- conform_table(df, merge_labels = "auto", types = "auto", quiet = TRUE)
str(cf2$data)
```

`data_card` *Write a data card for a masked clone*

Description

Describes a clone in a data card (card version 0.1): its size, its design columns and, for each column, a role, a kind and counts. An analysis planner can read the card in place of the data. No value of any column is written: a factor's levels are counted, never listed.

Usage

```
data_card(
  m,
  file = NULL,
  name = NULL,
  env = NULL,
  spatial = NULL,
  unit = NULL,
  primary_event = 1L
)
```

Arguments

`m`	A `masque` object from `mask()`, or a `masque_set` from `mask_set()`.
`file`	Where to write the card in YAML, or `NULL` (the default) to write nothing. For a set, a directory, which receives one file per table named `<table>.data_card.yaml`.
`name`	The table name on the card. Defaults to `"clone"` for a single table, and to the table names for a set (where `name` may give one name per table, in order). A name must start with a letter and hold only letters, digits, `_` and `.`. Any other name is refused with `masque_bad_card_name_refusal`, never cleaned.
`env`, `spatial`, `unit`	Optional names of clone columns: the environment columns (trials, sites, years), the columns giving a spatial or temporal position, and the one column identifying the unit of repeated measurement. Environment columns must have the role `design`. For a set, a named list giving the columns per table. A name that is not a column of the clone is refused with `masque_bad_card_column_refusal`.
`primary_event`	Which event pair of the first `events` spec is the card's event time and status, counted in the order the events occur. Ignored when the clone has no `events` spec.

Details

The card is computed from the clone alone: `synthetic()`, the role table from `roles()`, the mode, the package version and the column names of any `events` spec given to `mask()`. No other part of the private recipe is read, so the card can travel with the clone. On a clone whose column names were aliased, the card uses the aliases throughout.

Value

The card as a list, invisibly when `file` is given. For a set, a named list of cards, one per table.

Design block

The design block is copied from the role table, never re-detected. The treatment columns are those with the role `treatment`, and the block columns those with the role `design`, less any named in `env`, `spatial` or `unit`. Environment, spatial and unit columns are named by the caller, since the role table does not tell them apart from blocks. No design class is written. For a linked-event spec, the event pair chosen by `primary_event` gives the event time and status columns, the other status columns of that spec are listed as competing, and the card records the number of rows with the primary status equal to 1 and the share of rows with neither the primary nor a competing event (censored).

Provenance

Every statistic carries a provenance class.

- `structure`: exact in the clone, so equal to the same statistic on the original table. Row and column counts, missing counts and shares, the level counts and zero shares of kept and aliased columns and of event status columns, the treatment, block and event columns, the environment count and the event count.
- `clone_estimate`: re-simulated, so approximate. The level counts and zero shares of scrambled columns, including event times, and the censored share.
- `analyst`: named in this call: the environment, spatial and unit columns.

The YAML file

The card is written by a small deterministic writer: keys in the order of the contract, every string double-quoted and escaped, counts as integers, shares with six decimals, UTF-8 with `\n` line ends. The same clone gives the same bytes.

See Also

`roles()`, `synthetic()`, `mask()`, `mask_set()`.

Examples

```
r <- propose_roles(iris)
r <- set_role(r, "Species", role = "treatment", action = "alias")
r <- set_role(r, "Sepal.Length", role = "outcome")
m <- mask(iris, r, seed = 1)
card <- data_card(m)
card$table
card$design

path <- tempfile(fileext = ".yaml")
data_card(m, file = path)
head(readLines(path), 12)
```

`detect_design` *Detect environment scope and experimental-design structure*

Description

Inspects `df` and returns an S7 `design_summary`. Environment scope and experimental-design class are separate conclusions: a table can be a multi-environment trial (MET) even when its within-environment randomisation cannot be recovered from the recorded columns.

Usage

```
detect_design(
  df,
  roles = NULL,
  interactive = FALSE,
  threshold = 0.5,
  tie_delta = 0.02,
  env = NULL
)
```

Arguments

`df`	A data frame.
`roles`	Optional roles table (as returned by `propose_roles()`). When supplied, declared roles constrain candidate generation, and columns roled `treatment` define the treatment basis.
`interactive`	If `TRUE`, when the top-2 rule scores are within `tie_delta` the user is asked to choose between them via a cli menu. Default `FALSE`.
`threshold`	Minimum top-rule score for a class to be reported. Below this, `class_label` is `"none"`. Default `0.5`.
`tie_delta`	Score difference within which two rules are treated as tied. Default `0.02` – tight enough that 0.05-point score differences (the typical name-bonus / coverage gap) are decisive.
`env`	Environment specification. `NULL` performs conservative automatic resolution and leaves ambiguous or weak evidence unresolved. `FALSE` disables MET handling and runs the legacy whole-table path. A character vector names one or more columns whose interaction defines the environment.

Details

With `env = NULL`, exact environment names and a bounded set of site-year patterns are assessed conservatively. A site-only candidate auto-resolves only when treatments are replicated across sites. Weak or competing evidence gives an explicit uncertain result, never a guessed single trial. Supply `env` to define the environment basis, or use `env = FALSE` to run the pre-0.9 whole-table path exactly.

Before the scope step, the pooled legacy detector runs six independent design rules. Each returns a score in $[0, 1]$. The highest-scoring class above `threshold` is one of `"CRD"`, `"RCBD"`,

"IBD/alpha-lattice", "row-column", "split-plot", "factorial", or "none". Ties within tie_delta favour the simpler design. For a MET, per-environment classes, treatment connectivity, and near-disjoint experiment groups are diagnostics only. Dense connectivity calculations that exceed the package safety bound are reported as not_computed. None of these diagnostics proves the original randomisation protocol.

When scope resolves, block_cols is the block basis the environments agree on, or empty when they disagree; an environment column is never a block. evidence$block_revision records the change.

For a column without a treatment-pattern name, a treatment hint from the rules is auto-applied only when the column is crossed with a design column. A column with one value per level of a design column, or with an outcome-pattern name, is a covariate and never a candidate.

The detector never edits df. Its job is to recommend a role assignment, surface the evidence, and (optionally) draw a sanity check via plot().

Value

An S7 design_summary object. Legacy design fields include class_label, treatment_col, block_cols, whole_plot_col, sub_plot_col, spatial_cols, scores, evidence, recommended_roles, candidates, and warnings. Scope fields include scope_label, scope_status, scope_confidence, is_met, env_cols, env_method, n_env, group_cols, connectivity, per_env, and within_design_label.

See Also

propose_roles() for the role table that feeds detection and plot_design_summary() for the sanity-check visualisation.

Examples

```
# Classic alpha-lattice (24 genotypes, 3 reps, 6 blocks per rep).
if (requireNamespace("agridat", quietly = TRUE)) {
  d <- agridat::john.alpha
  ds <- detect_design(d)
  print(ds)
}

# Observational data frame -> class_label "none".
detect_design(mtcars)

# Explicit two-environment trial.
met <- expand.grid(
  env = factor(c("E1", "E2")),
  rep = factor(seq_len(2L)),
  gen = factor(c("G1", "G2", "G3"))
)
met$yield <- seq_len(nrow(met))
met_design <- detect_design(met, env = "env")
met_design@scope_label
met_design@connectivity$status
```

`jitter_coordinates` *Coarsen geographic coordinates by an on-land privacy jitter*

Description

Displaces latitude / longitude pairs by a small random distance, in place, so a synthetic table can carry realistic coordinates without revealing the true location of a field, farm or site. Two geographic-masking schemes are offered. The default `"donut"` displaces every point by a distance drawn uniformly (by area) from an annulus between `min_km` and `max_km`, in a random direction: this guarantees a minimum displacement (a privacy floor – no point is left almost where it started) while bounding the maximum drift. The `"gaussian"` scheme adds independent normal noise of standard deviation `sd_km` to each axis; it is simpler but has an unbounded tail, so a handful of points can move much further than intended.

Usage

```
jitter_coordinates(
  df,
  lat_col,
  lon_col,
  by = NULL,
  method = c("donut", "gaussian"),
  min_km = 5,
  max_km = 20,
  sd_km = 10,
  on_land = TRUE,
  max_tries = 100L,
  seed = NULL,
  .group = NULL
)
```

Arguments

`df`	A data frame containing the coordinate columns.
`lat_col`, `lon_col`	Column names of the latitude and longitude (numeric, in decimal degrees).
`by`	How rows are grouped into sites. `NULL` (default) groups rows that share an identical input coordinate pair. A character vector names one or more columns of `df` whose values identify a site. `FALSE` displaces every row independently (the pre-0.10.0 behaviour). See the section above.
`method`	`"donut"` (default) or `"gaussian"`.
`min_km`, `max_km`	Inner and outer radii of the donut, in kilometres (used when `method = "donut"`). Every site moves at least `min_km` and at most `max_km`.
`sd_km`	Per-axis standard deviation in kilometres (used when `method = "gaussian"`).

on_land	Controls the land constraint. TRUE (default) rejects any displacement that lands in the sea, using maps::map.where() (requires the maps package). A function function(lon, lat) returning a logical vector supplies your own test (for example an sf-based high-resolution coastline). FALSE applies no constraint.
max_tries	Maximum re-draws per site before giving up. Default 100. A site that cannot be placed on land is set to NA on both axes, with a warning: masque never emits the true coordinate as though it were masked.
seed	Optional integer seed for reproducibility. The caller's RNG state is preserved. Several published jitters of the same point average toward it: keep the seed (a mask() recipe records it) and do not publish more than one jittered release of the same sites.
.group	Internal. A pre-computed grouping vector of length nrow(df), used by mask() to group by the *original* table's site structure. Takes precedence over by. Not for direct use.

Details

The displacement is computed in kilometres and converted to degrees with a cos(latitude) correction on longitude, so the true ground distance matches the requested magnitude at any latitude. Each candidate is checked against a land mask and re-drawn until it falls on land, so a coastal point is never pushed offshore. The original NA pattern is preserved cell-by-cell and the two axes stay paired (if either coordinate is missing, both are set to NA).

Value

df, with lat_col and lon_col overwritten by the jittered coordinates (rounded to five decimal places, about one metre).

One displacement per site

One displacement is drawn per **site**, not per row, and broadcast to every row of that site. A table that holds many rows per physical place – plots within a trial, samples within a paddock, observations within a farm – therefore comes back with one masked coordinate per place, exactly as the source table has one true coordinate per place. by chooses how a site is identified:

- by = NULL (default) treats rows that share an identical input coordinate pair as one site. This needs no user action and cannot alter genuinely point-level data, where every input pair is already distinct.
- by = c("trial") (one or more column names) treats rows sharing those values as one site, for the case where a site's recorded coordinates differ slightly between rows. Where such a group holds more than one distinct input coordinate, the group's row-weighted centroid is used and the consolidation is reported.
- by = FALSE restores the pre-0.10.0 behaviour of displacing every row independently, and warns if the input carries repeated coordinate pairs. It exists only so a table masked by an earlier version can be reproduced.

Displacing each row of a repeated coordinate separately leaks the site. Donut displacement is isotropic, so the mean of many independent draws around one true site converges on that site:

averaging the 360 rows of a single trial recovers the true position to a median 0.58 km, against the 5 km floor the donut was asked for. One draw per site removes the estimator and leaves the full displacement in place.

Repeated rebuilds

One displacement per site holds *within* a masked table. It does not hold across several. Each rebuild with a fresh seed draws independently, so a site masked repeatedly and kept in several versions can be averaged back out, exactly as its rows could be before. Measured at the 5-20 km default:

Rebuilds kept	Median recovery error
1	14.2 km
2	10.1 km
4	6.4 km
8	4.5 km
16	3.1 km

Eight kept rebuilds fall below the 5 km floor the donut was asked for.

The remedy is to **reuse the seed**. The same `seed` reproduces the same displacement for the same sites, so any number of rebuilds yields one point and nothing to average. Draw a fresh seed only when you intend to supersede every earlier version, and retire the ones you replace.

Choose the unit carefully. It is the finest grouping that denotes **one physical place at one time**. For a multi-year trial series that is location by year, not location alone: trials at one named location in different seasons usually sit in different paddocks, and one displacement for all of them would misplace every season but one.

Choosing the magnitude

The right displacement depends on the density of the entities you are protecting: the masked point should be spatially k-anonymous (roughly, at least k comparable entities lie closer to the masked point than the true one). Individual-level urban health data is typically masked with a standard deviation of about 1 km, because cities are dense. Agricultural fields and farms are orders of magnitude sparser, so a comparable level of protection needs a much larger displacement – a donut of roughly 5 to 20 km (the default) moves a point across several properties while keeping it in the same agroclimatic region. For a formal guarantee, calibrate `min_km` / `max_km` to the local field density to reach a target k-anonymity.

References

Hampton, K. H., Fitch, M. K., Allshouse, W. B., Doherty, I. A., Gesink, D. C., Leone, P. A., Serre, M. L., & Miller, W. C. (2010). Mapping health data: improved privacy protection with donut method geomasking. *American Journal of Epidemiology, 172*(9), 1062-1069. doi:10.1093/aje/kwq248

Zandbergen, P. A. (2014). Ensuring confidentiality of geocoded health data: assessing geographic masking strategies for individual-level data. *Advances in Medicine, 2014*, 567049. doi:10.1155/2014/567049

See Also

synthesise_geospatial() to instead re-anchor coordinates at user-supplied fake centroids.

Examples

```
# Four plots at each of three sites: one masked coordinate per site.
df <- data.frame(
  site = rep(c("A", "B", "C"), each = 4),
  lat  = rep(c(-34.9, -35.2, -33.6), each = 4),
  lon  = rep(c(138.6, 142.0, 148.2), each = 4)
)
if (requireNamespace("maps", quietly = TRUE)) {
  out <- jitter_coordinates(df, "lat", "lon", min_km = 5, max_km = 20,
                            seed = 1)
  nrow(unique(out[c("lat", "lon")])) # 3, one pair per site
}
```

---

mask *Mask a tabular dataset into a structurally faithful development surrogate*

Description

Takes one data frame and a user-edited `roles` table (from propose_roles(), possibly adjusted with set_role()) and produces a synthetic clone according to each column's `action`. Returns a `masque` S7 object holding the synthetic data and a private `masque_recipe`.

Usage

```
mask(
  df,
  roles,
  mode = c("local", "collaborate"),
  seed = NULL,
  clean = c("auto", "report", "off"),
  alias_names = FALSE,
  conditional = FALSE,
  ladder = c("hierarchy", "levels"),
  coords = NULL,
  events = NULL,
  grain = TRUE,
  allow_unmasked_coords = FALSE,
  allow_kept_outcome = FALSE,
  quiet = FALSE,
  .shared_maps = list(),
  .cleaning = NULL,
  ...
)
```

Arguments

df	A data frame, or the masque_conformance object returned by conform_table(), whose data element is then masked. Its cleaning record and its merges and type changes go on the recipe, so apply_recipe() replays them; clean is not applied a second time.
roles	A roles table from propose_roles() (possibly edited). Tables from masque <= 0.5.0 are upgraded with a deprecation warning; see roles_validate().
mode	Either "local" or "collaborate". When omitted, inherit attr(roles, "mode"), falling back to "local" for a roles table with no mode provenance. An explicit collaborate-to-local downgrade raises a classed masque_mode_downgrade warning. A table of fewer than five rows in which any column would be re-simulated (a scrambled covariate, outcome, date or text column, or an events spec) is refused in collaborate mode with masque_too_few_rows_refusal, because such a clone is the original values resampled; local mode warns with masque_too_few_rows. Kept, aliased and dropped columns are not re-simulated, so a short lookup table of those masks in either mode.
seed	Optional integer for reproducibility.
clean	Label and column-name hygiene before masking, passed to clean_table(): one of "auto" (default - trim whitespace and report near-duplicate labels), "report" (report only), or "off" (skip). Invalid column names are legalised in **every** mode – an invalid name silently rewritten during synthesis corrupts the clone – and the repair is raised as a masque_name_repaired warning. The roles table's column references are remapped to the legalised names, the fixes are recorded in the recipe, and apply_recipe() / unmask() reverse them on the round-trip.
alias_names	Hide the column names themselves. FALSE (the default) keeps them. TRUE replaces every retained column name with an opaque alias (col_001, col_002, ... in column order). A character vector names just the columns to alias. The original-to-alias map is stored in the recipe and inverted by apply_recipe() / unmask(), so a pipeline written against the aliased synthetic round-trips. Alias the names when the schema itself is sensitive.
conditional	Logical scalar (default FALSE). The *collaborate-grade conditional clone*. When FALSE, scrambled numeric columns are re-simulated from one global Gaussian copula - marginals and global covariance survive, but the treatment-to-outcome relationship does not, so a causal model fitted on the clone recovers a null effect. When TRUE, the numeric block is re-simulated *within each treatment-by-design stratum*, so a row's synthetic outcome inherits the location of the treatment that row carries. A causal model fitted on the conditional clone recovers the real treatment effect within sampling tolerance. The conditioning columns (treatment plus retained design) are recorded on the recipe. The stratum is chosen by a **coarsening ladder**. Treatment crossed with every retained design column is the finest rung, but on a replicated factorial that rung holds one row per cell, which is too thin to synthesise. The ladder then drops design columns, in the order set by ladder, until the cells hold at least five rows; treatment columns are never dropped. Whatever remains below that floor is pooled into a global fallback. The rung reached is recorded on the recipe as conditioning_used, the columns given up as conditioning_dropped, the

pooled share as `fallback_frac`, and any coarsening or residual pooling raises a classed `masque_conditional_degraded` warning – including the case where no treatment or design column survives at all, in which case the clone is the pooled copula.

`ladder` How the coarsening ladder orders the design columns it may drop, and what it keeps of a dropped one. `"hierarchy"` (the default) drops plot coordinates (`row`, `col`, `range`, `plot`) first, then blocking columns, then environment columns (`site`, `year`, `county`, `trial`, ...), so an environment is never dropped before the replicates inside it; within the last two tiers the column explaining the least variance in the numeric block (adjusted for its number of levels) goes first. The main effect of every dropped column, and of each pair of dropped blocking or environment columns (the replicate inside a county is a different block from the same label in the next county), is estimated by least squares beside the stratum that was kept, removed before the within-stratum copula and added back after it. A clone that conditions on genotype alone therefore still has the original's environment, replicate and block means. Rows pooled into the fallback stratum keep the stratum columns' main effects the same way, which is how a three-replicate variety trial keeps its genotype means. A level with a single row is pooled with the other singletons before the fit, so no row's own outcome is carried. The terms carried are recorded on the recipe as `conditioning_shifted`. `"levels"` is the ladder of masque 0.11.1 to 0.12.0: columns are dropped in decreasing order of distinct values and a dropped column leaves no trace in the clone.

`coords` Optional geographic-coordinate declaration. Supply one or more latitude/longitude pairs and each is coarsened in place by an on-land jitter (see `jitter_coordinates()`) instead of being copula-scrambled into implausible locations. A pair is a named vector `c(lat = "lat_col", lon = "lon_col")` or a named list that also carries jitter parameters (`by`, `method`, `min_km`, `max_km`, `sd_km`, `on_land`); a single pair may be passed bare, and several pairs as a list of pairs. Defaults to a donut of 5-20 km on land. A declared column must have the action `keep` in `roles`; any other action, including the `drop` that `propose_roles()` proposes for a coordinate-named column, is refused with a `masque_coords_action_refusal` error. One displacement is drawn per **site** and broadcast to that site's rows: `by` names the columns of `df` that identify a site, and is omitted when rows sharing an identical input coordinate already identify one. A declared pair always survives masking, coarsened; the recipe records the parameters and the site grouping it was coarsened under, and `apply_recipe()` retargets to the real coordinates. Several published jitters of the same point average toward it: keep the seed, which the recipe records, and do not publish more than one jittered release of the same sites.

`events` Optional linked time-to-event outcomes. One spec per group of linked events, `list(time = c(...), status = c(...))`, with the time and status columns listed in the order the events occur; pass several specs as a list. Every column must have role `outcome` and action `scramble`, times must be numeric and in order in `df`, status columns must hold whole-numbered codes, and none may be missing. The times are turned into gaps between successive events, the rows' status patterns are permuted (within treatment when `conditional = TRUE`), and the gaps are re-simulated within each status pattern by the empirical copula, then summed back into times. Times therefore stay ordered, a gap that every row of a pattern

holds at zero stays zero, and the count of each status pattern is exact. A summed final time is capped at the longest final time observed in the rows its gaps were drawn from (the pattern, within treatment when conditional), so no clone row runs past the observed follow-up. Only the final time and the times tied to it move; a row whose earlier time already passes the cap is scaled down as a whole, which keeps its order and its zero gaps. The recipe records how many rows were capped. A pattern with fewer than five rows draws each of its free gaps from every pattern in which that gap is free, with a `masque_events_pooled` warning; when no other pattern has the gap free, `mask()` stops. No hazard model is fitted and no covariate-to-event relation is kept.

`grain` Single logical (default `TRUE`). Synthesise each column at its **grain**: a scrambled covariate, outcome or date column, or an aliased categorical covariate, that holds at most one distinct non-missing value within every unit of a retained design column, or of a pair of them (`site:year`), is re-simulated once per unit and each unit's new value is written to that unit's rows. The grain is the coarsest such unit with at least two units and at least two rows per unit on average; a unit with no value stays missing. Numeric columns of one grain are drawn jointly through the copula on the unit-level table (with collaborate-mode jitter against the unit-level values), and categorical and date columns are permuted across units, then aliased where their action is `alias`. The clone therefore keeps one value per unit, the frequencies of a categorical across units, and the correlation among columns of the same grain. It does **not** keep the correlation between a grained column and row-level columns, and grained columns are not conditioned on treatment when `conditional = TRUE`. In local mode the unit values are drawn from the original unit values. In collaborate mode a grain with fewer than five units is synthesised row by row instead, with a `masque_grain_small_units` warning, because resampling that few real values would disclose them. Each grain is recorded on the recipe (`grain`) with a line in its warnings. `FALSE` synthesises every column row by row, as masque 0.15 did; detected grains are still recorded. Coordinates declared to `coords`, `events` columns and linked columns are never grained.

`allow_unmasked_coords`
Single logical. A masked table must not carry a real coordinate, so by default `mask()` **stops** when a column whose name says coordinate (`gps`, `lat`, `lon`, `easting`, ...) would be written through unmasked, and warns when a column detected only by the shape of its values would be. State otherwise by declaring the pair to `coords` (coarsened), giving the column a masking action (`drop` or `scramble`), or – having decided the coordinate is not sensitive – setting this to `TRUE`, which is recorded on the recipe.

`allow_kept_outcome`
Single logical. In collaborate mode an outcome with the action `keep` carries its real values beside the re-simulated columns, so `audit_mask()` grades it HIGH and the collaborate write gate stops it. Having decided the real outcome may be shared, set this to `TRUE`: the column is then graded medium and the choice is recorded on the recipe. Local mode is unaffected.

`quiet` Single logical. `TRUE` silences the `masque_mode_downgrade` warning raised when a roles table prepared for `"collaborate"` is used in `"local"` mode, for a plan

that is deliberately shared across both modes. No other warning is affected; a HIGH-leakage finding is always raised.

`.shared_maps` Internal. A named list of pre-computed `original -> alias` level maps for cross-table linked columns, set by `mask_set()`. Not for direct use.

`.cleaning` Internal. The cleaning record of a table `mask_set()` has already cleaned; `mask()` then skips its own cleaning and stores this record on the recipe. Not for direct use.

`...` Must be empty. An unused argument, for example a misspelled name, is an error.

Details

`mode = "local"` marks the synthetic for owner development only; the reminder is recorded on the recipe and shown when the object prints. `mode = "collaborate"` additionally jitters re-simulated numerics within their measurement resolution (stochastically rounding integers) and runs `audit_mask()` automatically; a HIGH finding is raised as a classed warning (`masque_high_leakage`) and blocks the package-managed writers (`masque()`'s `out`, `write_set()`) until it is resolved or explicitly overridden. Which columns are aliased, kept, or dropped is decided by the `action` column of `roles` - `propose_roles()` resolves mode-appropriate defaults, so the table you reviewed is the plan that runs.

Collaborate mode changes the transformations and runs the audit. Whether a synthetic table is appropriate for a given collaborator, environment or jurisdiction is a release decision for the data custodian; masque informs that decision, it does not make it.

Value

A `masque` S7 object. Use `synthetic()` and `recipe()` to extract the components.

Behaviour by action

`keep` Byte-identical pass-through, both modes.

`scramble` Numeric outcome / covariate columns are re-simulated jointly via a Gaussian copula on the global Pearson covariance, with empirical-quantile marginals. Categorical, date, and text columns are row-permuted within non-NA positions, class preserved. Treatment columns get a seeded label permutation - the assignment structure never moves.

`alias` As `scramble` where applicable, plus opaque label substitution: treatments become `trt_NNN` (`<col>_trt_NNN` when two or more treatments are aliased. A numeric treatment becomes text, and `unmask()` restores its exact values), categorical covariates `<col>_LNNN`, design labels `<col>_DNNN` (in place - structure intact), ids `<col>_INNN` (in place - row linkage intact), text values `<col>_TNNN`. A key `mask_set()` shares across tables becomes `<col>_KNNN`. `NNN` has at least three digits.

`drop` Column excluded from the synthetic, both modes.

The NA mask of every retained column is preserved cell-by-cell. RNG state is preserved across the call.

See Also

propose_roles(), set_role(), roles_validate(), synthetic(), recipe(), roles(), reveal_maps(), audit_mask(), data_card().

Examples

```
r <- propose_roles(iris)
r <- set_role(r, "Sepal.Length", role = "outcome")
m <- mask(iris, r, seed = 1)
head(synthetic(m))
```

---

mask_set *Mask a multi-table set with cross-table-consistent aliasing*

Description

The set-level counterpart to mask(). Takes a folder of tabular files, an Excel workbook, or a named list of data frames and produces one synthetic table per input table plus a single private recipe bundle. Columns that are shared across tables - a site code, a genotype name, a plot id appearing in several tables - are aliased *identically everywhere they occur*, so a join written against the synthetic set still resolves on the masked data.

Usage

```
mask_set(
  input,
  roles = NULL,
  links = NULL,
  mode = c("local", "collaborate"),
  seed = NULL,
  clean = c("auto", "report", "off"),
  alias_names = FALSE,
  conditional = FALSE,
  ladder = c("hierarchy", "levels"),
  grain = TRUE,
  allow_kept_outcome = FALSE,
  quiet = FALSE,
  sheets = NULL,
  info = NULL,
  ...
)
```

Arguments

input	A folder path, an .xlsx / .xls path, or a named list of data frames - anything read_set() accepts. A list element may also be a conform_table() result: its cleaning record and conform decisions are used for that table and recorded on its recipe.
roles	Optional named list of roles tables, one per input table (names must match). When NULL (default), propose_roles() runs on each table for the chosen mode. Supply edited tables for full control.
links	Optional override for link detection: a character vector of column names to treat as cross-table links, or FALSE to disable linking entirely (each table masked independently). When NULL (default), links are detected automatically.
mode	Either "local" or "collaborate". When omitted and roles are supplied, inherit their common mode. Tables prepared for different modes must be reconciled explicitly. Otherwise default to "local".
seed	Optional integer for reproducibility.
clean	Hygiene mode passed to clean_table() for every table ("auto", "report", or "off").
alias_names	Hide column names. FALSE (default) keeps them; TRUE aliases every non-link column (linked join keys keep their names so the synthetic set stays joinable).
conditional	Logical scalar (default FALSE). Passed through to each per-table mask() call: when TRUE, every table's numeric block is re-simulated within its own treatment-by-design strata so the treatment-to-outcome relationship survives the clone. See mask() for the full account.
ladder	Drop order of the conditioning ladder, "hierarchy" (default) or "levels"; passed through to each mask() call.
grain	Single logical (default TRUE); passed through to each mask() call, which synthesises a unit-constant column once per unit.
allow_kept_outcome	Single logical, passed to each mask() call: TRUE allows a kept outcome in a collaborate clone and records it.
quiet	Logical; suppress the link / hygiene report, including the hint that a table looks like documentation.
sheets	For an Excel workbook, the sheets to read (default: all); passed to read_set().
info	Optional character vector of table names (sheet names, or file names without extension) to set aside as documentation, for example a data dictionary. They are set aside before masking: not masked, not in the synthetic set, not written by write_set(), and their content is not kept. Their names are recorded on the recipe bundle. A table that looks like documentation and is not named here raises a hint (a message of class masque_info_hint); the hint decides nothing.
...	Must be empty: any argument here is refused with masque_unused_arg_refusal.

Value

A masque_set S7 object. Use synthetic() for the named list of synthetic tables and recipe() for the private recipe bundle.

Links

A *link* is a column name that appears in two or more tables with a compatible kind and overlapping values - the join keys of the set. `mask_set()` proposes links automatically and prints them; pass `links` to override. Linked columns are aliased in place with a shared map (built over the union of values across all tables), so identical original values map to identical aliases in every table. The aliases read `<col>_K001`, `<col>_K002`, ..., assigned at random and, when `seed` is set, the same for the same seed. Set a linked column's action to `keep` in its roles table to pass it through unmasked instead.

See Also

`mask()`, `read_set()`, `write_set()`, `audit_mask()`, `data_card()`, `reveal_maps()`.

Examples

```
tables <- list(
  plots = data.frame(
    site = rep(c("A", "B", "C", "D", "E"), each = 2),
    gen = rep(c("x", "y"), 5),
    yield = c(3.1, 2.9, 4.0, 3.7, 3.3, 3.5, 2.8, 3.0, 3.9, 3.6)
  ),
  sites = data.frame(
    site = c("A", "B", "C", "D", "E"),
    rainfall = c(420, 560, 390, 610, 480)
  )
)
roles <- lapply(tables, propose_roles, mode = "collaborate")
roles <- lapply(roles, set_role, "site", action = "alias")
m <- mask_set(tables, roles = roles, seed = 1, quiet = TRUE)
synthetic(m)$plots$site
synthetic(m)$sites$site # same aliases -> the join still works
```

`masque` *Mask a dataset end to end with one guided call*

Description

The front door. `masque()` walks the whole procedure - read the data, propose column roles, (in an interactive session) pause for you to review the plan, mask, audit, and optionally write the result - from a single call. It dispatches on the input: a single file or data frame goes through `mask()`; a folder, an Excel workbook, or a named list of tables goes through `mask_set()`.

Usage

```
masque(
  input,
  roles = NULL,
```

```
  out = NULL,
  mode = c("local", "collaborate"),
  seed = NULL,
  clean = c("auto", "report", "off"),
  alias_names = FALSE,
  conditional = FALSE,
  ladder = c("hierarchy", "levels"),
  ask = interactive(),
  overwrite = FALSE,
  quiet = FALSE,
  allow_high = FALSE,
  detect = TRUE,
  sheets = NULL,
  info = NULL,
  ...
)
```

Arguments

input	A data frame, a single tabular file (`.csv` / `.tsv` / `.fst`), a folder of such files, an Excel workbook, or a named list of data frames.
roles	Optional. A roles table (single-table input) or named list of roles tables (set input). When supplied, the interactive review is skipped.
out	Optional output path. For a single table, a `.csv` file (or `.xlsx` with `writexl`). For a set, a folder (one CSV per table) or an `.xlsx` workbook. When `NULL` (default) nothing is written.
mode	Either `"local"` or `"collaborate"`. `"local"` makes an owner-only development copy (real labels may remain); `"collaborate"` makes a copy meant to be shared, with labels aliased, numbers jittered, free text dropped and the leakage audit run. When omitted and `roles` are supplied, the mode the roles were proposed for is used, exactly as in `mask()` and `mask_set()`; with no `roles`, the proposal is made for `"local"`.
seed	Optional integer for reproducibility.
clean	Hygiene mode passed to `clean_table()` (`"auto"`, `"report"`, `"off"`).
alias_names	Hide column names; see `mask()` / `mask_set()`.
conditional	Logical scalar (default `FALSE`). The conditional clone mode passed through to `mask()` / `mask_set()`: when `TRUE`, numeric columns are re-simulated within each treatment-by-design stratum so the treatment-to-outcome relationship survives the clone. See `mask()` for the full account.
ladder	Drop order of the conditioning ladder, `"hierarchy"` (default) or `"levels"`; passed through to `mask()` / `mask_set()`.
ask	Whether to pause for interactive review when `roles` is not supplied. Defaults to `interactive()`. Set `FALSE` to proceed with the proposed plan without prompting.
overwrite	Passed to the writer when `out` is set.

quiet	Suppress progress messages. Warnings - including the HIGH-leakage finding - are never suppressed.
allow_high	Logical (default FALSE). When out is set and the collaborate-mode audit flagged HIGH leakage, the write is refused. Pass TRUE to write anyway after your own review; the override raises a masque_high_override warning and is recorded in the recipe's warnings, so the exception stays auditable.
detect	Logical (default TRUE). Passed to propose_roles() when roles are proposed: FALSE skips design detection and proposes from column names and types alone.
sheets	For an Excel workbook, the sheets to read (default: all). See mask_set().
info	For a set, names of documentation tables to set aside: not masked and not written. See mask_set().
...	Further arguments for the masking verb: mask() for one table (for example coords, events, allow_unmasked_coords, grain), mask_set() for a set (for example links, grain). An argument the verb does not take is refused (masque_unused_arg_refusal).

Details

It is also fully scriptable. Pass an edited roles table (or named list of them) to skip the interactive review, and an out path to write the masked result in one go. The returned object is the same masque / masque_set you would get from the lower-level verbs, so anything you can do with those you can do with the result here.

Value

A masque object (single-table input) or a masque_set object (set input), invisibly. Use synthetic() and recipe().

The guided flow

1. **Read** the input into one or more clean rectangular tables.
2. **Propose roles** for every column (skipped if you pass roles).
3. **Review** - in an interactive session with no roles supplied, the proposed plan is printed and you are asked to proceed, edit, or stop. Editing opens the roles table in the spreadsheet editor (utils::edit()) where the platform provides one; when it cannot start (for example macOS without XQuartz, or a headless session), a console editor takes over - pick a column, then a role and an action from numbered menus, with every change applied through set_role() so default actions re-resolve exactly as on the scriptable path. With ask = FALSE (the default in non-interactive use) the proposed plan is used as-is, with a note.
4. **Mask** the data in the chosen mode.
5. **Audit** - in collaborate mode the leakage audit runs and its headline is printed. A HIGH finding surfaces as a classed warning (masque_high_leakage) - it is never silenced by the guided flow.
6. **Write** - if out is set, the masked data is written there (mirroring the input format), *unless* the audit left unresolved HIGH findings: then the write is refused (nothing is written) and the flagged columns are listed. Resolve them and mask again, or pass allow_high = TRUE after your own review - the override is warned (masque_high_override) and recorded on the recipe. The private recipe is never written automatically; persist it yourself with save_recipe().

See Also

mask(), mask_set(), propose_roles(), set_role(), apply_recipe() and unmask() for the round trip, save_recipe() to keep the recipe, audit_mask(), data_card() and roles() to describe the clone, write_set().

Examples

```
# Scripted single-table use (no prompt because roles are supplied):
r <- propose_roles(iris)
r <- set_role(r, "Sepal.Length", role = "outcome")
m <- masque(iris, roles = r, seed = 1, ask = FALSE, quiet = TRUE)
head(synthetic(m))
```

---

plot_design_summary *Sanity-check visualisation for detected scope and design*

Description

Plots the structure that drove the detect_design() verdict. A detected MET uses a compact environment-coverage overview by default. Set environment to draw one selected environment. A single-trial panel depends on the detected design class:

Usage

```
plot_design_summary(
  x,
  df,
  engine = c("base", "ggplot2"),
  environment = NULL,
  ...
)
```

Arguments

x	A design_summary object from detect_design().
df	The data frame that was passed to detect_design(). It is used to draw replication tiles, spatial layouts, and the NA-pattern. The data are deliberately not stored in the summary.
engine	"base" (default) or "ggplot2". The latter requires ggplot2 and falls back to "base" with a warning if unavailable.
environment	Optional single environment label. For a detected MET, the default is a compact all-environment overview. Supplying a label draws the selected environment's field layout or legacy design diagnostic.
...	Ignored.

Details

- `CRD`, `factorial`, `none` -> frequency-of-treatment + NA-pattern.
- `RCBD`, `IBD/alpha-lattice` -> treatment x block replication tile.
- `row-column` -> spatial layout tile (row x col, fill = treatment).
- `split-plot` -> factor-nesting tree + within-block replication.

Output is purely diagnostic. Do not use it as a publication figure (use `desplot::desplot()` or `ggplot2`-based packages for that).

Value

With `engine = "ggplot2"`, the `ggplot` object, so it can be modified or saved with `ggplot2::ggsave()`; nothing is drawn until it is printed. With `engine = "base"`, the input `x` invisibly, after the plot has been drawn.

Examples

```
if (requireNamespace("agridat", quietly = TRUE)) {
  d <- agridat::john.alpha
  ds <- detect_design(d)
  plot(ds, df = d)
}

met <- expand.grid(
  env = factor(c("E1", "E2")),
  rep = factor(seq_len(2L)),
  gen = factor(c("G1", "G2", "G3"))
)
met$yield <- seq_len(nrow(met))
met_design <- detect_design(met, env = "env")
plot(met_design, df = met)
```

`propose_roles` *Propose role and action classifications for the columns of a data frame*

Description

Generates a heuristic two-axis classification for every column of `df`: a `role` (what the column *is*) and an `action` (what `mask()` will *do* to it). The user is expected to inspect this table and edit it - directly or via `set_role()` - before passing it to `mask()`.

Usage

```
propose_roles(df, mode = c("local", "collaborate"), detect = TRUE)
```

Arguments

df	A data frame. Must have at least one column.
mode	The masking mode the table is being prepared for: "local" (default) or "collaborate". Stored as attr(roles, "mode") and used to resolve default actions.
detect	Logical scalar (default TRUE). When TRUE, run detect_design() and overlay its recommended role hints.

Value

A data.table with one row per column: col, role, action, kind (storage kind: numeric, integer, factor, character, logical, date, datetime, other), freq_or_range, pii_suspected, and notes. The target mode is stored as attr(roles, "mode").

The two axes

role describes the column and determines the *mechanics* of any synthesis:

design Experimental / structural columns (site, block, rep, plot, year). Mechanics of alias: labels are substituted in place, structure intact.

treatment Assignment columns (variety, genotype, dose). Labels are remapped in place - the assignment structure never moves. scramble = seeded label permutation; alias = opaque labels (trt_001).

outcome Numeric response columns. scramble re-simulates via the Gaussian copula, jointly with scrambled numeric covariates. Multiple outcomes are supported.

covariate Everything measured alongside. Numeric: copula re-simulation. Categorical: row permutation, plus opaque label aliasing under alias.

date Date / POSIX / difftime columns. scramble row- permutes within the observed values; class and NA pattern are preserved.

id Identifier columns. Never scrambled (that would break row linkage); alias substitutes opaque per-value labels in place, preserving linkage.

text Free-text columns. scramble row-permutes; alias tokenises each distinct string.

other Classes masque cannot synthesise (list columns, exotic S4, ...). Keep or drop only.

action sets the masking depth per column:

keep Byte-identical pass-through, both modes.

scramble Re-simulate (numeric) or row-permute (categorical / date / text); original label vocabulary remains visible.

alias Scramble where applicable, plus opaque label substitution - the vocabulary itself is hidden.

drop Column excluded from the synthetic, both modes.

The proposed action column is resolved for mode, so the table you edit shows the actual masking plan. Re-assigning a column's role with set_role() re-resolves its default action; a direct roles$role[...] <- ... edit leaves action untouched (set it to NA to have mask() re-resolve the default).

Default classification rules, applied in order

1. PII-pattern column names (`contact`, `email`, `phone`, `gps`, `latitude` / `longitude`, `postcode`, `ssn`, `password`, `owner`, `farmer`, `operator`, etc., case-insensitive substring) -> `pii_suspected = TRUE` and action `drop` in **both** modes. Re-role deliberately if the column must survive.
2. Date / POSIXct / POSIXlt / difftime columns -> role `date`, action `scramble` (row permutation).
3. ID-pattern names (`\\bid\\b`, `_id$`, `^id_`) -> role `id`; kept in local mode, dropped in collaborate mode.
4. Design-pattern names (`rep`, `block`, `row`, `col(umn)?`, `range`, `plot(no)?`, `site`, `site_name`, `trial_site`, `env(ironment)?`, `trial`, `year`, `season`, `colrep`, `tos`; the two-word names also spaced or dotted, as in `Site Name` and `Site.Name`) -> role `design`, action `keep`.
5. Treatment-pattern names (`treatment`, `variety`, `cultivar`, `genotype`, `^trt`, `^dose`) -> role `treatment`; kept in local mode, aliased in collaborate mode.
6. Numeric columns with an outcome-pattern name (`yield`, `yield_t_ha`, `Yield..t.ha.`, `grain_yield`, `response`, `outcome`) -> role `outcome`, action `scramble`.
7. Character columns with > 50% unique values on non-NA -> role `text`; kept in local mode, dropped in collaborate mode.
8. Unsupported classes -> role `other`, action `keep`, with a note.
9. Everything else -> role `covariate`, action `scramble`. Re-role other response variables as `outcome`.

No outcome is required: with no column roled `outcome`, the copula simply re-simulates all scrambled numeric columns jointly.

Since masque 0.3.0, `propose_roles()` also calls `detect_design()` by default (`detect = TRUE`) and applies the detected design's `recommended_roles` on top of the name-based heuristic, re-resolving the default action for any promoted column. The design summary is stashed as `attr(roles, "design")`. Pass `detect = FALSE` for the name-only heuristic.

For a column without a treatment-pattern name, a treatment hint from the design rules is applied only when the column is crossed with a design column: a design-pattern name, a column roled `design`, or a detected environment. A column with one value per level of a design column is a unit-level covariate and is never promoted. Hints that fail are listed in `attr(roles, "design")@recommended_roles` with `auto_apply = FALSE` and the reason.

Multi-environment trials

Since masque 0.9.1, high-confidence environment columns detected by `detect_design()` are promoted to role `design`. In local mode their values remain byte-identical. In collaborate mode categorical environment labels default to `alias`, which preserves row assignment and factor codes while hiding the vocabulary. Numeric environment columns default to `keep` and raise a `masque_environment_disclosure` warning for explicit review.

Weak or competing environment candidates never change roles automatically. Inspect `attr(roles, "design")` and use `set_role()` when domain knowledge should override an uncertain result. Preserving environment allocation does not imply that synthesised outcomes preserve treatment-by-environment effects. An explicitly chosen action is pinned and is not overwritten by a later design-role promotion.

See Also

set_role() to edit; roles_validate() for the fail-closed validation applied at mask() time.

Examples

```
propose_roles(iris)
propose_roles(iris, mode = "collaborate")

met <- expand.grid(
  env = factor(c("E1", "E2")),
  rep = factor(seq_len(2L)),
  gen = factor(c("G1", "G2", "G3"))
)
propose_roles(met, mode = "collaborate")[, c("col", "role", "action")]
```

---

read_recipe *Read a masque recipe from disk*

Description

Loads a recipe written by save_recipe(). Validates that the file contains a masque_recipe or a masque_recipe_set bundle and informs (does not error) if it was written by a different package version than the one currently installed. Bundles are read from masque 0.16.0 on.

Usage

```
read_recipe(path)
```

Arguments

path	File path.

Value

A masque_recipe, or a masque_recipe_set bundle.

See Also

save_recipe(), recipe().

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
m <- mask(iris, r, mode = "collaborate", seed = 1)
tmp <- tempfile(fileext = ".rds")
save_recipe(recipe(m), tmp)
rec2 <- read_recipe(tmp)
```

```
rec2
```

`read_set` *Read a set of tables from a folder, an Excel workbook, or a list*

Description

Ingests a multi-table dataset into a named list of data frames, ready for `mask_set()`. Three input shapes are accepted:

Usage

```
read_set(input, sheets = NULL, pattern = NULL)
```

Arguments

`input`	A folder path, an `.xlsx` / `.xls` path, or a named list of data frames.
`sheets`	For an Excel workbook, an optional character vector of sheet names to read (default: all sheets). Any other input with `sheets` set is refused (`masque_bad_sheets_refusal`).
`pattern`	For a folder, an optional regular expression to select files (default: all `.csv` / `.tsv` / `.fst`).

Details

a folder path Every `.csv`, `.tsv`, and `.fst` file in the folder becomes one table, named by its file name (without extension). CSV / TSV are read with `data.table::fread()`; `.fst` needs the Suggested `fst` package.

an Excel workbook (`.xlsx` / `.xls`) Every sheet becomes one table, named by its sheet name. Needs the Suggested `readxl` package.

a named list of data frames Returned as-is after validation.

Only clean rectangular tables are supported. A sheet or file that does not read as a rectangle - missing header row (auto-named `...1` columns), zero rows, or zero columns - fails with an explanatory error naming the offending table. `masque` does not attempt to recover merged cells, multi-row headers, or junk rows; tidy the source first.

Value

A named list of data.tables.

See Also

`mask_set()`, `write_set()`.

Examples

```
tables <- list(
  plots = data.frame(site = c("A", "B"), yield = c(3.1, 4.2)),
  sites = data.frame(site = c("A", "B"), rainfall = c(400, 550))
)
read_set(tables)
```

`recipe` *Extract the recipe from a masque object*

Description

The recipe is **private**: at least as sensitive as the original data. Never share alongside the synthetic. By default `print(recipe(m))` redacts all level maps; use `reveal_maps()` for an explicit reveal.

Usage

```
recipe(m)
```

Arguments

`m`	A `masque` object from `mask()`, or a `masque_set` from `mask_set()`.

Value

A `masque_recipe`, or for a set a `masque_recipe_set` bundle.

See Also

`synthetic()`, `reveal_maps()`, `mask()`, `mask_set()`.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
m <- mask(iris, r, seed = 1)
recipe(m)
```

`reveal_maps` *Reveal the level maps held inside a recipe*

Description

Explicit, audited reveal: prints each `original -> synthetic` level map held by the recipe. **Use sparingly.** Recipe maps are at least as sensitive as the original data; printing them defeats the redaction built into the `print()`, `format()`, `str()` and `knitr::knit_print()` methods of recipes, recipe bundles, `masque` and `masque_set` objects. `saveRDS()` and `dput()` write the maps by design.

Usage

```
reveal_maps(rec, table = NULL)
```

Arguments

`rec`	A `masque_recipe` object (e.g., from `recipe(m)`), or the `masque_recipe_set` bundle of a `mask_set()` result.
`table`	For a bundle, the names of the tables whose maps to reveal. `NULL` (default) reveals every table. Must be `NULL` for a single recipe.

Details

For each recipe it prints the column-name map (when names were aliased), every level map as `original -> alias`, and the seed. The alias labels follow the scheme described under `alias` in `mask()`: `trt_NNN` or `<col>_trt_NNN` for treatments, `<col>_LNNN`, `<col>_DNNN`, `<col>_INNN` and `<col>_TNNN` for covariates, design columns, ids and text, and `<col>_KNNN` for a key shared across the tables of a set.

Value

`rec`, invisibly.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
r$role[r$col == "Species"] <- "covariate"
m <- mask(iris, r, mode = "collaborate", seed = 1)
rec <- recipe(m)
reveal_maps(rec)
```

roles *The role table of a masked clone*

Description

Returns the role each column of the clone was masked under, in the clone's own vocabulary: one row per column of `synthetic()`, in the same order and under the same names. When the column names were aliased (`alias_names` in `mask()`), the rows carry the aliases, so the table can travel with the clone without disclosing the original names. A dropped column is not in the clone and has no row.

Usage

```
roles(m)
```

Arguments

`m`	A `masque` object from `mask()`, or a `masque_set` from `mask_set()`.

Details

Only the four public columns are returned. The proposal's notes and its value summaries describe the original table and stay on the private recipe, as does the full role table used on the original (`recipe(m)@roles`).

Value

For a `masque`, a data.table with columns `col`, `role`, `action` and `kind`. For a `masque_set`, a named list of such tables, one per table of the set.

See Also

`synthetic()`, `data_card()`, `mask()`.

Examples

```
r <- propose_roles(iris)
r <- set_role(r, "Sepal.Length", role = "outcome")
m <- mask(iris, r, seed = 1)
roles(m)
```

`roles_validate` *Validate a roles table*

Description

Fail-closed validation of a two-axis `roles` table before `mask()` consumes it. Returns the validated table - with any `NA` actions resolved to their (role, kind, mode) defaults - so callers can use the return value directly.

Usage

```
roles_validate(roles, df = NULL, mode = NULL)
```

Arguments

`roles`	A roles table from `propose_roles()` (possibly edited), or a v1 roles table (deprecated, upgraded with a warning).
`df`	Optional data frame. If supplied, `roles` is checked for one-to-one column-name correspondence with `df`.
`mode`	Optional mode (`"local"` or `"collaborate"`) used to resolve `NA` actions and to upgrade v1 tables. Defaults to `attr(roles, "mode")`, falling back to `"local"`.

Details

Tables produced by masque 0.5.0 and earlier (no `action` column; v1 vocabulary with `keep` / `ignore` roles and the optional `mask_levels` column) are upgraded in place with a deprecation warning. The upgrade preserves the v1 semantics exactly: v1 `keep` becomes action `keep`; v1 `ignore` becomes role `id` / `text` / `other` with action `keep` in local mode and `drop` in collaborate mode; v1 treatment `mask_levels = "permute"` becomes action `scramble`.

Hard errors:

- missing required columns (`col`, `role`, `action`, `kind`);
- unknown role (not in `design`, `treatment`, `outcome`, `covariate`, `date`, `id`, `text`, `other`) or unknown action (not in `keep`, `scramble`, `alias`, `drop`);
- any `NA` role (an `NA` *action* is allowed - it resolves to the default for the row's role and kind);
- an incompatible (role, action, kind) combination, e.g. `design` + `scramble`, numeric + `alias`, `id` + `scramble`, `other` + anything but keep / drop;
- duplicate `col` entries;
- if `df` supplied: any `df` column missing from `roles`, or any `roles` column missing from `df`.

Loud advisories (warnings, not errors):

- every action is `keep` - the "synthetic" would equal the original byte-for-byte;
- the table was proposed for one mode but is being validated for another (actions are taken as-is; defaults are not re-resolved).

Value

The validated (and possibly upgraded / resolved) roles table, invisibly.

See Also

propose_roles(), set_role().

Examples

```
r <- propose_roles(iris)
r <- set_role(r, "Sepal.Length", role = "outcome")
roles_validate(r, iris)
```

---

role_options *List every role and action combination masque accepts*

Description

Renders the two-axis vocabulary as a table: every role paired with every action, the storage kinds the pair works for, and the reason the pair is constrained when it is. The table is generated from the same compatibility rules roles_validate() applies at mask() time, so what it shows is exactly what a roles table is allowed to say.

Usage

```
role_options(kind = NULL)
```

Arguments

kind	Optional storage kind to filter for: one of "numeric", "integer", "factor", "character", "logical", "date", "datetime", or "other". When supplied, only the combinations workable for a column of that kind are returned. The default NULL returns the full grid.

Details

A column's kind is never chosen: propose_roles() derives it from the column's class. To move a column to a different kind, convert the column in the data and re-propose.

Value

A data.table with one row per role-action pair and four columns: role, action, kinds (the storage kinds the pair is valid for: "all", "none", "all except other", or a comma-separated list), and notes (why the pair is constrained, empty when it is not).

See Also

propose_roles(), set_role(), roles_validate().

Examples

```
role_options()
role_options(kind = "factor")
subset(role_options(), role == "design")
```

`save_recipe` *Save a masque recipe to disk*

Description

Writes the recipe to a single `.rds` file. The default is **runtime-minimal**: no simulator state (copula covariance, raw margins) is written, only the translation maps, factor metadata, storage classes, integrity fingerprint, and warnings. This keeps the saved artefact small and reduces the information that would leak if the recipe file alone were shared.

Usage

```
save_recipe(rec, path, include_simulator = FALSE)
```

Arguments

`rec`	A `masque_recipe`, or the `masque_recipe_set` bundle of a `mask_set()` result, e.g. from `recipe(m)`.
`path`	File path ending in `.rds` (any case); other extensions are refused.
`include_simulator`	Logical. Has no effect; the recipe is always written without simulator state.

Details

`include_simulator = TRUE` is accepted and has no effect: the recipe does not carry simulator state.

Recipes are at least as sensitive as the original data. Protect the saved file at the same security class as the original. Note that `save_recipe()` writes plain R serialisation - it does not encrypt. The saved recipe is a re-identification key: store it under your organisation's access controls and key-management practice, not alongside the synthetic output.

Value

`path`, invisibly.

See Also

`read_recipe()`, `recipe()`.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
m <- mask(iris, r, mode = "collaborate", seed = 1)
tmp <- tempfile(fileext = ".rds")
save_recipe(recipe(m), tmp)
rec2 <- read_recipe(tmp)
```

`set_role` *Set the role and action of one or more columns in a roles table*

Description

Ergonomic editor for the two-axis roles table returned by `propose_roles()`. Setting a `role` without an explicit `action` re-resolves the action to the default for the new role, the column's kind, and the mode the table was proposed for – so a re-roled column never silently carries a stale action from its previous role.

Usage

```
set_role(roles, cols, role = NULL, action = NULL)
```

Arguments

`roles`	A roles table from `propose_roles()` (possibly edited).
`cols`	Character vector of column names to edit. Every entry must be present in `roles$col`.
`role`	Optional single role to assign to all of `cols`. One of `"design"`, `"treatment"`, `"outcome"`, `"covariate"`, `"date"`, `"id"`, `"text"`, `"other"`.
`action`	Optional single action to assign to all of `cols`. One of `"keep"`, `"scramble"`, `"alias"`, `"drop"`. When `NULL` (the default) and `role` was supplied, the action is re-resolved to the default for the new role.

Details

Direct edits (`roles$role[roles$col == "x"] <- "outcome"`) also work, but leave `roles$action` unchanged.

Value

The edited roles table.

See Also

`propose_roles()`, `roles_validate()`.

Examples

```
r <- propose_roles(iris)
r <- set_role(r, "Sepal.Length", role = "outcome")
r <- set_role(r, "Species", action = "keep")
r[, c("col", "role", "action")]
```

`synthesise_geospatial` *Re-anchor synthetic geospatial coordinates at plausible-but-fake locations*

Description

Replaces the latitude / longitude values in a masqued data frame with coordinates anchored at user-supplied centroids and clustered to preserve the original's site-count-per-anchor structure. The function never reads the real coordinates beyond counting how many distinct sites the original holds per anchor level – so it leaks the replication-per-site distribution and the count of distinct sites, nothing more.

Usage

```
synthesise_geospatial(
  synth,
  original,
  anchor_col,
  lat_col,
  lon_col,
  anchor_centroids,
  site_spread_deg = 0.6,
  jitter_deg = 0.05,
  seed = NULL
)
```

Arguments

`synth`	A synthetic data frame (typically `synthetic(mask(...))`).
`original`	The original data frame from which `synth` was derived (needed only to count distinct sites per anchor).
`anchor_col`	Name of the column whose levels anchor each cluster (e.g., `"M_STATE"`). Must exist in both `synth` and `original`.
`lat_col, lon_col`	Column names of the latitude and longitude fields to overwrite in `synth`.
`anchor_centroids`	Named list keyed by anchor levels; each element is a length-2 numeric named `c(lat, lon)`. The user supplies plausible centroids (e.g., state centroids); the

function never infers them from the original to avoid leaking position information. Any other shape (a data frame, an unnamed list, an element that is not a `c(lat, lon)` pair, or a list naming none of the anchor levels) is refused with a `masque_bad_anchor_centroids_refusal` error. A level absent from an otherwise well-formed list is warned about and its rows get `NA`.

`site_spread_deg`
: Half-width of the box (in decimal degrees) around each anchor centroid within which fake site centroids are uniformly placed. Default `0.6`.

`jitter_deg`
: Within-site uniform jitter (in decimal degrees) added to each row's assigned site centroid. Default `0.05`.

`seed`
: Optional integer seed for reproducibility. The function uses `withr::local_preserve_seed()` so the caller's RNG state is left untouched.

Details

Typical use: after `mask()` produces a synthetic with copula-drawn or missing coordinates, call `synthesise_geospatial()` to substitute plausible points. The synthetic ends up with:

the same number of distinct sites per anchor level (e.g., if the original has five distinct trial sites in NSW, the synthetic will have five fake sites in NSW);

the original's per-site replication distribution (each fake site receives a share of the synthetic rows proportional to its real counterpart's count);

within-site tight clustering and between-site spread (small jitter within site; larger spread between sites within each anchor centroid's neighbourhood).

What the function does **not** preserve:

- the real positions of the sites (they are random within a user-defined neighbourhood of each anchor centroid);
- the relative spacing or bearings between real sites;
- any spatial autocorrelation in the outcome.

Coordinates that are `NA` in the original remain `NA` in the synthetic – the NA pattern is preserved cell-by-cell.

Value

`synth`, with `lat_col` and `lon_col` overwritten by the re-anchored coordinates.

Examples

```
# Toy example: 50 rows split across two states.
set.seed(1)
n <- 50
df <- data.frame(
  state = sample(c("NSW", "VIC"), n, replace = TRUE),
  lat   = stats::rnorm(n, -33, 0.3),
  lon   = stats::rnorm(n, 145, 0.3),
  y     = stats::rnorm(n)
```

```
)
roles <- propose_roles(df, detect = FALSE)
roles$role[roles$col == "y"] <- "outcome"
roles$role[roles$col %in% c("lat", "lon")] <- "covariate"
roles$role[roles$col == "state"] <- "design"
m <- mask(df, roles, mode = "collaborate", seed = 1L)
centroids <- list(
  NSW = c(lat = -32.5, lon = 147),
  VIC = c(lat = -36.5, lon = 144)
)
synth_geo <- synthesise_geospatial(
  synthetic(m), df,
  anchor_col = "state", lat_col = "lat", lon_col = "lon",
  anchor_centroids = centroids, seed = 2L
)
head(synth_geo[, c("state", "lat", "lon")])
```

`synthetic` *Extract the synthetic data from a masque object*

Description

Extract the synthetic data from a masque object

Usage

```
synthetic(m)
```

Arguments

`m`	A `masque` object from `mask()`, or a `masque_set` from `mask_set()`.

Value

For a `masque`, a data.table (the synthetic table); for a `masque_set`, a named list of synthetic tables.

See Also

`recipe()`, `mask()`, `mask_set()`.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Sepal.Length"] <- "outcome"
m <- mask(iris, r, seed = 1)
head(synthetic(m))
```

unmask *Translate data from the synthetic namespace back to the original*

Description

Inverse of `apply_recipe()`. Accepts a data frame or an atomic vector. For an atomic factor / character vector with a recipe that holds multiple level maps, `column` must name which map to invert. Atomic numeric, integer, logical, and `Date` / `POSIXct` vectors are returned unchanged (no inverse map applies – these are pass-through under `apply_recipe()` too).

Usage

```
unmask(x, rec, column = NULL)
```

Arguments

`x`	A data frame or an atomic vector to translate from synthetic-namespace to original-namespace.
`rec`	A `masque_recipe` object.
`column`	Optional column name. Only consulted for atomic factor / character `x` when the recipe holds more than one level map. Ignored for atomic numeric / logical / Date-like input (pass-through), but if supplied is validated against the recipe's known columns.

Details

The most common pattern is round-tripping pipeline predictions:

```
fit                <- my_model(synthetic(m))           # train on synthetic
orig_in_synth_space <- apply_recipe(original, recipe(m)) # forward
preds_synth        <- predict(fit, orig_in_synth_space)
preds_orig         <- unmask(preds_synth, recipe(m))    # inverse
```

An unknown level (a synthetic alias not in the recipe's map) is an error; it is never coerced to `NA`.

Value

An object of the same type as `x`, in the original namespace.

See Also

`apply_recipe()`, `mask()`.

Examples

```
r <- propose_roles(iris)
r$role[r$col == "Species"] <- "treatment"
m <- mask(iris, r, mode = "collaborate", seed = 1)
synth <- synthetic(m)
rec <- recipe(m)
# round-trip a synthetic-namespace value back to its original label
head(unmask(synth$Species, rec, column = "Species"))
```

`write_set` *Write a masked set to disk, mirroring the input format*

Description

Writes the synthetic tables of a `mask_set()` result to disk. The output format mirrors the request: a `.xlsx` path produces one workbook with one sheet per table; a folder path produces one `.csv` per table. The private recipe bundle is **never** written by this function - persist it separately with `save_recipe()` and protect it at the same security class as the original data.

Usage

```
write_set(m, path, overwrite = FALSE, allow_high = FALSE)
```

Arguments

`m`	A `masque_set` object from `mask_set()`.
`path`	Output location. A path ending in `.xlsx` writes a workbook (needs the Suggested `writexl` package); any other path is treated as a folder, created if necessary, and receives one `<table>.csv` per table.
`overwrite`	Logical. When `FALSE` (default), writing over an existing file or a non-empty folder errors.
`allow_high`	Logical (default `FALSE`). Override the HIGH-leakage write refusal after your own review; the override is raised as a `masque_high_override` warning.

Details

In collaborate mode the mask-time audit gates the write: unresolved HIGH leakage findings refuse the write (nothing is written) until the flagged columns are re-roled, aliased, or dropped - or the refusal is explicitly overridden with `allow_high = TRUE`, which raises a `masque_high_override` warning so the exception stays visible.

Value

`path`, invisibly.

See Also

mask_set(), read_set(), save_recipe().

Examples

```
tables <- list(
  plots = data.frame(site = c("A", "B"), yield = c(3.1, 4.2)),
  sites = data.frame(site = c("A", "B"), rain = c(420, 560))
)
m <- mask_set(tables, seed = 1, quiet = TRUE)
dir <- file.path(tempdir(), "masked_set")
write_set(m, dir)
list.files(dir)
```

Index

apply_recipe, 2
apply_recipe(), *7*, *9*, *18, 19*, *27*, *43*
audit_mask, 4
audit_mask(), *20–22*, *24*, *27*

clean_table, 6
clean_table(), *3*, *7–9*, *18*, *23*, *25*
conform_table, 7
conform_table(), *3*, *6*, *18*, *23*

data_card, 10
data_card(), *22*, *24*, *27*, *35*
detect_design, 12
detect_design(), *27*, *29, 30*

interactive(), *25*

jitter_coordinates, 14
jitter_coordinates(), *19*

maps::map.where(), *15*
mask, 17
mask(), *2–7*, *9–11*, *15*, *22–29*, *33–37*, *42, 43*
mask_set, 22
mask_set(), *4*, *9–11*, *21*, *24–27*, *32–35*, *38*, *42*, *44, 45*
masque, 24
masque(), *21*

plot(), *13*
plot_design_summary, 27
plot_design_summary(), *13*
propose_roles, 28
propose_roles(), *7*, *9*, *12, 13*, *17–19*, *21–23*, *26, 27*, *30*, *36, 37*, *39*

read_recipe, 31
read_recipe(), *38*
read_set, 32
read_set(), *23, 24*, *45*
recipe, 33
recipe(), *21–23*, *26*, *31*, *38*, *42*
reveal_maps, 34
reveal_maps(), *22*, *24*, *33*
role_options, 37
roles, 35
roles(), *10, 11*, *22*, *27*
roles_validate, 36
roles_validate(), *18*, *22*, *31*, *37*, *39*

save_recipe, 38
save_recipe(), *26, 27*, *31*, *44, 45*
set_role, 39
set_role(), *17*, *22*, *26–31*, *37*
synthesise_geospatial, 40
synthesise_geospatial(), *17*
synthetic, 42
synthetic(), *10, 11*, *21–23*, *26*, *33*, *35*

unmask, 43
unmask(), *3*, *18*, *21*, *27*
utils::edit(), *26*

write_set, 44
write_set(), *21*, *23, 24*, *27*, *32*