Package: SynthesizerPlus (via r-universe)

October 16, 2026

Type Package

Title Synthetic Data Generation with Copulas, Evaluation and Multi-Format I/O

Version 0.2.9

Description Generates synthetic versions of numeric, categorical, mixed-type and time-series data. Marginal distributions are modelled with interpolated empirical quantile functions and dependence with Gaussian or Student-t copulas (Sklar, 1959). Discrete and categorical variables enter the copula through the distributional transform (Ruschendorf, 2009, <doi:10.1016/j.jspi.2009.05.030>), and their latent correlations are estimated by maximum likelihood as polychoric (Olsson, 1979, <doi:10.1007/BF02296207>) and polyserial (Olsson, Drasgow and Dorans, 1982, <doi:10.1007/BF02294164>) correlations. Missing values are modelled by missingness indicators inside the copula, so that missingness may depend on other variables. Supports stratified models, a closeness parameter with calibration to a target distance or disclosure-risk level, chunked generation of very large data sets, copula vector-autoregressive and bootstrap time-series synthesis, augmentation of small training sets for machine learning, and random generation from multivariate distributions and elliptical and Archimedean copulas. Fidelity, utility and disclosure risk are assessed with marginal distances, association matrices, propensity score mean-squared error (Snoke et al., 2018, <doi:10.1111/rssa.12358>), confidence-interval overlap, correct attribution probability and distance to closest record, with 'ggplot2' visualisations. Data can be read from and written to CSV, TSV, RDS, RData, Parquet, Feather, HDF5, JSON, NDJSON, Excel, SPSS, Stata, SAS and fst files. The empirical-quantile and copula approach builds on ideas of the 'synthesizer' package by Mark van der Loo.

License EUPL (>= 1.2)

URL https://github.com/bogdanoancea/SynthesizerPlus

BugReports https://github.com/bogdanoancea/SynthesizerPlus/issues

Depends R (>= 4.1.0)

Imports ggplot2 (>= 3.4.0), stats, tools, utils

Suggests arrow, data.table, fst, haven, hdf5r, jsonlite, knitr, MASS, nanoparquet, openxlsx, readxl, rmarkdown, rpart, testthat (>= 3.0.0), writexl

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Language en-GB

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Bogdan Oancea [aut, cre, cph]

Maintainer Bogdan Oancea <bogdan.oancea@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-16 13:30:15 UTC

RemoteUrl https://github.com/cran/SynthesizerPlus

RemoteRef HEAD

RemoteSha 0432dc5c4cba088a48f1784c462767381b330353

Contents

association_matrix . . . 3
augment_data . . . 4
autoplot.sp_comparison . . . 6
calibrate_closeness . . . 6
census_sim . . . 8
ci_overlap . . . 9
compare_synthetic . . . 10
copula_correlation . . . 11
dcr . . . 12
disclosure_risk . . . 13
discriminator_auc . . . 15
fit_synthesizer . . . 16
fit_synthesizer.ts . . . 20
generate . . . 22
generate_to_file . . . 23
make_corr . . . 24
margin_categorical . . . 25
marginal_metrics . . . 26
match_types . . . 27

mvdist 27
perturbation 29
plot_association 31
plot_dcr 32
plot_marginals 32
plot_pairs 33
plot_qq 34
plot_ts 35
pmse 35
r_copula 37
r_mvdist 38
read_data 39
simulate.sp_synthesizer 40
supported_formats 41
synthesize 41
ts_windows 42
write_data 43

Index **45**

`association_matrix` *Mixed-type association matrix*

Description

Computes pairwise association for data with numeric and categorical variables: Spearman correlation for numeric–numeric pairs, bias-corrected Cramér's V for categorical–categorical pairs, and the correlation ratio η (on ranks) for numeric–categorical pairs. Pairwise complete observations are used.

Usage

```
association_matrix(data, vars = NULL)
```

Arguments

`data`	A data frame.
`vars`	Optional subset of variables.

Value

A symmetric matrix with ones on the diagonal.

References

Bergsma, W. (2013). A bias-correction for Cramér's V and Tschuprow's T. *Journal of the Korean Statistical Society*, 42(3), 323–328. doi:10.1016/j.jkss.2012.10.002

Examples

```
round(association_matrix(iris), 2)
```

---

augment_data *Augment a small training set with synthetic data*

Description

Fits a synthesizer to `data` and returns the real records together with synthetic ones, ready to train a machine-learning model.

- For a data frame (or matrix), `n` synthetic rows are appended to the real rows. An indicator column can be added with `label`. A vector is extended with `n` synthetic values.
- For a time series, `nsim` synthetic series of length `n` are drawn (see `fit_synthesizer.ts()`) and returned in a list, preceded by the real series. Use `ts_windows()` to turn the list into a table of lagged features.

Synthetic data cannot add information that is not in the real data. What they add is a smoothed version of the training distribution, which acts as a regulariser and mainly helps flexible, high-variance learners (trees, nearest neighbours, neural networks) trained on small samples. Always **fit the synthesizer on the training data only** and measure the effect on held-out *real* data; see `vignette("augmentation")`.

Usage

```
augment_data(data, ...)

## S3 method for class 'data.frame'
augment_data(
  data,
  n = 10L * nrow(data),
  keep_real = TRUE,
  label = NULL,
  closeness = 1,
  perturb = NULL,
  seed = NULL,
  ...
)

## Default S3 method:
augment_data(data, ...)

## S3 method for class 'matrix'
augment_data(data, ...)

## S3 method for class 'ts'
augment_data(
```

```
  data,
  nsim = 20L,
  n = NROW(data),
  keep_real = TRUE,
  seed = NULL,
  ...
)
```

Arguments

data	Training data: a data frame, matrix or ts object.
...	Passed to fit_synthesizer() (e.g. by, copula, or for time series method, order).
n	For tabular data: number of synthetic rows (default: 10 times the training size). For time series: length of each synthetic series (default: the length of data).
keep_real	Include the real data in the result?
label	Optional name of a logical column marking synthetic rows (tabular data only).
closeness, perturb	How close the synthetic rows are to the real ones (tabular data only); see generate(). Lower closeness gives more diverse synthetic rows.
seed	Optional random seed.
nsim	For time series: number of synthetic series.

Value

A data frame (tabular data) or a list of ts objects (time series).

See Also

ts_windows(), fit_synthesizer()

Examples

```
train <- mtcars[1:20, ]
big <- augment_data(train, n = 500, label = "synthetic", seed = 1)
table(big$synthetic)

series <- augment_data(ldeaths, nsim = 5, order = 12, seed = 1)
length(series)
head(ts_windows(series, lags = 12))
```

autoplot.sp_comparison

Plot a comparison of real and synthetic data

Description

Plot a comparison of real and synthetic data

Usage

```
## S3 method for class 'sp_comparison'
autoplot(
  object,
  type = c("marginals", "association", "qq", "pairs", "dcr"),
  ...
)

## S3 method for class 'sp_comparison'
plot(x, type = c("marginals", "association", "qq", "pairs", "dcr"), ...)
```

Arguments

object, x	An sp_comparison from compare_synthetic().
type	What to plot: "marginals", "association", "qq", "pairs" or "dcr".
...	Passed to the underlying plot function.

Value

autoplot() returns a ggplot (or sp_plot_list); plot() prints it and returns it invisibly.

Examples

```
syn <- synthesize(iris, seed = 1)
cmp <- compare_synthetic(iris, syn, seed = 1)
plot(cmp, type = "qq")
```

calibrate_closeness

Calibrate the closeness of synthetic to real data

Description

Finds the value of `closeness` (see `generate()`) for which synthetic data reach a target value of a distance metric. The metric is evaluated on a grid of closeness values with common random numbers, the grid interval containing the target is located, and the solution is refined by bisection, followed by a final linear interpolation step. Metric values are themselves estimates, so the achieved value varies somewhat with the random seed.

Available metrics (all computed against `data`, which should be the data the synthesizer was fitted on):

- `"auc"` – cross-validated discriminator AUC (`discriminator_auc()`); 0.5 means indistinguishable, larger values mean more distant;
- `"dcr"` – median distance to the closest real record relative to the real-to-real nearest-neighbour distance (`dcr()`); larger means further from real records;
- `"pmse"` – pMSE ratio (`pmse()`) with the quadratic propensity model;
- `"marginal"` – mean marginal distance (KS / TVD, `marginal_metrics()`);
- a **function** `function(real, synthetic)` returning a single number, for any other criterion, e.g. a disclosure-risk measure from `disclosure_risk()`.

Usage

```
calibrate_closeness(
  object,
  data,
  target,
  metric = c("auc", "dcr", "pmse", "marginal"),
  n = NULL,
  grid = seq(0, 1, by = 0.1),
  refine = 4L,
  max_rows = 2000L,
  seed = 1,
  ...
)
```

Arguments

`object`	A fitted `sp_synthesizer`.
`data`	The real data the synthesizer was fitted on.
`target`	Target value of the metric.
`metric`	One of `"auc"`, `"dcr"`, `"pmse"`, `"marginal"`, or a function `function(real, synthetic)` returning a number.
`n`	Number of synthetic records generated per evaluation (default: size of `data`).
`grid`	Closeness values evaluated first.
`refine`	Number of bisection steps after the grid search.
`max_rows`	Maximum rows used by the metric computations.
`seed`	Random seed (the same seed is used at every closeness value).

... Passed to generate() (e.g. perturb for directional perturbations kept fixed during calibration).

Value

An object of class sp_calibration with elements closeness (the calibrated value), achieved (metric value at that closeness), target, metric and path (all evaluations). It has print() and plot() methods.

Examples

```
fit <- fit_synthesizer(iris)
cal <- calibrate_closeness(fit, iris, target = 0.75, metric = "auc",
                           grid = seq(0, 1, 0.25), refine = 2)
cal
syn <- generate(fit, closeness = cal$closeness, seed = 1)
```

---

census_sim *Simulated census-like microdata*

Description

A **simulated** data set of 5000 persons with the structure of census or household-survey microdata, used to illustrate synthetic data for statistical disclosure control. The records were generated with r_mvdist() (Gaussian copula with plausible correlations, see data-raw/census_sim.R in the package sources); they do not describe real people. Region names are the eight development regions of Romania; their shares only roughly follow the population distribution.

Usage

```
census_sim
```

Format

A data frame with 5000 rows and 8 columns:

person_id Direct identifier (character).

age Age in years, 18–90 (integer).

sex Factor: male, female.

region Factor: development region (8 levels).

education Ordered factor: primary < lower secondary < upper secondary < tertiary.

employment Factor: employed, unemployed, inactive, retired.

household_size Number of persons in the household (integer).

income Monthly income in national currency (numeric).

Source

Simulated with SynthesizerPlus.

Examples

```
str(census_sim)
table(census_sim$employment, census_sim$sex)
```

---

ci_overlap *Confidence-interval overlap of regression coefficients*

Description

A specific utility measure: the same regression model is fitted to the real and the synthetic data and, for each coefficient, the overlap of the two confidence intervals is computed (Karr et al., 2006):

$$IO = \frac{1}{2}\left(\frac{U-L}{u_r - l_r} + \frac{U-L}{u_s - l_s}\right)$$

where $[L, U]$ is the intersection of the intervals. Overlap is 1 for identical intervals and can become negative when they are disjoint.

Usage

```
ci_overlap(real, synthetic, formula, family = stats::gaussian(), level = 0.95)
```

Arguments

`real`, `synthetic`	Data frames.
`formula`	Model formula.
`family`	A `stats::family()` object (default Gaussian, i.e. linear regression).
`level`	Confidence level.

Value

A data frame with estimates, intervals, the standardised difference of the estimates and the interval overlap per coefficient.

References

Karr, A. F., Kohnen, C. N., Oganian, A., Reiter, J. P. and Sanil, A. P. (2006). A framework for evaluating the utility of data altered to protect confidentiality. *The American Statistician*, 60(3), 224–232. doi:10.1198/000313006X124640

Examples

```
syn <- synthesize(mtcars, seed = 1)
ci_overlap(mtcars, syn, mpg ~ wt + hp)
```

compare_synthetic — *Compare synthetic data with the real data*

Description

`compare_synthetic()` evaluates the fidelity, utility and disclosure risk of a synthetic data set in one call:

- **marginal**: `marginal_metrics()` per variable;
- **association**: `association_matrix()` of both data sets and their difference;
- **utility**: `pmse()` and cross-validated `discriminator_auc()`;
- **privacy**: `dcr()` distance to closest record.

The result has `print()`, `summary()`, `plot()` and `ggplot2::autoplot()` methods.

Usage

```
compare_synthetic(
  real,
  synthetic,
  vars = NULL,
  metrics = c("marginal", "association", "utility", "privacy"),
  max_rows = 5000L,
  seed = NULL
)
```

Arguments

`real`, `synthetic`	Data frames (or objects coercible to data frames) with common columns.
`vars`	Optional subset of variables.
`metrics`	Which groups of metrics to compute.
`max_rows`	Maximum rows per data set for the classifier and distance computations.
`seed`	Optional random seed.

Value

An object of class `sp_comparison`.

Examples

```
syn <- synthesize(iris, seed = 1)
cmp <- compare_synthetic(iris, syn, seed = 1)
cmp
plot(cmp, type = "association")
```

copula_correlation *Inspect a fitted synthesizer*

Description

`copula_correlation()` returns the (latent) copula correlation matrix of a fitted model, or with `what = "n_pair"` the number of rows on which each pairwise estimate is based; `summary()` returns a table describing how each column is modelled.

Usage

```
copula_correlation(
  object,
  stratum = NULL,
  include_missing = FALSE,
  what = c("correlation", "n_pair")
)

## S3 method for class 'sp_synthesizer'
summary(object, ...)
```

Arguments

`object`	A fitted `sp_synthesizer`.
`stratum`	Optional stratum label for stratified models (as shown by `summary()`); defaults to the pooled model.
`include_missing`	Include the missingness indicators in the matrix?
`what`	`"correlation"` (default) or `"n_pair"` (an integer matrix of jointly observed rows; the diagonal is the number of observed values).
`...`	Unused.

Details

A pairwise correlation is treated as *unidentified* when the two variables are observed together in fewer than 3 rows, or when its estimate is not finite (a numerical safeguard). Such entries are not estimated from data but completed as described in the "Estimation details" section of `fit_synthesizer()`; `n_pair` alone therefore does not identify all of them.

Value

`copula_correlation()`: a correlation matrix (or a count matrix for `what = "n_pair"`). `summary()`: an object of class `summary.sp_synthesizer` (a list with a `columns` data frame).

Examples

```
fit <- fit_synthesizer(airquality)
round(copula_correlation(fit), 2)
summary(fit)
```

`dcr` *Distance to closest record (disclosure risk)*

Description

For every synthetic record (in a subsample of at most `max_rows`), the Gower distance to the closest real record is computed. As a baseline the same is done for real records with respect to the *other* real records. Synthetic records that are much closer to real records than real records are to each other, and in particular exact copies (distance 0), indicate a disclosure risk.

Usage

```
dcr(
  real,
  synthetic,
  vars = NULL,
  max_rows = 2000L,
  seed = NULL,
  close_quantile = 0.05,
  na = c("category", "exclude")
)
```

Arguments

`real`, `synthetic`	Data frames with common columns.
`vars`	Optional subset of variables.
`max_rows`	Maximum number of records used from each data set.
`seed`	Optional random seed (subsampling).
`close_quantile`	Quantile of the real-to-real distances that defines a synthetic record as "too close" to a real record (see `close_share`).
`na`	Treatment of missing values: `"category"` (default) or `"exclude"` (standard Gower); see Details.

Details

Numeric variables contribute their absolute difference divided by the range in the real data, categorical variables 0 (equal) or 1 (different). Missing values are handled according to `na`:

- `"category"` (default): missingness is treated as a category of its own – two missing values match, a missing and an observed value differ – and the distance is averaged over all variables. Because the synthesizer reproduces missingness patterns, and an intruder sees them, this is the appropriate view for disclosure risk.

- "exclude": the standard Gower (1971) coefficient; a variable missing in either record is left out of the comparison and the distance is averaged over the variables observed in both. Records with no variable in common have no distance (NA) and are ignored.

Value

A list with the distance vectors synthetic and real, their quantiles, the share of synthetic records that are exact copies of a real record on *all* variables (exact_match_rate), the share at distance zero on the variables compared (zero_distance_rate; with na = "exclude" this can include records that agree with a real record only on the few variables observed in both), the average share of variables on which the nearest real record was compared (compared_share; always 1 with na = "category"), the share of exact duplicates within the real data (real_duplicate_rate) and ratio, the median synthetic distance divided by the median real-to-real distance (values well below 1 are a warning sign), and close_share, the share of synthetic records that are closer to a real record than close_quantile (5\ are to their nearest real neighbour. For synthetic records that are as far from real ones as real records are from each other, close_share is about close_quantile; larger values indicate records that may be recognised as near-copies of real persons.

References

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 27(4), 857–871. doi:10.2307/2528823

Examples

```
syn <- synthesize(iris, seed = 1)
str(dcr(iris, syn, seed = 1)[c("exact_match_rate", "ratio")])

# data with missing values: missingness as a category, or standard Gower
aq <- synthesize(airquality, seed = 1)
dcr(airquality, aq, seed = 1)$ratio
dcr(airquality, aq, seed = 1, na = "exclude")$ratio
```

---

disclosure_risk *Disclosure risk of synthetic microdata*

Description

Measures the two classical disclosure risks of a synthetic version of confidential microdata (census, survey, register data), from the point of view of an intruder who knows the *key variables* (quasi-identifiers such as age, sex and region) of a person in the real data.

- **Identity disclosure – replicated uniques.** Records whose combination of key variables is unique in the real data are the most exposed. The share of these real uniques whose key combination also occurs exactly once in the synthetic data (replicated_uniques, Nowok, Raab and Dibben, 2016) indicates how many synthetic records could be taken for a real, identifiable person.

- **Attribute disclosure – correct attribution probability (CAP).** For every real record, the intruder looks up the synthetic records with the same key values and predicts each `target` (sensitive) variable: the most frequent category, or the median for numeric targets. `cap` is the share of real records for which the prediction is correct (numeric targets: within `tolerance` relative error or `abs_tolerance` absolute error), among those with at least one matching synthetic record. `cap_baseline` is the same share when the intruder ignores the synthetic data and predicts the overall mode or median; `cap_uniques` restricts the CAP to real uniques. A CAP close to the baseline means that the synthetic data reveal little beyond what is known from population-level statistics (Taub et al., 2018).

Numeric key variables are matched exactly; coarsen them first (e.g. age in years or 5-year bands) if they are continuous.

Usage

```
disclosure_risk(
  real,
  synthetic,
  keys,
  target = NULL,
  ignore = NULL,
  tolerance = 0.1,
  abs_tolerance = 0
)
```

Arguments

`real`, `synthetic`	Data frames with the key and target variables.
`keys`	Character vector of key variables (quasi-identifiers).
`target`	Character vector of sensitive variables for attribute disclosure (optional).
`ignore`	Optional columns to leave out entirely, typically direct identifiers (which differ between real and synthetic files and would otherwise hide exact copies).
`tolerance`	Relative tolerance for a correct prediction of numeric targets (default 0.1, i.e. 10 percent).
`abs_tolerance`	Absolute tolerance for numeric targets (default 0). A prediction is correct if $\lvert\hat{y} - y\rvert \leq \max(\text{tolerance} \cdot \lvert y\rvert, \text{abs_tolerance})$. A purely relative tolerance is (essentially) zero for a true value of 0, so use `abs_tolerance` for targets with meaningful zeros (e.g. income of non-earners), in the units of the target.

Value

An object of class `sp_disclosure` with elements `replicated_uniques`, `real_unique_rate` (share of real records that are unique on the keys), `exact_copies` (share of synthetic records that are identical to some real record on all common variables except `ignore`) and a data frame `attribute` with, per target, `match_rate`, `cap`, `cap_baseline` and `cap_uniques`.

References

Nowok, B., Raab, G. M. and Dibben, C. (2016). synthpop: Bespoke creation of synthetic data in R. *Journal of Statistical Software*, 74(11), 1–26. doi:10.18637/jss.v074.i11

Taub, J., Elliot, M., Pampaka, M. and Smith, D. (2018). Differential correct attribution probability for synthetic data: an exploration. In *Privacy in Statistical Databases*, LNCS 11126, 122–137. Springer. doi:10.1007/9783319997711_9

Examples

```
syn <- synthesize(mtcars, seed = 1)
disclosure_risk(mtcars, syn, keys = c("cyl", "gear", "am"), target = "mpg")

# census-like microdata with a direct identifier
fit <- fit_synthesizer(census_sim, id_cols = "person_id", by = "employment")
syn <- generate(fit, seed = 1)
disclosure_risk(census_sim, syn, keys = c("age", "sex", "region"),
                target = c("income", "education"), ignore = "person_id")
```

`discriminator_auc` *Discriminator AUC*

Description

Cross-validated area under the ROC curve of a logistic-regression classifier trained to tell real from synthetic records. Out-of-fold predictions are used, and all preprocessing (imputation, scaling, choice of categories) is learned within the training folds, so the AUC is not optimistically biased. Values near 0.5 mean the classifier cannot distinguish the data sets; values near 1 mean the synthetic data are easy to spot.

Usage

```
discriminator_auc(
  real,
  synthetic,
  vars = NULL,
  folds = 5L,
  max_rows = 10000L,
  seed = NULL,
  model = c("quadratic", "linear")
)
```

Arguments

`real`, `synthetic`	Data frames with common columns.
`vars`	Optional subset of variables.
`folds`	Number of cross-validation folds.

`max_rows`	Each data set is subsampled to at most this many rows.
`seed`	Optional random seed (subsampling).
`model`	Classifier features: `"quadratic"` (default) or `"linear"`.

Details

Values clearly *below* 0.5 are a warning sign of a different kind: they arise when synthetic records are copies of real ones. A held-out record then has an identical twin with the opposite label in the training folds, and the classifier systematically predicts the wrong class. Check such cases with `dcr()`. The AUC is therefore deliberately not folded into a symmetric measure such as $\max(AUC, 1 - AUC)$: that would report copies of the real data as easily distinguishable.

The default `model = "quadratic"` adds squares and pairwise products of the numeric variables (ridge-penalised), so that differences in spread and in correlation are detected, not only differences in location. `model = "linear"` uses main effects only.

Value

The cross-validated AUC (a number).

Examples

```
syn <- synthesize(iris, seed = 1)
discriminator_auc(iris, syn, seed = 1)
```

`fit_synthesizer` *Fit a synthetic-data model*

Description

`fit_synthesizer()` learns a generative model from a data set. The model combines

- a **marginal model per column**: an *interpolated* empirical quantile function for continuous variables (linear or monotone-spline interpolation between order statistics), and an empirical probability table for discrete, categorical, ordinal and logical variables. Interpolation means that synthetic values of a continuous variable lie between, rather than on, the observed values, always within the observed range. With small samples this slightly reduces the spread in the tails compared with the data;
- a **copula** (Gaussian, Student-*t* or independence) that couples the columns. Discrete and categorical variables take part in the dependence structure: their pseudo-observations use the distributional transform, and their latent correlations are estimated by maximum likelihood (polyserial / polychoric), see *Estimation details*;
- an optional **missing-data model**: for each column with missing values a missingness indicator is added to the copula, so that the rate of missingness *and* its (monotone) association with other variables are reproduced;
- optional **stratification** (`by`), fitting separate models within groups, with a pooled fallback for small groups.

Use `generate()` to draw synthetic data from the fitted model, or `synthesize()` to fit and generate in one step.

Usage

```
fit_synthesizer(data, ...)

## S3 method for class 'data.frame'
fit_synthesizer(
  data,
  copula = c("gaussian", "t", "independence"),
  by = NULL,
  min_stratum_size = 20L,
  missing = c("joint", "independent", "drop"),
  interpolation = c("linear", "spline"),
  discrete_threshold = 20L,
  knots = NULL,
  id_cols = NULL,
  pd_method = c("auto", "higham", "eigen", "none"),
  seed = NULL,
  ...
)

## S3 method for class 'matrix'
fit_synthesizer(data, ...)

## Default S3 method:
fit_synthesizer(data, ...)
```

Arguments

`data` A data frame, matrix, atomic vector (numeric, integer, logical, character, factor, `Date`, `POSIXct`) or `ts` object. Time series are handled by `fit_synthesizer.ts()`.

`...` Further arguments passed to methods.

`copula` Dependence model: `"gaussian"` (default), `"t"` (heavier joint tails) or `"independence"`. See *Estimation details*.

`by` Optional character vector of column names used to stratify the model. Each stratum gets its own marginals and copula; strata are sampled in proportion to their training size.

`min_stratum_size` Strata with fewer rows than this use a pooled model fitted on all rows.

`missing` How missing values are treated: `"joint"` (default; model missingness inside the copula), `"independent"` (reproduce the missing rate of each column independently) or `"drop"` (do not reproduce missingness: partially observed columns are generated without missing values; a column with *no* observed training value has no distribution to draw from and necessarily stays entirely missing). Under `"joint"`, missingness may depend on the other variables (missing at random); it cannot depend on the unobserved value itself, which the data do not identify (see *Estimation details*).

`interpolation` Interpolation of the empirical quantile function for continuous variables: `"linear"` (default) or `"spline"` (monotone Hyman spline, smoother densities).

`discrete_threshold`
: Numeric columns with at most this many distinct values are treated as discrete (sampled from their observed support), provided their values are whole numbers or actually repeat (fewer distinct values than half the observations). A small sample of a continuous variable is therefore still modelled as continuous.

`knots`
: Optional maximum number of quantile knots stored per continuous column. `NULL` (default) keeps every observation; a smaller value shrinks the model and avoids storing the original values.

`id_cols`
: Optional character vector of identifier columns. They are not modelled; synthetic data get fresh sequential identifiers.

`pd_method`
: How to repair a correlation matrix that is not positive definite: `"auto"`/`"higham"` (nearest correlation matrix), `"eigen"` (eigenvalue flooring) or `"none"` (fail if not positive definite).

`seed`
: Optional random seed for the fit. Fitting is stochastic (random tie-breaking and the randomised distributional transform used for the pseudo-observations, row subsampling for the *t* copula), so a seed makes the fitted model reproducible when fitting and generation are separate steps. The global random number generator state is restored afterwards. (`synthesize()` has its own `seed`, covering both steps.)

Value

An object of class `sp_synthesizer`.

Estimation details

Latent correlations. For a pair of continuous variables the Gaussian copula uses the normal scores $\Phi^{-1}(U)$ and the t copula inverts Kendall's tau, $\rho = \sin(\pi\tau/2)$. As Kendall's tau is quadratic in the sample size, it is computed on a random subsample of 2000 rows; a pair that the subsample leaves with fewer than $\min(n_{ij}, 500)$ jointly observed rows is re-estimated from up to 2000 of its own n_{ij} joint rows, so every pair with $n_{ij} \geq 3$ is estimated whatever the subsample.

Discrete and categorical variables. The pseudo-observations of a discrete margin fill each category's probability interval with independent uniform noise (the distributional transform). Correlating these jittered scores would bias every dependence involving a discrete variable towards zero, most strongly for binary variables. Pairs with at least one discrete margin are therefore estimated by maximum likelihood under the Gaussian copula, with the thresholds fixed at the normal quantiles of the marginal cumulative probabilities: the polyserial correlation for a discrete-continuous pair (Olsson, Drasgow and Dorans, 1982) and the polychoric correlation for a discrete-discrete pair (Olsson, 1979), using the bivariate normal distribution function of Genz (2004). The t copula uses the same estimates for these pairs. Unordered categories enter in the order of their levels, which is also the order used when generating, so the estimate is the maximum likelihood estimate of the model actually fitted.

Missing data. All estimates use **pairwise complete observations**. The rows in which two variables are both observed may be a selected subset (e.g. `y` is observed mainly for low `x`), with a restricted range. For continuous pairs the estimate is therefore the pairwise *covariance* of the normal scores, scaled by each variable's standard deviation over all of its observed values (for the t copula, the tau-based correlation rescaled in the same way). This reproduces the relation among the observed

rows under such selection; with complete data it equals the ordinary correlation. With `missing = "joint"`, the missingness indicators are binary variables of the copula, so missingness can depend on the other variables (missing at random). The latent correlation between a variable and its *own* indicator is fixed at 0: the data do not identify it (the value is never seen when missing), and this choice keeps the distribution of the observed values of each variable as in the training data. Pairwise estimates based on different subsets of rows need not form a coherent joint estimate, and the resulting matrix need not be positive definite; it is then projected onto the set of valid correlation matrices according to `pd_method` (Higham's nearest correlation matrix by default). The projection guarantees a valid model, not an optimal joint estimator.

Unidentified pairs. If two variables are observed together in fewer than 3 rows (e.g. questions from different survey modules), their correlation cannot be estimated (with 2 rows it is trivially ± 1); the same applies when the pairwise estimate is not finite. Such entries are not set to zero, which would assert independence and, through the projection, distort the identified entries. They are filled by the maximum-determinant positive-definite completion of the identified entries (Dempster, 1972), which keeps every identified correlation and makes the pair conditionally independent given the other variables. If the identified entries cannot be completed, the unknown entries fall back to 0. In either case the matrix may then still be modified by the positive-definiteness repair. A warning reports how many pairs were affected and how they were filled; `copula_correlation(fit, what = "n_pair")` gives the pairwise counts, and the fitted copula records the method (`completion`: `"none"`, `"maxdet"` or `"zero_fallback"`) and the largest change made by the repair (`pd_adjustment`), both shown by `summary()`. Pair counts and identification always refer to the full data.

t-copula degrees of freedom. With the correlation matrix held fixed, `df` is chosen by profile likelihood over the grid 2, 3, 4, 5, 6, 8, 10, 15, 20, 30, 50, 100 using the complete rows (at most 5000), so it is a discrete grid estimate rather than a continuous maximum likelihood estimate. If fewer than 10 complete rows are available, `df` is **not estimated** and is fixed at 10; a warning is issued and the fitted copula stores `df_estimated = FALSE` (see `summary()`).

References

Sklar, A. (1959). Fonctions de répartition à n dimensions et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris*, 8, 229–231.

Dempster, A. P. (1972). Covariance selection. *Biometrics*, 28(1), 157–175. doi:10.2307/2528966

Genz, A. (2004). Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing*, 14(3), 251–260. doi:10.1023/B:STCO.0000035304.20635.31

Olsson, U. (1979). Maximum likelihood estimation of the polychoric correlation coefficient. *Psychometrika*, 44(4), 443–460. doi:10.1007/BF02296207

Olsson, U., Drasgow, F. and Dorans, N. J. (1982). The polyserial correlation coefficient. *Psychometrika*, 47(3), 337–347. doi:10.1007/BF02294164

Rüschendorf, L. (2009). On the distributional transform, Sklar's theorem, and the empirical copula process. *Journal of Statistical Planning and Inference*, 139(11), 3921–3927. doi:10.1016/j.jspi.2009.05.030

Higham, N. J. (2002). Computing the nearest correlation matrix – a problem from finance. *IMA Journal of Numerical Analysis*, 22(3), 329–343. doi:10.1093/imanum/22.3.329

See Also

`generate()`, `synthesize()`, `compare_synthetic()`

Examples

```
fit <- fit_synthesizer(iris)
fit
syn <- generate(fit, n = 300, seed = 1)
head(syn)

# Stratified model and missing-data model
fit_aq <- fit_synthesizer(airquality, by = "Month")
colSums(is.na(generate(fit_aq, seed = 2)))
```

`fit_synthesizer.ts` *Fit a synthesizer to a (multivariate) time series*

Description

Methods for `ts` objects. Four generators are available:

- `"copula_var"` (default): each series is mapped to normal scores through its empirical marginal distribution; a vector autoregression (order chosen by AIC) is fitted to the scores, and simulated paths are mapped back through the marginal quantile functions. This reproduces the marginal distributions *and* the auto- and cross-correlation structure. Innovations are resampled from the fitted residuals, so their joint distribution is preserved.
- `"block"`: moving (circular) block bootstrap of the observed series.
- `"stationary"`: stationary bootstrap (Politis and Romano, 1994) with geometrically distributed block lengths.
- `"iid"`: ignores serial dependence; rows are drawn from a cross-sectional copula model (see `fit_synthesizer()`).

Missing values are linearly interpolated before fitting (with a warning).

Scope. All four methods treat the series as (approximately) **stationary**: the empirical marginal distribution and the dependence structure are assumed constant over time. A trend, a level shift, changing variance or strong deterministic seasonality is reproduced only imperfectly (e.g. a trending series is resampled around its overall distribution, and a seasonal pattern only as far as the VAR order or block length covers it). For such data, model the deterministic part yourself (detrend, difference or deseasonalise), synthesise the stationary remainder, and add the deterministic part back. Arguments that the chosen method does not use trigger a warning.

Usage

```
## S3 method for class 'ts'
fit_synthesizer(
  data,
  method = c("copula_var", "block", "stationary", "iid"),
  max_lag = 10L,
  order = NULL,
  block_length = NULL,
```

```
  burn_in = 100L,
  seed = NULL,
  ...
)
```

Arguments

data	A ts object (univariate or multivariate).
method	One of "copula_var", "block", "stationary", "iid".
max_lag	Maximum VAR order considered by "copula_var" when the order is selected by AIC.
order	Optional fixed VAR order for "copula_var" (skips AIC selection). Useful for seasonal data on short series, where AIC tends to choose an order below the seasonal period, e.g. order = 12 for monthly data.
block_length	Block length ("block") or mean block length ("stationary"). Default: ceiling(n^(1/3)).
burn_in	Number of burn-in steps discarded when simulating a VAR.
seed	Optional random seed making the fit reproducible (see fit_synthesizer()).
...	Passed to fit_synthesizer.data.frame() when method = "iid".

Value

An object of class sp_ts_synthesizer. Use generate() to draw synthetic series of any length; the result is a ts with the same frequency (and, by default, the same start) as data.

References

Politis, D. N. and Romano, J. P. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89(428), 1303–1313. doi:10.1080/01621459.1994.10476870

Examples

```
fit <- fit_synthesizer(ldeaths)
syn <- generate(fit, seed = 1)
plot_ts(ldeaths, syn)

# window() keeps the ts class (row-subsetting with [1:500, ] would not)
eu <- window(EuStockMarkets, end = time(EuStockMarkets)[500])
fit_mv <- fit_synthesizer(eu, method = "stationary")
head(generate(fit_mv, n = 100, seed = 1))
```

generate *Generate synthetic data from a fitted model*

Description

`generate()` draws synthetic records from a fitted model such as an `sp_synthesizer` (see `fit_synthesizer()`) or an `sp_ts_synthesizer` (see `fit_synthesizer.ts()`).

Usage

```
generate(object, ...)

## S3 method for class 'sp_synthesizer'
generate(
  object,
  n = object$n_train,
  seed = NULL,
  dependence = 1,
  closeness = 1,
  perturb = NULL,
  stratum_sizes = c("multinomial", "proportional"),
  ...
)

## S3 method for class 'sp_ts_synthesizer'
generate(object, n = object$n_train, seed = NULL, start = object$start, ...)
```

Arguments

`object`	A fitted model.
`...`	Further arguments passed to methods.
`n`	Number of records (rows) to generate. Defaults to the number of rows in the training data.
`seed`	Optional random seed. The global random number generator state is restored afterwards.
`dependence`	Number in [0, 1] scaling the strength of the dependence between variables: 1 (default) reproduces the fitted copula, 0 generates independent columns. Intermediate values shrink the copula correlation matrix towards the identity; the marginal distributions are unaffected.
`closeness`	Number in [0, 1] controlling how close the synthetic data are to the real data: 1 (default) reproduces the fitted model as closely as possible; smaller values move records away from the real ones (noise on continuous variables, random re-drawing of categories) and loosen the distributions (flatter category probabilities, weaker dependence). With `closeness = 0` the variables are independent and strongly smoothed. See `perturbation()` for the exact mapping and `calibrate_closeness()` to choose a value that reaches a target distance.

perturb	Optional perturbation() for fine control: record-level noise and swapping, and distribution-level shifts, spread, category temperature and dependence strength. Combined with closeness and dependence.
stratum_sizes	For stratified models: "multinomial" (default; random stratum sizes) or "proportional" (deterministic sizes proportional to the training data).
start	Start time of the generated series (default: the start of the training series).

Value

Synthetic data of the same type as the training data: a data frame, matrix, vector or ts object.

See Also

fit_synthesizer(), synthesize(), generate_to_file()

Examples

```
fit <- fit_synthesizer(mtcars)
generate(fit, n = 5, seed = 42)

# weaker dependence
syn <- generate(fit, n = 500, dependence = 0.5, seed = 1)
```

---

generate_to_file *Generate a large synthetic data set directly to disk*

Description

Generates n synthetic rows in chunks of chunk_size, so that data sets much larger than memory can be produced. For appendable formats (csv, tsv, ndjson, optionally compressed) all chunks go to one file. For other formats (Parquet, Feather, HDF5, fst, RDS, ...) a directory path is created holding one part file per chunk (part-00001.parquet, ...), which read_data() (or e.g. arrow::open_dataset()) reads as a whole.

Usage

```
generate_to_file(
  object,
  n,
  path,
  format = NULL,
  chunk_size = 1e+05,
  seed = NULL,
  progress = interactive(),
  ...
)
```

Arguments

object	A fitted sp_synthesizer.
n	Total number of rows.
path	Output file (appendable formats) or directory (other formats).
format	Format name; inferred from path for appendable formats. Required when path is a directory name without extension.
chunk_size	Rows per chunk.
seed	Optional random seed (the whole output is reproducible).
progress	Print progress messages?
...	Passed to generate() (e.g. dependence).

Value

Invisibly, a list with the path, format, number of rows and the files written.

Examples

```
fit <- fit_synthesizer(iris)
out <- tempfile(fileext = ".csv")
generate_to_file(fit, n = 1000, path = out, chunk_size = 300, seed = 1)
nrow(read_data(out))

dir <- tempfile()
generate_to_file(fit, n = 1000, path = dir, format = "rds", chunk_size = 400)
list.files(dir)
```

---

make_corr *Build a correlation matrix with a given structure*

Description

Build a correlation matrix with a given structure

Usage

```
make_corr(
  d,
  type = c("exchangeable", "ar1", "toeplitz", "random", "identity"),
  rho = 0.5,
  names = NULL
)
```

Arguments

d	Dimension.
type	Structure: "exchangeable" (all off-diagonal entries rho), "ar1" (rho^|i-j|), "toeplitz" (first row given by rho, a vector of length d - 1), "random" (a random positive-definite correlation matrix) or "identity".
rho	Correlation parameter (see type).
names	Optional variable names.

Value

A d x d correlation matrix.

Examples

```
make_corr(4, "ar1", rho = 0.7)
make_corr(3, "toeplitz", rho = c(0.5, 0.2))
```

---

margin_categorical *Marginal quantile functions for r_mvdist()*

Description

margin_categorical() builds a quantile function for a categorical variable; margin_empirical() builds the (interpolated) empirical quantile function of observed data, preserving its class.

Usage

```
margin_categorical(levels, probs = NULL, ordered = FALSE)

margin_empirical(x, interpolation = c("linear", "spline"))
```

Arguments

levels	Category labels.
probs	Category probabilities (normalised internally).
ordered	Return an ordered factor?
x	Observed data (numeric, integer, Date, POSIXct, factor, character, logical).
interpolation	"linear" or "spline".

Value

A function of p returning values.

Examples

```
q <- margin_categorical(c("low", "mid", "high"), c(0.2, 0.5, 0.3), ordered = TRUE)
table(q(runif(1000)))
qe <- margin_empirical(faithful$eruptions)
summary(qe(runif(1000)))
```

`marginal_metrics` *Compare marginal distributions of real and synthetic data*

Description

For each variable, `marginal_metrics()` reports

- for numeric variables (incl. dates): the Kolmogorov–Smirnov distance between the empirical CDFs and the Wasserstein-1 distance scaled by the standard deviation of the real data;
- for categorical variables: the total variation distance between the category proportions;
- means, standard deviations and missing-value rates of both data sets.

Usage

```
marginal_metrics(real, synthetic, vars = NULL)
```

Arguments

`real`, `synthetic`	Data frames with common columns.
`vars`	Optional subset of variables.

Details

All distances are 0 for identical distributions; KS and TVD are bounded by 1.

Value

A data frame with one row per variable.

Examples

```
syn <- synthesize(iris, seed = 1)
marginal_metrics(iris, syn)
```

match_types — *Coerce the columns of a data frame to the classes of a template*

Description

Useful after reading formats that do not store R classes (CSV, JSON, Excel): factors regain their levels (and ordering), dates and date-times are parsed, and integers/logicals are restored. Columns not present in the template are left unchanged.

Usage

```
match_types(x, template)
```

Arguments

`x`	A data frame.
`template`	A data frame with the desired column classes.

Details

Date-time strings ending in `Z` are read as UTC, strings with an explicit offset (e.g. `+02:00`) are converted from that offset, and strings without zone information are read as local times in the time zone of the template column.

Value

`x` with coerced columns.

Examples

```
f <- tempfile(fileext = ".csv")
write.csv(iris, f, row.names = FALSE)
x <- match_types(read.csv(f), iris)
identical(levels(x$Species), levels(iris$Species))
```

mvdist — *Multivariate random number generators*

Description

Random generation for common multivariate distributions. All functions return an `n x d` matrix with column names, and accept an optional `seed`.

- `r_mvnorm()`: multivariate normal $N(\mu, \Sigma)$.
- `r_mvt()`: multivariate Student-*t* with `df` degrees of freedom and scale matrix `sigma`.
- `r_mvlnorm()`: multivariate log-normal: `exp()` of a multivariate normal with parameters `meanlog`, `sigmalog`.
- `r_mvskewnorm()`: multivariate skew-normal of Azzalini and Dalla Valle (1996) with location `xi`, scale matrix `omega` and shape `alpha`.
- `r_dirichlet()`: Dirichlet distribution on the simplex, generated by normalising gamma draws. Draws are computed on the log scale (with the $YU^{1/a}$ boost for shapes below 1), so very small concentration parameters do not underflow to `0/0`; as $\alpha \to 0$ the draws approach the vertices of the simplex.
- `r_mvmixture()`: finite mixture of multivariate normals.

Usage

```
r_mvnorm(n, mean = NULL, sigma, seed = NULL)

r_mvt(n, df, mean = NULL, sigma, seed = NULL)

r_mvlnorm(n, meanlog = NULL, sigmalog, seed = NULL)

r_mvskewnorm(n, xi = NULL, omega, alpha, seed = NULL)

r_dirichlet(n, alpha, seed = NULL)

r_mvmixture(n, weights, means, sigmas, return_component = FALSE, seed = NULL)
```

Arguments

`n`	Number of draws.
`mean`, `xi`	Mean / location vector (finite; defaults to zero).
`sigma`, `omega`	Covariance (scale) matrix: square, finite, symmetric and positive semi-definite. `omega` must also have a strictly positive diagonal, because the skew-normal is parameterised through the correlation matrix $\bar{\Omega} = \omega^{-1}\Omega\omega^{-1}$. `sigma` may be singular (positive semi-definite), giving a degenerate normal or *t* distribution; `omega` must be positive definite, as in the standard definition of the skew-normal density.
`seed`	Optional random seed.
`df`	Degrees of freedom (`Inf` gives the normal distribution).
`meanlog`, `sigmalog`	Mean vector and covariance matrix on the log scale.
`alpha`	Shape (skewness) vector for `r_mvskewnorm()`; concentration vector for `r_dirichlet()`.

`weights`	Mixture weights (normalised internally).
`means`	List of component mean vectors.
`sigmas`	List of component covariance matrices.
`return_component`	Should `r_mvmixture()` add the component label as attribute `"component"`?

Value

A numeric matrix with `n` rows.

References

Azzalini, A. and Dalla Valle, A. (1996). The multivariate skew-normal distribution. *Biometrika*, 83(4), 715–726. doi:10.1093/biomet/83.4.715

Examples

```
S <- make_corr(3, "ar1", 0.6)
x <- r_mvnorm(1000, mean = c(0, 1, 2), sigma = S, seed = 1)
round(cor(x), 2)

t3 <- r_mvt(500, df = 3, sigma = S)
sn <- r_mvskewnorm(500, omega = S, alpha = c(5, 0, -5))
dir <- r_dirichlet(5, alpha = c(1, 2, 3))
rowSums(dir)

mix <- r_mvmixture(500, weights = c(0.3, 0.7),
                   means = list(c(-2, -2), c(2, 2)),
                   sigmas = list(diag(2), diag(2)))
```

`perturbation` *Perturbations of the synthetic data distribution*

Description

`perturbation()` describes how synthetic data should *deviate* from the fitted model. It is passed to `generate()` (argument `perturb`) and has two groups of settings.

Record-level settings move synthetic records away from the real records while (approximately) keeping the distributions:

- `noise` – smoothing of numeric variables. Each value is replaced by $\mu+(x-\mu+hs\varepsilon)/\sqrt{1+h^2}$ with $\varepsilon \sim N(0,1)$, where h is `noise` and μ, s are the mean and standard deviation of the variable in the fitted model. Mean and variance are approximately preserved (exactly only if μ and s matched the generated values exactly; rounding of whole-number variables adds a little variance); the distribution becomes smoother and values move away from observed ones.
- `swap` – probability that a value of a categorical or discrete variable is replaced by an independent draw from its marginal distribution. The marginal distribution is preserved; the association with other variables is weakened for the affected records.

Distribution-level settings deliberately change the distributions, e.g. for stress tests or to simulate a domain shift:

- `shift` – shift of numeric variables, in units of their standard deviation;
- `scale` – multiplier of the spread of numeric variables around their median;
- `temperature` – reshapes the probabilities of categorical and discrete variables, $p_k \propto p_k^{1/T}$: `T > 1` makes them more uniform, `T < 1` concentrates them on the most frequent values;
- `dependence` – strength of the dependence between variables: 0 gives independent variables, 1 the fitted dependence, values above 1 (up to 2) strengthen it.

Numeric variables that only take whole numbers (counts, ages, dates) stay whole numbers; they are rounded stochastically after perturbation, so that shifts and noise are not biased by rounding.

`noise`, `swap`, `shift`, `scale` and `temperature` can be single numbers (applied to every eligible variable) or named vectors that apply to the named variables only.

Usage

```
perturbation(
  noise = 0,
  swap = 0,
  shift = 0,
  scale = 1,
  temperature = 1,
  dependence = 1,
  bounds = c("sign", "observed", "none")
)
```

Arguments

`noise`	Non-negative smoothing bandwidth for numeric variables.
`swap`	Probability in [0, 1] of independent re-drawing for categorical and discrete variables.
`shift`	Shift of numeric variables in standard deviations.
`scale`	Positive spread multiplier for numeric variables.
`temperature`	Positive temperature for categorical and discrete probabilities.
`dependence`	Dependence strength in [0, 2].
`bounds`	How to keep perturbed numeric values in range: `"sign"` (default; variables that are never negative, or never positive, keep that sign), `"observed"` (clip to the observed range) or `"none"`.

Value

An object of class `sp_perturbation`.

See Also

`generate()`, `calibrate_closeness()`

Examples

```
fit <- fit_synthesizer(iris)

# records further from real ones, same distributions
far <- generate(fit, perturb = perturbation(noise = 0.5, swap = 0.2), seed = 1)

# a shifted scenario: longer petals, more uniform species, weaker dependence
p <- perturbation(shift = c(Petal.Length = 1), temperature = 3, dependence = 0.5)
p
scen <- generate(fit, perturb = p, seed = 1)
tapply(scen$Petal.Length, scen$Species, mean)
```

---

plot_association *Plot association matrices of real and synthetic data*

Description

Heatmaps of the mixed-type `association_matrix()` of the real data, the synthetic data and their difference (synthetic minus real), on a common diverging scale.

Usage

```
plot_association(real, synthetic = NULL, vars = NULL, show_values = NULL)
```

Arguments

`real`	A data frame (or vector/matrix) with the real data.
`synthetic`	Optional synthetic data with the same columns.
`vars`	Variables to plot (default: all, up to `max_vars` per kind).
`show_values`	Print the values in the cells? (Default: when there are at most 8 variables.)

Value

A `ggplot` object.

Examples

```
syn <- synthesize(mtcars, seed = 1)
plot_association(mtcars, syn, vars = c("mpg", "cyl", "disp", "hp", "wt"))
```

plot_dcr — *Plot distances to the closest record*

Description

Compares the distribution of distances from synthetic records to their closest real record with the real-to-real baseline (see `dcr()`).

Usage

```
plot_dcr(x)
```

Arguments

x	The result of `dcr()` or `compare_synthetic()`.

Value

A `ggplot` object.

Examples

```
syn <- synthesize(iris, seed = 1)
plot_dcr(dcr(iris, syn, seed = 1))
```

plot_marginals — *Plot marginal distributions of real and synthetic data*

Description

`plot_marginals()` overlays the distributions of numeric variables (density, histogram or empirical CDF) and shows side-by-side category proportions for categorical variables. When the selected variables contain both kinds, a list of two plots (class `sp_plot_list`) is returned; printing it draws both.

Usage

```
plot_marginals(
  real,
  synthetic = NULL,
  vars = NULL,
  kind = c("density", "histogram", "ecdf"),
  max_vars = 12L,
  max_levels = 15L,
  bins = 30L
)

plot_categorical(real, synthetic = NULL, vars = NULL, max_levels = 15L)
```

Arguments

real	A data frame (or vector/matrix) with the real data.
synthetic	Optional synthetic data with the same columns.
vars	Variables to plot (default: all, up to max_vars per kind).
kind	For numeric variables: "density", "histogram" or "ecdf".
max_vars	Maximum number of variables of each kind shown by default.
max_levels	Maximum categories per categorical variable (the rest are pooled into "(other)").
bins	Number of histogram bins.

Value

A ggplot object or an sp_plot_list.

Examples

```
syn <- synthesize(iris, seed = 1)
plot_marginals(iris, syn)
plot_marginals(iris, syn, vars = "Sepal.Length", kind = "ecdf")
```

---

plot_pairs *Scatter-plot matrix of real and/or synthetic data*

Description

Pairwise scatter plots of numeric variables, with real and synthetic records side by side (columns of the panel grid are variables, the colour identifies the source). Useful to inspect multivariate structure, also for data generated with r_mvdist() or mvdist functions.

Usage

```
plot_pairs(
  real,
  synthetic = NULL,
  vars = NULL,
  max_points = 1500L,
  alpha = 0.35
)
```

Arguments

real	A data frame (or vector/matrix) with the real data.
synthetic	Optional synthetic data with the same columns.
vars	Variables to plot (default: all, up to max_vars per kind).
max_points	Maximum number of points drawn per data set.
alpha	Point transparency.

Value

A ggplot object.

Examples

```
syn <- synthesize(iris, seed = 1)
plot_pairs(iris, syn, vars = c("Sepal.Length", "Petal.Length", "Petal.Width"))

x <- as.data.frame(r_copula(1000, "clayton", dim = 3, theta = 3, seed = 1))
plot_pairs(x)
```

---

plot_qq *Quantile-quantile plots of synthetic against real data*

Description

For each numeric variable, quantiles of the synthetic data are plotted against those of the real data; points on the diagonal indicate matching distributions.

Usage

```
plot_qq(
  real,
  synthetic,
  vars = NULL,
  max_vars = 12L,
  probs = seq(0.01, 0.99, by = 0.01)
)
```

Arguments

real	A data frame (or vector/matrix) with the real data.
synthetic	Optional synthetic data with the same columns.
vars	Variables to plot (default: all, up to max_vars per kind).
max_vars	Maximum number of variables of each kind shown by default.
probs	Probabilities at which quantiles are compared.

Value

A ggplot object.

Examples

```
syn <- synthesize(mtcars, seed = 1)
plot_qq(mtcars, syn, vars = c("mpg", "hp", "wt"))
```

plot_ts — *Plot real and synthetic time series*

Description

`plot_ts()` draws the real and synthetic series one above the other for each component; `plot_acf()` compares their autocorrelation functions.

Usage

```
plot_ts(real, synthetic = NULL)

plot_acf(real, synthetic, lag_max = 24L)
```

Arguments

`real`	A `ts` object (or numeric vector/matrix).
`synthetic`	A synthetic `ts` with the same number of series.
`lag_max`	Maximum lag for the autocorrelation function.

Value

A `ggplot` object.

Examples

```
fit <- fit_synthesizer(ldeaths)
syn <- generate(fit, seed = 1)
plot_ts(ldeaths, syn)
plot_acf(ldeaths, syn)
```

pmse — *Propensity-score utility (pMSE)*

Description

The real and synthetic records are stacked and a logistic regression (main effects; categorical variables as dummies, missing values as indicators) predicts membership of the synthetic set. The propensity score mean-squared error

$$pMSE = \frac{1}{N}\sum_i (\hat{p}_i - c)^2, \quad c = n_{syn}/N$$

is 0 when the two sets are indistinguishable. `pmse_ratio` divides it by its expectation under the null hypothesis that both sets come from the same distribution, $(k-1)(1-c)^2 c/N$ with k model parameters; values near 1 indicate high utility.

Usage

```
pmse(
  real,
  synthetic,
  vars = NULL,
  max_rows = 10000L,
  seed = NULL,
  model = c("linear", "quadratic")
)
```

Arguments

`real`, `synthetic`	Data frames with common columns.
`vars`	Optional subset of variables.
`max_rows`	Each data set is subsampled to at most this many rows.
`seed`	Optional random seed (subsampling).
`model`	Propensity model: `"linear"` (main effects, default) or `"quadratic"`.

Details

With `model = "quadratic"`, squares and pairwise products of the numeric variables are added to the logistic model (as recommended by Snoke et al. for detecting differences in the dependence structure).

Value

A list with `pmse`, `pmse_null`, `pmse_ratio` and `n_params`.

References

Snoke, J., Raab, G. M., Nowok, B., Dibben, C. and Slavkovic, A. (2018). General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society A*, 181(3), 663–688. doi:10.1111/rssa.12358

Examples

```
syn <- synthesize(iris, seed = 1)
pmse(iris, syn)
```

r_copula *Sample from a copula*

Description

Draws uniforms on $[0,1]^d$ from elliptical (Gaussian, *t*) or Archimedean (Clayton, Gumbel, Frank) copulas. Archimedean copulas in dimension `d > 2` are sampled with the Marshall–Olkin frailty algorithm; the bivariate Frank copula also allows negative dependence (`theta < 0`), sampled by conditional inversion.

Usage

```
r_copula(
  n,
  family = c("gaussian", "t", "clayton", "gumbel", "frank", "independence"),
  dim = 2L,
  corr = NULL,
  df = 4,
  theta = NULL,
  seed = NULL
)
```

Arguments

`n`	Number of draws.
`family`	Copula family.
`dim`	Dimension (ignored when `corr` is given).
`corr`	Correlation matrix for elliptical copulas (default: exchangeable with `rho = 0.5`). It must have a unit diagonal; a covariance matrix is rejected (convert it with `stats::cov2cor()`).
`df`	Degrees of freedom for the *t* copula: a positive number; `Inf` gives the Gaussian copula.
`theta`	Archimedean parameter: Clayton `theta > 0`, Gumbel `theta >= 1`, Frank `theta != 0` (`> 0` when `dim > 2`).
`seed`	Optional random seed.

Value

An `n x dim` matrix of uniforms.

References

Marshall, A. W. and Olkin, I. (1988). Families of multivariate distributions. *Journal of the American Statistical Association*, 83(403), 834–841. doi:10.1080/01621459.1988.10478671

Hofert, M. (2008). Sampling Archimedean copulas. *Computational Statistics & Data Analysis*, 52(12), 5163–5174. doi:10.1016/j.csda.2008.05.019

Examples

```
u <- r_copula(1000, "clayton", dim = 3, theta = 2, seed = 1)
cor(u, method = "kendall")   # theoretical tau = theta / (theta + 2) = 0.5
```

`r_mvdist` *Generate data from a copula with arbitrary marginal distributions*

Description

`r_mvdist()` combines a copula (see `r_copula()`) with user-specified marginal distributions, returning a data frame. This is a flexible way to simulate mixed-type data sets with known dependence, e.g. for simulation studies or to benchmark synthesizers.

Each element of `margins` is either

- a quantile function `function(p)` (e.g. from `margin_categorical()` or `margin_empirical()`),
- a character string naming a distribution with a `q*` function, e.g. `"norm"`, `"gamma"`, `"pois"`, or
- a list with element `dist` and further parameters, e.g. `list(dist = "gamma", shape = 2, rate = 1)`.

Usage

```
r_mvdist(n, margins, copula = "gaussian", ..., seed = NULL)
```

Arguments

`n`	Number of rows.
`margins`	A (named) list of marginal specifications.
`copula`	Copula family, see `r_copula()`.
`...`	Further arguments passed to `r_copula()` (`corr`, `df`, `theta`).
`seed`	Optional random seed.

Value

A data frame with one column per margin.

Examples

```
df <- r_mvdist(
  500,
  margins = list(
    income = list(dist = "lnorm", meanlog = 10, sdlog = 0.5),
    age = list(dist = "unif", min = 18, max = 80),
    sector = margin_categorical(c("agri", "industry", "services"),
                                c(0.1, 0.3, 0.6))
  ),
  copula = "gaussian", corr = make_corr(3, "exchangeable", 0.4), seed = 1
)
str(df)
```

read_data *Read a data set from a file*

Description

read_data() reads tabular data from many formats commonly used in data science (see supported_formats()) and returns a plain data.frame. The format is inferred from the file extension unless format is given.

If path is a directory (for example as written by generate_to_file()), all files in it with a recognised extension are read and row-bound.

Because text formats (CSV, JSON, Excel) do not store column types, a template data frame can be supplied: columns are then coerced to the classes of the template (factor levels, Date, POSIXct, integer, ...) with match_types().

Usage

```
read_data(
  path,
  format = NULL,
  template = NULL,
  object = NULL,
  dataset = "data",
  sheet = 1L,
  ...
)
```

Arguments

path	File or directory path.
format	Optional format name or extension, e.g. "parquet".
template	Optional data frame whose column classes are imposed on the result.
object	For rdata files: name of the object to load (default: the first data frame in the file).
dataset	For hdf5 files: name of the group (written by write_data()) or of a 1- or 2-dimensional dataset to read. Default "data".
sheet	For Excel files: sheet name or number.
...	Further arguments passed to the backend reader.

Value

A data.frame.

See Also

write_data(), supported_formats(), match_types()

Examples

```
f <- tempfile(fileext = ".csv")
write_data(iris, f)
x <- read_data(f, template = iris)
str(x)

f2 <- tempfile(fileext = ".rds")
write_data(iris, f2)
identical(read_data(f2), iris)
```

---

simulate.sp_synthesizer

Simulate several synthetic data sets

Description

Draws `nsim` independent synthetic data sets from a fitted model, for example to account for synthesis uncertainty with multiple synthetic data sets (Raghunathan, Reiter and Rubin, 2003).

Usage

```
## S3 method for class 'sp_synthesizer'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'sp_ts_synthesizer'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

`object`	A fitted `sp_synthesizer` or `sp_ts_synthesizer`.
`nsim`	Number of synthetic data sets.
`seed`	Optional random seed.
`...`	Passed to `generate()`, e.g. `n`.

Value

A list of `nsim` synthetic data sets.

References

Raghunathan, T. E., Reiter, J. P. and Rubin, D. B. (2003). Multiple imputation for statistical disclosure limitation. *Journal of Official Statistics*, 19(1), 1–16.

Examples

```
fit <- fit_synthesizer(mtcars)
sims <- simulate(fit, nsim = 3, seed = 1)
vapply(sims, function(d) mean(d$mpg), numeric(1))
```

supported_formats *Supported file formats*

Description

Lists the file formats understood by `read_data()`, `write_data()` and `generate_to_file()`, with their file extensions and the package used as backend. Backends other than base R are optional (`Suggests`) and only needed for the corresponding format. Text formats (`csv`, `tsv`, `json`, `ndjson`) can additionally be compressed by appending `.gz`, `.bz2` or `.xz` to the file name.

Usage

```
supported_formats(available_only = FALSE)
```

Arguments

`available_only` If `TRUE`, only formats whose backend is installed are returned.

Value

A data frame with columns `format`, `extensions`, `read`, `write` and `available`.

Examples

```
supported_formats()
```

synthesize *Fit a synthesizer and generate synthetic data in one step*

Description

`synthesize()` is a convenience wrapper around `fit_synthesizer()` and `generate()`. Optionally the result is written to a file in any format supported by `write_data()`.

Usage

```
synthesize(
  data,
  n = NULL,
  seed = NULL,
  dependence = 1,
  closeness = 1,
  perturb = NULL,
  file = NULL,
  format = NULL,
  ...
)
```

Arguments

data	Training data (data frame, matrix, vector or ts).
n	Number of synthetic records. Defaults to the size of data.
seed	Optional random seed (applies to fitting and generation).
dependence	Dependence strength in [0, 1]; see generate().
closeness, perturb	Closeness to the real data and optional perturbation; see generate().
file	Optional output file path. The format is inferred from the extension (see supported_formats()).
format	Optional explicit file format, overriding the extension.
...	Arguments passed to fit_synthesizer().

Value

The synthetic data (invisibly if file is given).

Examples

```
syn <- synthesize(iris, n = 200, seed = 1)
summary(syn)

out <- tempfile(fileext = ".csv")
synthesize(mtcars, file = out, seed = 1)
head(read_data(out))
```

---

ts_windows *Build a lagged-feature table from one or more time series*

Description

Converts time series into a data frame suitable for training forecasting models: each row holds the target y (the value horizon steps ahead) and the lags preceding values lag1 (most recent) ... lagp. With a list of series (e.g. from augment_data()), the windows of all series are stacked and a series column identifies their origin; windows never cross series boundaries.

Usage

```
ts_windows(x, lags, horizon = 1L, target = 1L)
```

Arguments

x	A ts object, numeric vector/matrix, or a list of them.
lags	Number of lags.
horizon	Forecast horizon (1 = one step ahead).
target	For multivariate series: name or index of the target column.

Details

For multivariate series, lags of every component are created (`<name>_lag1`, ...) and the target is the column named by `target`.

Value

A data frame with columns `y`, the lag columns and (for lists) `series`.

Examples

```
w <- ts_windows(ldeaths, lags = 3)
head(w)
fit <- lm(y ~ ., data = w)
```

---

write_data — *Write a data set to a file*

Description

`write_data()` writes a data frame to any format listed by `supported_formats()`. The format is inferred from the file extension unless `format` is given; text formats are compressed when the file name ends in `.gz`, `.bz2` or `.xz`.

HDF5 files are written with one dataset per column inside a group (`dataset`, default `"data"`), with attributes that record column order, factor levels, `Date`/`POSIXct` classes, time zones and missing strings, so that `read_data()` restores the original column types.

Date-times (`POSIXct`) keep their instant in every format. Text formats (CSV, TSV, JSON, NDJSON) store them as ISO 8601 strings in UTC; Excel, SPSS, Stata and SAS files, which store clock time without a time zone, receive the UTC clock time; Parquet, Feather, fst and HDF5 store the instant together with the time zone. Pass the original data as `template` to `read_data()` to restore the original time zone.

Usage

```
write_data(
  x,
  path,
  format = NULL,
  overwrite = TRUE,
  object = "data",
  dataset = "data",
  ...
)
```

Arguments

`x`	A data frame (matrices are converted).
`path`	Output file path.
`format`	Optional format name or extension.
`overwrite`	Overwrite an existing file?
`object`	For `rdata`: name under which the object is saved.
`dataset`	For `hdf5`: name of the group to write.
`...`	Further arguments passed to the backend writer.

Value

The path, invisibly.

See Also

`read_data()`, `generate_to_file()`

Examples

```
f <- tempfile(fileext = ".tsv.gz")
write_data(mtcars, f)
head(read_data(f))
```

Index

∗ **datasets**
census_sim, 8

association_matrix, 3
association_matrix(), *10*, *31*
augment_data, 4
augment_data(), *42*
autoplot.sp_comparison, 6

calibrate_closeness, 6
calibrate_closeness(), *22*, *30*
census_sim, 8
ci_overlap, 9
compare_synthetic, 10
compare_synthetic(), *6*, *19*, *32*
copula_correlation, 11

dcr, 12
dcr(), *7*, *10*, *16*, *32*
disclosure_risk, 13
disclosure_risk(), *7*
discriminator_auc, 15
discriminator_auc(), *7*, *10*

fit_synthesizer, 16
fit_synthesizer(), *5*, *11*, *20–23*, *41*, *42*
fit_synthesizer.data.frame(), *21*
fit_synthesizer.ts, 20
fit_synthesizer.ts(), *4*, *17*, *22*

generate, 22
generate(), *5*, *7*, *8*, *16*, *19*, *21*, *24*, *29*, *30*, *40–42*
generate_to_file, 23
generate_to_file(), *23*, *39*, *41*, *44*

make_corr, 24
margin_categorical, 25
margin_categorical(), *38*
margin_empirical (margin_categorical), 25
margin_empirical(), *38*
marginal_metrics, 26
marginal_metrics(), *7*, *10*
match_types, 27
match_types(), *39*
mvdist, 27, *33*

perturbation, 29
perturbation(), *22*, *23*
plot.sp_comparison (autoplot.sp_comparison), 6
plot_acf (plot_ts), 35
plot_association, 31
plot_categorical (plot_marginals), 32
plot_dcr, 32
plot_marginals, 32
plot_pairs, 33
plot_qq, 34
plot_ts, 35
pmse, 35
pmse(), *7*, *10*

r_copula, 37
r_copula(), *38*
r_dirichlet (mvdist), 27
r_mvdist, 38
r_mvdist(), *8*, *33*
r_mvlnorm (mvdist), 27
r_mvmixture (mvdist), 27
r_mvnorm (mvdist), 27
r_mvskewnorm (mvdist), 27
r_mvt (mvdist), 27
read_data, 39
read_data(), *23*, *41*, *43*, *44*

simulate.sp_synthesizer, 40
simulate.sp_ts_synthesizer (simulate.sp_synthesizer), 40
stats::cov2cor(), *37*
stats::family(), *9*

summary(), *19*
summary.sp_synthesizer
(copula_correlation), 11
supported_formats, 41
supported_formats(), *39*, *42*, *43*
synthesize, 41
synthesize(), *16*, *18*, *19*, *23*

ts_windows, 42
ts_windows(), *4*, *5*

write_data, 43
write_data(), *39*, *41*