

# Package: rsdv (via r-universe)

June 9, 2026

**Type** Package

**Title** Synthetic Tabular Data Generation with Gaussian Copulas

**Version** 0.2.0

**Description** Generates synthetic tabular data from real datasets using Gaussian copula models, with parametric marginal selection for numerical columns and a cumulative-frequency embedding that brings categorical and boolean columns into the same joint copula. Includes a metadata system with column types and primary keys, declarative constraints enforced via rejection sampling, conditional sampling, and quality, validity and privacy reports modeled on those of the 'SDMetrics' library. Inspired by the Python 'SDV' (Synthetic Data Vault) library by 'DataCebo'; see Patki, Wedge and Veeramachaneni (2016) ``The Synthetic Data Vault" <doi:10.1109/DSAA.2016.49>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.3.3

**URL** <https://kvenkita.github.io/rsdv/>, <https://github.com/kvenkita/rsdv>

**BugReports** <https://github.com/kvenkita/rsdv/issues>

**Depends** R (>= 4.3.0)

**Imports** copula (>= 1.1-0), generics (>= 0.1.3), jsonlite (>= 1.8.0), ggplot2 (>= 3.4.0), tibble (>= 3.2.0), FNN (>= 1.1.3), rpart (>= 4.1.0), scales (>= 1.2.0), stats, utils

**Suggests** testthat (>= 3.0.0), withr, knitr (>= 1.40), rmarkdown (>= 2.20)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**LazyData** true

**NeedsCompilation** no

**Author** Kailas Venkitasubramanian [aut, cre]  
**Maintainer** Kailas Venkitasubramanian <kailasv@gmail.com>  
**Repository** <https://cran.r-universe.dev>  
**Date/Publication** 2026-06-09 08:44:33 UTC  
**RemoteUrl** <https://github.com/cran/rsdv>  
**RemoteRef** HEAD  
**RemoteSha** 18dde482a96189c7e7d7f8ab72b2d6dff089bc2

## Contents

add_constraint . . . . .	3
adult_income . . . . .	3
attribute_disclosure_risk . . . . .	4
autoplot.rsdv_diagnostic_report . . . . .	5
autoplot.rsdv_privacy_report . . . . .	6
autoplot.rsdv_quality_report . . . . .	6
check_constraint . . . . .	7
check_constraints . . . . .	8
contingency_similarity . . . . .	8
correlation_similarity . . . . .	9
custom_constraint . . . . .	10
diagnostic_report . . . . .	11
equality_constraint . . . . .	12
fixed_combinations_constraint . . . . .	12
gaussian_copula_synthesizer . . . . .	13
inequality_constraint . . . . .	14
is_fitted . . . . .	14
ks_similarity . . . . .	15
load_metadata . . . . .	16
metadata . . . . .	16
metadata_from_json . . . . .	17
metadata_to_json . . . . .	17
ml_efficacy . . . . .	18
nndr . . . . .	19
print.custom_constraint . . . . .	20
print.equality_constraint . . . . .	20
print.fixed_combinations_constraint . . . . .	21
print.inequality_constraint . . . . .	21
print.rsdv_diagnostic_report . . . . .	22
print.rsdv_metadata . . . . .	22
print.rsdv_privacy_report . . . . .	23
print.rsdv_quality_report . . . . .	23
privacy_report . . . . .	24
quality_report . . . . .	25
sample . . . . .	26
sample_conditions . . . . .	27

<code>add_constraint</code>	3
<code>save_metadata</code> . . . . .	27
<code>set_column_type</code> . . . . .	28
<code>set_primary_key</code> . . . . .	29
<code>tvdsimilarity</code> . . . . .	29
<code>validate_data</code> . . . . .	30
<b>Index</b>	<b>31</b>

---

<code>add_constraint</code>	<i>Add a constraint to metadata</i>
-----------------------------	-------------------------------------

---

### Description

Add a constraint to metadata

### Usage

```
add_constraint(meta, constraint)
```

### Arguments

<code>meta</code>	An <code>rsdv_metadata</code> object.
<code>constraint</code>	A constraint object from <code>equality_constraint()</code> , <code>inequality_constraint()</code> , <code>fixed_combinations_constraint()</code> , or <code>custom_constraint()</code> .

### Value

Updated `rsdv_metadata` (for piping).

### Examples

```
meta <- metadata() |>
  set_column_type("low", "numerical") |>
  set_column_type("high", "numerical") |>
  add_constraint(inequality_constraint("low", "high", type = "lt"))
```

---

<code>adult_income</code>	<i>Adult Income dataset (500-row sample)</i>
---------------------------	--

---

### Description

A 500-row random sample of the UCI Adult Income dataset, used in package examples and vignettes.

### Usage

```
adult_income
```

**Format**

A tibble with 500 rows and 16 variables:

**id** Row identifier (integer)  
**age** Age in years (integer)  
**workclass** Employment type (character)  
**fnlwgt** Final weight, a census sampling weight (integer)  
**education** Highest level of education (character)  
**education\_num** Education encoded as an integer (integer)  
**marital\_status** Marital status (character)  
**occupation** Occupation category (character)  
**relationship** Relationship to householder (character)  
**race** Race (character)  
**sex** Sex (character)  
**capital\_gain** Capital gains (integer)  
**capital\_loss** Capital losses (integer)  
**hours\_per\_week** Hours worked per week (integer)  
**native\_country** Country of origin (character)  
**income** Income bracket: <=50K or >50K (character)

**Source**

<https://archive.ics.uci.edu/dataset/2/adult>

---

attribute\_disclosure\_risk

*Attribute disclosure risk*

---

**Description**

Estimates the fraction of synthetic rows where a sensitive column value can be correctly inferred from known columns via a k-NN lookup in the real training data.

**Usage**

```
attribute_disclosure_risk(real, synthetic, sensitive_col, known_cols, k = 1L)
```

**Arguments**

**real, synthetic** Data frames.  
**sensitive\_col** Name of the column to protect.  
**known\_cols** Character vector of **numeric** columns assumed known to an adversary. Categorical columns are rejected with a clear error.  
**k** Number of nearest neighbors used in inference.

**Details**

known\_cols must be numeric, because nearest-neighbour lookup operates on Euclidean distance over the columns. If you want to use a categorical column as a known attribute, one-hot encode it first (e.g. with `model.matrix(~ col - 1, data)`).

**Value**

A scalar in  $[0, 1]$ ; lower = more private.

**Examples**

```
real <- data.frame(age = sample(20:60, 50, replace = TRUE),
                  income = sample(c("low", "high"), 50, replace = TRUE),
                  stringsAsFactors = FALSE)
syn <- real[sample(50), ]
attribute_disclosure_risk(real, syn, sensitive_col = "income", known_cols = "age")
```

---

```
autoplot.rsdv_diagnostic_report
  Plot a diagnostic report
```

---

**Description**

Bar chart of per-column validity scores.

**Usage**

```
## S3 method for class 'rsdv_diagnostic_report'
autoplot(object, ...)
```

**Arguments**

object	An rsdv_diagnostic_report object.
...	Unused.

**Value**

A ggplot object.

**Examples**

```
meta <- metadata(adult_income)
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth <- sample(syn, n = 500)
ggplot2::autoplot(diagnostic_report(adult_income, synth, meta))
```

---

```
autoplot.rsdv_privacy_report  
  Plot a privacy report
```

---

**Description**

Plots the NNDR score as a gauge-style bar.

**Usage**

```
## S3 method for class 'rsdv_privacy_report'  
autoplot(object, ...)
```

**Arguments**

object	An rsdv_privacy_report object.
...	Unused.

**Value**

A ggplot object.

**Examples**

```
syn <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)  
synth <- sample(syn, n = 500)  
pr <- privacy_report(adult_income, synth)  
ggplot2::autoplot(pr)
```

---

```
autoplot.rsdv_quality_report  
  Plot a quality report
```

---

**Description**

Produces a bar chart of per-column similarity scores, with a horizontal line at the overall score.

**Usage**

```
## S3 method for class 'rsdv_quality_report'  
autoplot(object, ...)
```

**Arguments**

object	An rsdv_quality_report object.
...	Unused.

**Value**

A ggplot object.

**Examples**

```
syn <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)
synth <- sample(syn, n = 500)
qr <- quality_report(adult_income, synth, metadata(adult_income))
ggplot2::autoplot(qr)
```

---

check_constraint	<i>Check a single constraint against each row of a data frame</i>
------------------	---

---

**Description**

Check a single constraint against each row of a data frame

**Usage**

```
check_constraint(data, constraint)
```

**Arguments**

data	A data frame.
constraint	An rsdv_constraint object.

**Value**

Logical vector of length nrow(data).

**Examples**

```
df <- data.frame(a = c(1, 2, 3), b = c(1, 2, 9))
check_constraint(df, equality_constraint("a", "b"))
```

---

check_constraints	<i>Check all constraints in metadata against a data frame</i>
-------------------	---

---

**Description**

Check all constraints in metadata against a data frame

**Usage**

```
check_constraints(data, meta)
```

**Arguments**

data	A data frame.
meta	An rsdv_metadata object.

**Value**

Logical vector of length `nrow(data)`. TRUE = row passes all constraints.

**Examples**

```
meta <- metadata() |>
  set_column_type("x", "numerical") |>
  add_constraint(custom_constraint(function(row) row$x > 0))
check_constraints(data.frame(x = c(1, -1, 2)), meta)
```

---

contingency_similarity	<i>Contingency similarity between real and synthetic categorical column pairs</i>
------------------------	---

---

**Description**

For each pair of categorical columns, compares the joint (normalized contingency) distributions of real and synthetic data via total variation distance, scoring  $1 - \text{TVD}$  (the `SDMetrics ContingencySimilarity` score). This is the categorical analogue of correlation similarity and captures categorical-vs-categorical dependence.

**Usage**

```
contingency_similarity(real, synthetic, meta)
```

**Arguments**

<code>real</code>	A data frame of real data.
<code>synthetic</code>	A data frame of synthetic data.
<code>meta</code>	An <code>rsdv_metadata</code> object.

**Value**

A list with pairs (a tibble of `column_1`, `column_2`, `score`) and `score` (the mean over pairs). `score` is `NA_real_` when there are fewer than two categorical columns — there is no dependence to measure, so propagating NA (rather than 1) avoids overstating fidelity in the aggregated quality report.

**Examples**

```
meta <- metadata(adult_income)
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth <- sample(syn, n = 500)
contingency_similarity(adult_income, synth, meta)
```

---

`correlation_similarity`

*Correlation similarity between real and synthetic numerical column pairs*

---

**Description**

For each pair of numerical columns, computes  $1 - |\text{corr\_real} - \text{corr\_syn}| / 2$  (the SDMetrics CorrelationSimilarity score), where `corr` is the Pearson correlation. Returns one row per pair plus the mean.

**Usage**

```
correlation_similarity(real, synthetic, meta)
```

**Arguments**

<code>real</code>	A data frame of real data.
<code>synthetic</code>	A data frame of synthetic data.
<code>meta</code>	An <code>rsdv_metadata</code> object.

**Value**

A list with pairs (a tibble of `column_1`, `column_2`, `score`) and `score` (the mean over pairs). `score` is `NA_real_` when there are fewer than two numerical columns — there is no dependence to measure, so propagating NA (rather than 1) avoids overstating fidelity in the aggregated quality report.

**Examples**

```
syn      <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)
synth_data <- sample(syn, n = 500)
correlation_similarity(adult_income, synth_data, metadata(adult_income))
```

---

custom\_constraint      *Constraint: arbitrary row-wise predicate*

---

**Description**

With `vectorized = FALSE` (default) `fn` is invoked once per row with a one-row data frame and must return a single logical — easy to write but slow on large frames. With `vectorized = TRUE` `fn` is invoked **once** with the full data frame and must return a logical vector of length `nrow(data)`; use this when your predicate is vectorisable for substantial speedups on large synthetic samples.

**Usage**

```
custom_constraint(fn, vectorized = FALSE)
```

**Arguments**

<code>fn</code>	A predicate function. If <code>vectorized = FALSE</code> , signature is <code>f(row)</code> returning a single logical. If <code>vectorized = TRUE</code> , signature is <code>f(data)</code> returning a logical vector of length <code>nrow(data)</code> .
<code>vectorized</code>	Logical. See above. Default <code>FALSE</code> .

**Value**

An `rsdv_constraint` object.

**Examples**

```
custom_constraint(function(row) row$x > 0)
# Vectorised – usually much faster:
custom_constraint(function(data) data$x > 0, vectorized = TRUE)
```

---

diagnostic_report	<i>Generate a diagnostic (validity) report for synthetic data</i>
-------------------	---

---

### Description

Checks whether synthetic data is *structurally valid* against the real data and metadata — independent of how closely it matches the real distributions (that is the job of `quality_report()`). Mirrors the SDMetrics `DiagnosticReport` two-property hierarchy:

### Usage

```
diagnostic_report(real, synthetic, metadata)
```

### Arguments

<code>real</code>	A data frame of real data.
<code>synthetic</code>	A data frame of synthetic data.
<code>metadata</code>	An <code>rsdv_metadata</code> object.

### Details

- **Data Validity** — per-column checks:
  - numerical: boundary adherence (fraction of values within the real min/max range),
  - categorical: category adherence (fraction of values whose category was seen in the real data),
  - boolean: always valid,
  - primary key: key uniqueness (all values unique and non-missing).
- **Data Structure** — fraction of expected columns present in the synthetic data.

Missing (NA) values are excluded from adherence denominators, since missingness is modeled separately.

### Value

An `rsdv_diagnostic_report` object.

### Examples

```
meta <- metadata(adult_income)
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth <- sample(syn, n = 500)
diagnostic_report(adult_income, synth, meta)
```

---

equality\_constraint     *Constraint: two columns must be equal row-wise*

---

### Description

For continuous numerical columns, `exact ==` is almost never satisfied by the copula sampler; use the tolerance argument or `inequality_constraint()` with a narrow band. With `tolerance > 0`, equality is `abs(a - b) <= tolerance` for numeric columns and `exact ==` otherwise.

### Usage

```
equality_constraint(col_a, col_b, tolerance = 0)
```

### Arguments

`col_a, col_b`     Column names (character).  
`tolerance`        Numeric. When non-zero, numeric columns compare with `abs(a - b) <= tolerance` instead of `exact ==`. Ignored for non-numeric columns. Default `0` (exact equality).

### Value

An `rsdv_constraint` object.

### Examples

```
equality_constraint("city", "city_copy")
equality_constraint("price_left", "price_right", tolerance = 1e-6)
```

---

fixed\_combinations\_constraint

*Constraint: only observed column combinations are valid*

---

### Description

Constraint: only observed column combinations are valid

### Usage

```
fixed_combinations_constraint(columns, reference_data)
```

### Arguments

`columns`            Character vector of column names.  
`reference_data`     Data frame containing the allowed combinations.

**Value**

An `rsdv_constraint` object.

**Examples**

```
ref <- data.frame(city = c("NY", "LA"), state = c("NY", "CA"),
                 stringsAsFactors = FALSE)
fixed_combinations_constraint(c("city", "state"), ref)
```

---

`gaussian_copula_synthesizer`

*Create a Gaussian Copula synthesizer*

---

**Description**

Fits a single Gaussian copula over **all** modeled columns. Numerical columns use a fitted parametric marginal (see `default_distribution`); categorical and boolean columns are embedded into the copula via their cumulative-frequency intervals, so cross-column dependence (numeric vs. categorical, categorical vs. categorical) is preserved.

**Usage**

```
gaussian_copula_synthesizer(
  metadata,
  enforce_min_max = TRUE,
  numerical_distributions = list(),
  default_distribution = "auto"
)
```

**Arguments**

`metadata` An `rsdv_metadata` object.

`enforce_min_max` Logical. Clamp sampled numerical values to the observed range. Default `TRUE`.

`numerical_distributions` Optional named character vector/list mapping numerical column names to a distribution in `"norm"`, `"beta"`, `"gamma"`, `"truncnorm"`, `"uniform"`, or `"auto"`.

`default_distribution` Distribution used for numerical columns not named in `numerical_distributions`. `"auto"` (default) selects the best-fitting family per column by Kolmogorov-Smirnov distance.

**Value**

An unfitted `gaussian_copula_synthesizer` object.

**Examples**

```
meta <- metadata(adult_income) |>
  set_column_type("age", "numerical") |>
  set_column_type("occupation", "categorical")
syn <- gaussian_copula_synthesizer(meta, default_distribution = "auto")
syn <- fit(syn, adult_income)
```

---

`inequality_constraint` *Constraint: col\_a must be less than / greater than col\_b*

---

**Description**

Constraint: col\_a must be less than / greater than col\_b

**Usage**

```
inequality_constraint(col_a, col_b, type = c("lt", "lte", "gt", "gte"))
```

**Arguments**

`col_a, col_b` Column names (character).  
`type` One of "lt", "lte", "gt", "gte".

**Value**

An `rsdv_constraint` object.

**Examples**

```
inequality_constraint("low", "high", type = "lt")
```

---

`is_fitted` *Check whether a synthesizer has been fitted*

---

**Description**

Check whether a synthesizer has been fitted

**Usage**

```
is_fitted(x)
```

**Arguments**

`x` A synthesizer object.

**Value**

TRUE if `fit()` has been called; FALSE otherwise.

**Examples**

```
syn <- gaussian_copula_synthesizer(metadata())  
is_fitted(syn) # FALSE before fitting
```

---

ks_similarity	<i>Kolmogorov-Smirnov similarity score per numerical column</i>
---------------	---

---

**Description**

Kolmogorov-Smirnov similarity score per numerical column

**Usage**

```
ks_similarity(real, synthetic, meta)
```

**Arguments**

real	A data frame of real data.
synthetic	A data frame of synthetic data.
meta	An <code>rsdv_metadata</code> object.

**Value**

A tibble with columns `column` (chr) and `score` (dbl, 0–1, higher = better).

**Examples**

```
syn <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)  
synth <- sample(syn, n = 500)  
ks_similarity(adult_income, synth, metadata(adult_income))
```

load\_metadata                      *Load metadata from a JSON file*

---

**Description**

Load metadata from a JSON file

**Usage**

```
load_metadata(path)
```

**Arguments**

path                      Path to a JSON file produced by [save\\_metadata\(\)](#).

**Value**

An `rsdv_metadata` object.

**Examples**

```
meta <- metadata() |> set_column_type("age", "numerical")
tmp <- tempfile(fileext = ".json")
save_metadata(meta, tmp)
load_metadata(tmp)
```

---

metadata                      *Create a metadata object describing a dataset's column types*

---

**Description**

Create a metadata object describing a dataset's column types

**Usage**

```
metadata(data = NULL)
```

**Arguments**

data                      Optional data frame. If supplied, column types are auto-detected. You can override them with [set\\_column\\_type\(\)](#).

**Value**

An `rsdv_metadata` object.

### Examples

```
meta <- metadata(adult_income) |>
  set_column_type("age", "numerical") |>
  set_column_type("occupation", "categorical")
```

---

metadata_from_json	<i>Deserialize metadata from a JSON string</i>
--------------------	--

---

### Description

Deserialize metadata from a JSON string

### Usage

```
metadata_from_json(json)
```

### Arguments

json                    A JSON character string produced by [metadata\\_to\\_json\(\)](#).

### Value

An `rsdv_metadata` object. Constraints are reconstructed with their original S3 classes so [check\\_constraint\(\)](#) dispatches correctly.

### Examples

```
meta <- metadata() |>
  set_column_type("a", "numerical") |>
  set_column_type("b", "numerical") |>
  add_constraint(inequality_constraint("a", "b", type = "lt"))
metadata_from_json(metadata_to_json(meta))
```

---

metadata_to_json	<i>Serialize metadata to a JSON string</i>
------------------	--

---

### Description

Round-trips column types, primary key, and the structural constraints (equality, inequality, fixed\_combinations). `custom_constraint` cannot be serialized — it holds an R closure — and is dropped with a warning.

### Usage

```
metadata_to_json(meta)
```

**Arguments**

meta                    An rsdv\_metadata object.

**Value**

A JSON character string. Inverse of `metadata_from_json()`.

**Examples**

```
meta <- metadata() |>
  set_column_type("a", "numerical") |>
  set_column_type("b", "numerical") |>
  add_constraint(inequality_constraint("a", "b", type = "lt"))
json <- metadata_to_json(meta)
meta2 <- metadata_from_json(json)
```

---

ml\_efficacy

*ML efficacy: train-on-synthetic / test-on-real accuracy ratio (TSTR)*

---

**Description**

Trains an rpart decision tree on synthetic data and on a real training split, evaluates both on a real held-out test set, and returns the ratio TSTR / TRTR. A score near 1 means synthetic data is as informative as real data for this prediction task.

**Usage**

```
ml_efficacy(
  real,
  synthetic,
  meta,
  target_col,
  test_fraction = 0.2,
  seed = NULL
)
```

**Arguments**

real                    A data frame of real data.

synthetic              A data frame of synthetic data.

meta                    An rsdv\_metadata object.

target\_col             Name of a categorical column to use as the outcome.

test\_fraction         Fraction of real to hold out as the test set. Must be strictly between 0 and 1.

seed                    Optional integer seed. When supplied, the train/test split is reproducible across calls without affecting the caller's RNG stream.

**Value**

A list with elements `tstr` (accuracy), `trtr` (accuracy), and `score` (ratio, capped at 1).

**Examples**

```
meta      <- metadata(adult_income)
syn       <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth_data <- sample(syn, n = 500)
ml_efficacy(adult_income, synth_data, meta, target_col = "income", seed = 1)
```

---

nndr

*Nearest-Neighbor Distance Ratio privacy score*


---

**Description**

For each synthetic row, computes the ratio of its distance to the nearest real row vs. its distance to the second-nearest real row. A high ratio (close to 1) means the synthetic row is not unusually close to any specific real row — low disclosure risk. Score =  $\text{mean}(\text{ratio} > 0.5)$ .

**Usage**

```
nndr(real, synthetic, normalize = TRUE)
```

**Arguments**

`real`, `synthetic` Data frames; only numerical columns are used.

`normalize` Logical. When TRUE (default), columns are z-scored using the real-data mean and standard deviation before distance computation. Constant columns in `real` are dropped to avoid division by zero.

**Details**

By default columns are z-scored using the real-data mean and standard deviation before the Euclidean distance is computed; without this, a single large-scale column (e.g. income in dollars) dominates the distance and the score becomes a function of measurement units rather than of similarity.

**Value**

A scalar score in  $[0, 1]$ ; higher = more private.

**Examples**

```
real <- data.frame(x = rnorm(50), y = rnorm(50))
syn  <- data.frame(x = rnorm(50), y = rnorm(50))
nndr(real, syn)
```

```
print.custom_constraint
```

*Print method for a custom\_constraint*

---

**Description**

Print method for a custom\_constraint

**Usage**

```
## S3 method for class 'custom_constraint'  
print(x, ...)
```

**Arguments**

x	A custom_constraint object.
...	Unused.

**Value**

x, invisibly.

---

```
print.equality_constraint
```

*Print method for an equality\_constraint*

---

**Description**

Print method for an equality\_constraint

**Usage**

```
## S3 method for class 'equality_constraint'  
print(x, ...)
```

**Arguments**

x	An equality_constraint object.
...	Unused.

**Value**

x, invisibly.

---

```
print.fixed_combinations_constraint
    Print method for a fixed_combinations_constraint
```

---

### **Description**

Print method for a `fixed_combinations_constraint`

### **Usage**

```
## S3 method for class 'fixed_combinations_constraint'
print(x, ...)
```

### **Arguments**

<code>x</code>	A <code>fixed_combinations_constraint</code> object.
<code>...</code>	Unused.

### **Value**

`x`, invisibly.

---

```
print.inequality_constraint
    Print method for an inequality_constraint
```

---

### **Description**

Print method for an `inequality_constraint`

### **Usage**

```
## S3 method for class 'inequality_constraint'
print(x, ...)
```

### **Arguments**

<code>x</code>	An <code>inequality_constraint</code> object.
<code>...</code>	Unused.

### **Value**

`x`, invisibly.

```
print.rsdv_diagnostic_report  
    Print method for rsdv_diagnostic_report
```

---

**Description**

Print method for rsdv\_diagnostic\_report

**Usage**

```
## S3 method for class 'rsdv_diagnostic_report'  
print(x, ...)
```

**Arguments**

x	An rsdv_diagnostic_report object.
...	Unused.

**Value**

x, invisibly.

---

```
print.rsdv_metadata    Print method for rsdv_metadata
```

---

**Description**

Print method for rsdv\_metadata

**Usage**

```
## S3 method for class 'rsdv_metadata'  
print(x, ...)
```

**Arguments**

x	An rsdv_metadata object.
...	Unused.

**Value**

x, invisibly.

**Examples**

```
print(metadata())
```

---

```
print.rsdv_privacy_report
      Print method for rsdv_privacy_report
```

---

**Description**

Print method for rsdv\_privacy\_report

**Usage**

```
## S3 method for class 'rsdv_privacy_report'
print(x, ...)
```

**Arguments**

x	An rsdv_privacy_report object.
...	Unused.

**Value**

x, invisibly.

**Examples**

```
syn <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)
synth <- sample(syn, n = 500)
pr <- privacy_report(adult_income, synth)
print(pr)
```

---

```
print.rsdv_quality_report
      Print method for rsdv_quality_report
```

---

**Description**

Print method for rsdv\_quality\_report

**Usage**

```
## S3 method for class 'rsdv_quality_report'
print(x, ...)
```

**Arguments**

x	An rsdv_quality_report object.
...	Unused.

**Value**

x, invisibly.

---

privacy_report	<i>Generate a privacy report comparing real and synthetic data</i>
----------------	--

---

**Description**

Generate a privacy report comparing real and synthetic data

**Usage**

```
privacy_report(real, synthetic, sensitive_col = NULL, known_cols = NULL)
```

**Arguments**

real	A data frame of real data.
synthetic	A data frame of synthetic data.
sensitive_col	Optional. Column name for attribute disclosure risk.
known_cols	Optional. Column names known to an adversary (required if sensitive_col is supplied).

**Value**

An rsdv\_privacy\_report object.

**Examples**

```
syn  <- gaussian_copula_synthesizer(metadata(adult_income)) |>
      fit(adult_income)
synth <- sample(syn, n = 500)
pr    <- privacy_report(adult_income, synth)
print(pr)
```

---

quality_report	<i>Generate a quality report comparing real and synthetic data</i>
----------------	--

---

### Description

Aggregates metrics into the two-property hierarchy used by SDMetrics:

### Usage

```
quality_report(real, synthetic, metadata, target_col = NULL)
```

### Arguments

real	A data frame of real data.
synthetic	A data frame of synthetic data.
metadata	An <code>rsdv_metadata</code> object.
target_col	Optional. Name of a categorical column for ML efficacy. Reported alongside the score but excluded from the overall.

### Details

- **Column Shapes** — per-column marginal fidelity: KS similarity for numerical columns and TVD similarity for categorical columns.
- **Column Pair Trends** — pairwise dependence: correlation similarity for numerical pairs and contingency similarity for categorical pairs.

The overall score is the mean of the two property scores, so a table with many categorical columns and few numerical ones is not weighted by raw column counts. ML efficacy, when requested, is reported separately and does **not** enter the overall score (matching SDMetrics).

### Value

An `rsdv_quality_report` object.

### Examples

```
meta <- metadata(adult_income) |>
  set_column_type("age", "numerical") |>
  set_column_type("occupation", "categorical")
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth <- sample(syn, n = 500)
qr <- quality_report(adult_income, synth, meta)
print(qr)
```

---

sample	<i>Sample synthetic rows from a fitted synthesizer</i>
--------	--

---

## Description

Dispatches to the synthesizer-specific method when `x` is an `rsdv_synthesizer`. For plain R vectors, integers, or characters it falls back to `base::sample()`, preserving backward compatibility.

## Usage

```
sample(x, n = NULL, ...)
```

## Arguments

<code>x</code>	A fitted synthesizer object, or a vector for <code>base::sample()</code> compat.
<code>n</code>	Number of synthetic rows to generate (synthesizer path), or sample size ( <code>base::sample</code> path).
<code>...</code>	Additional arguments passed to the method or to <code>base::sample()</code> .

## Value

When `x` inherits from `rsdv_synthesizer`, a data frame of `n` synthetic rows whose columns match the metadata. When `x` is any other object, the value returned by `base::sample()` — typically a vector of the same type as `x` and length `n`.

## Examples

```
# Falls back to base::sample for non-synthesizer objects:
sample(1:10, 3)

meta <- metadata(adult_income) |>
  set_column_type("age", "numerical") |>
  set_column_type("income", "categorical")
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
synth <- sample(syn, n = 100)
head(synth)
```

---

sample_conditions	<i>Sample synthetic rows that match fixed column values (conditional sampling)</i>
-------------------	--

---

### Description

Generates rows in which one or more **categorical or boolean** columns are held to specified values, via rejection sampling against the fitted copula. This preserves the modeled dependence between the conditioned columns and the rest of the table (unlike overwriting values after the fact).

### Usage

```
sample_conditions(x, conditions, max_tries = 100L)
```

### Arguments

x	A fitted gaussian_copula_synthesizer.
conditions	A data frame whose columns are the variables to fix. Each row is one condition; an optional integer column .n gives how many rows to generate for that condition (default 1 per row).
max_tries	Maximum rejection-sampling rounds per condition.

### Value

A data frame of synthetic rows satisfying the conditions.

### Examples

```
meta <- metadata(adult_income)
syn <- gaussian_copula_synthesizer(meta) |> fit(adult_income)
sample_conditions(syn, data.frame(income = ">50K", .n = 20))
```

---

save_metadata	<i>Save metadata to a JSON file</i>
---------------	-------------------------------------

---

### Description

Save metadata to a JSON file

### Usage

```
save_metadata(meta, path)
```

**Arguments**

meta            An rsdv\_metadata object.  
 path            File path to write to.

**Value**

Invisibly returns meta.

**Examples**

```
meta <- metadata() |> set_column_type("age", "numerical")
tmp <- tempfile(fileext = ".json")
save_metadata(meta, tmp)
meta2 <- load_metadata(tmp)
```

---

set\_column\_type            *Set the type of a column in metadata*

---

**Description**

Set the type of a column in metadata

**Usage**

```
set_column_type(meta, column, type)
```

**Arguments**

meta            An rsdv\_metadata object.  
 column         Column name (character).  
 type            One of "numerical", "categorical", "boolean", "datetime", "id".  
 For categorical columns the level *order* used by the synthesizer follows the input: a factor keeps its levels() order (including ordered factors), while a plain character column gets a sorted unique-value order for determinism. The sort is **lexicographic**, so numeric-like character columns (c("2", "10")) come back ordered "10", "2". Coerce these to factor with the desired level order before fitting if order matters.

**Value**

The updated rsdv\_metadata object (for piping).

**Examples**

```
metadata() |> set_column_type("age", "numerical")
```

---

set_primary_key	<i>Set the primary key column of the metadata</i>
-----------------	---

---

**Description**

Set the primary key column of the metadata

**Usage**

```
set_primary_key(meta, column)
```

**Arguments**

meta	An <code>rsdv_metadata</code> object.
column	Name of the primary key column. Must already be registered via <code>set_column_type()</code> .

**Value**

The updated `rsdv_metadata` object (for piping).

**Examples**

```
meta <- metadata() |>
  set_column_type("id", "id") |>
  set_primary_key("id")
```

---

tv_dsimilarity	<i>Total variation distance similarity score per categorical column</i>
----------------	---

---

**Description**

Total variation distance similarity score per categorical column

**Usage**

```
tv_dsimilarity(real, synthetic, meta)
```

**Arguments**

real	A data frame of real data.
synthetic	A data frame of synthetic data.
meta	An <code>rsdv_metadata</code> object.

**Value**

A tibble with columns `column` (chr) and `score` (dbl, 0–1, higher = better).

**Examples**

```
syn  <- gaussian_copula_synthesizer(metadata(adult_income)) |> fit(adult_income)
synth <- sample(syn, n = 500)
tvd_similarity(adult_income, synth, metadata(adult_income))
```

---

`validate_data`*Validate that a data frame is compatible with metadata*

---

**Description**

Checks that all columns registered in meta are present in data.

**Usage**

```
validate_data(data, meta)
```

**Arguments**

<code>data</code>	A data frame.
<code>meta</code>	An <code>rsdv_metadata</code> object.

**Value**

Invisibly TRUE; throws an error if validation fails.

**Examples**

```
meta <- metadata() |> set_column_type("age", "numerical")
validate_data(data.frame(age = 1:5), meta)
```

# Index

- \* **datasets**
  - adult\_income, 3
- add\_constraint, 3
- adult\_income, 3
- attribute\_disclosure\_risk, 4
- autoplot.rsdrv\_diagnostic\_report, 5
- autoplot.rsdrv\_privacy\_report, 6
- autoplot.rsdrv\_quality\_report, 6
  
- base::sample(), 26
  
- check\_constraint, 7
- check\_constraint(), 17
- check\_constraints, 8
- contingency\_similarity, 8
- correlation\_similarity, 9
- custom\_constraint, 10
- custom\_constraint(), 3
  
- diagnostic\_report, 11
  
- equality\_constraint, 12
- equality\_constraint(), 3
  
- fit(), 15
- fixed\_combinations\_constraint, 12
- fixed\_combinations\_constraint(), 3
  
- gaussian\_copula\_synthesizer, 13
  
- inequality\_constraint, 14
- inequality\_constraint(), 3, 12
- is\_fitted, 14
  
- ks\_similarity, 15
  
- load\_metadata, 16
  
- metadata, 16
- metadata\_from\_json, 17
- metadata\_from\_json(), 18
  
- metadata\_to\_json, 17
- metadata\_to\_json(), 17
- ml\_efficacy, 18
  
- nndr, 19
  
- print.custom\_constraint, 20
- print.equality\_constraint, 20
- print.fixed\_combinations\_constraint, 21
- print.inequality\_constraint, 21
- print.rsdrv\_diagnostic\_report, 22
- print.rsdrv\_metadata, 22
- print.rsdrv\_privacy\_report, 23
- print.rsdrv\_quality\_report, 23
- privacy\_report, 24
  
- quality\_report, 25
- quality\_report(), 11
  
- sample, 26
- sample\_conditions, 27
- save\_metadata, 27
- save\_metadata(), 16
- set\_column\_type, 28
- set\_column\_type(), 16, 29
- set\_primary\_key, 29
  
- tvd\_similarity, 29
  
- validate\_data, 30