

# Package ‘WoodSimulatR’

January 20, 2025

**Title** Generate Simulated Sawn Timber Strength Grading Data

**Version** 0.6.1

**Date** 2023-12-04

**Description** Tools for generating simulated sawn timber strength grading data with a main focus on statistical simulation based on covariance matrices. Simulation data for Norway spruce sawn timber from Austria and reference values of means and standard deviations of grade determining properties from literature for a number of European countries are provided, as well.

**License** MIT + file LICENSE

**Imports** stats, dplyr, rlang (>= 0.4.6), purrr, tibble, tidyselect, tidyr

**Encoding** UTF-8

**LazyData** true

**Language** en-GB

**RoxygenNote** 7.2.3

**Config/testthat/edition** 3

**Suggests** ggplot2, knitr, magrittr, pander, rmarkdown, scales, testthat, utf8

**VignetteBuilder** knitr

**Depends** R (>= 2.10)

**NeedsCompilation** no

**Author** Andreas Weidenhiller [cre, aut]  
(<<https://orcid.org/0000-0003-0938-2159>>),  
Anton Wegscheider [aut]

**Maintainer** Andreas Weidenhiller <a.weidenhiller@holzforschung.at>

**Repository** CRAN

**Date/Publication** 2023-12-04 12:20:02 UTC

## Contents

gdp_data . . . . .	2
get_subsample_definitions . . . . .	3
get_transform_names . . . . .	5
simbase . . . . .	6
simbase_covar . . . . .	7
simbase_labeler . . . . .	9
simbase_list . . . . .	9
simulate_conditionally . . . . .	10
simulate_conditionally.simbase_list . . . . .	11
simulate_dataset . . . . .	12

<b>Index</b>	<b>15</b>
--------------	-----------

---

gdp_data	<i>Means and standard deviations of grade determining properties (GDPs) from literature</i>
----------	---

---

### Description

Means and standard deviations of grade determining properties (GDPs) from literature

### Usage

gdp\_data

### Format

**species** Wood species as a four letter code according to EN 13556. Currently, this is always "PCAB" for Norway spruce (*Picea abies*).

**loadtype** Kind of destructive testing applied to the material – "t" for material tested in tension, "be" for material tested in bending.

**project** Research project from which the data originated; "null" if unknown or not applicable.

**country** Country from which the material originated, as a two letter country code.

**share** Number of pieces on which the values are based.

**f\_mean, f\_sd** Mean and standard deviation of strength, in N/mm<sup>2</sup>.

**E\_mean, E\_sd** Mean and standard deviation of the static modulus of elasticity, in N/mm<sup>2</sup>.

**rho\_mean, rho\_sd** Mean and standard deviation of density, in kg/m<sup>3</sup>.

**literature** Reference to the literature source; "null" if not published yet.

**subsample** For distinguishing multiple rows with the same species, loadtype and country – if there are no duplicates, it is the same as country; if there are duplicates, it is country plus a suffixed number separated by "\_".

## Details

For simulation of an entire dataset with different subsamples with different characteristics (see [simulate\\_dataset](#)), it may be useful to be able to refer to existing results from literature as a basis.

In the dataset `gdp_data`, means and standard deviations for a number of such subsamples have been collected.

The GDP values collected in `gdp_data` were selected from publications which aimed at representative sampling within the respective countries. All the same, care must be taken when using these values, due to the natural high variability of timber properties.

## Source

The values have been extracted from the following publications:

Ranta-Maunus, Alpo, Julia K. Denzler, and Peter Stapel. 2011. *Strength of European Timber. Part 2. Properties of Spruce and Pine Tested in Gradewood Project*. VTT.

Rohanová, Alena, and Erika Nunez. 2014. "Prediction Models of Slovakian Structural Timber." *Wood Research* 59 (5): 757–69.

Stapel, Peter, and Jan-Willem G. van de Kuilen. 2014. "Efficiency of Visual Strength Grading of Timber with Respect to Origin, Species, Cross Section, and Grading Rules: A Critical Evaluation of the Common Standards." *Holzforschung* 68 (2): 203–16.

---

get\_subsample\_definitions

*Retrieve descriptive data for samples from literature*

---

## Description

In the `WoodSimulatR` package, means and standard deviations of grade determining properties (GDPs) for a number of Norway spruce (*Picea abies*) samples from literature are stored for use in [simulate\\_dataset](#). They are indexed by a two-letter country code (and a suffixed number if disambiguation is required).

## Usage

```
get_subsample_definitions(country = NULL, loadtype = "t", species = "PCAB")
```

## Arguments

country	Can be either the number of desired samples, or a named vector of relative subsample sizes where the names can be abbreviations of country names. Alternatively, country can also be a character vector of country abbreviations.
loadtype	Can be either "be" for "bending edgewise" or "t" for "tension".
species	A species code according to EN 13556:2003. Currently, only 'PCAB' ( <i>Picea abies</i> = Norway spruce) is supported.

## Details

The direct descriptive data can also be directly accessed (`gdp_data`). The present function is meant to prepare the data as input to the `subsets` argument of `simulate_dataset`. It allows picking multiple samples from the same country (disambiguating by creating appropriately named entries in the column `subsample`) and creating random sample data (uniformly distributed within the range of values given in the full dataset `gdp_data` for the respective loadtype and species) for sample names not found in this dataset.

The dataset `gdp_data` contains a column `share` which gives the number of pieces in the original sample. Unless relative subsample sizes are explicitly asked for by providing a named numeric vector for the argument `country`, the present function always resets `share` to 1, prompting `simulate_dataset` to create (approximately) equal-sized subsamples.

The GDPs depend on the type of destructive testing done (`loadtype`) – therefore, giving the proper `loadtype` is required for realistic values.

If `country` is NULL (or omitted), the full dataset `gdp_data` for the respective `loadtype` (and species) is returned.

For sample names not contained in the internal list, a warning is issued and random sample data is returned (uniformly distributed within the range of values given in the full table for the respective `loadtype` and species).

If `country` is just a number (and *not* a named vector), also random sample data is returned; the different "countries" are then named "C1", "C2" and so on.

## Value

A data frame with `country` and `subsample` names, relative subsample sizes and some meta-information like project and literature references, as well as mean strength and standard deviation of strength, static modulus of elasticity and density.

## Notes

The GDP values collected in `gdp_data` were selected from publications which aimed at representative sampling within the respective countries. All the same, care must be taken when using these values, due to the natural high variability of timber properties.

## Examples

```
# get all subsample data for loadtype bending, or tension
get_subsample_definitions()
get_subsample_definitions(loadtype='be')

# get six random samples, explicitly state loadtype tension
get_subsample_definitions(country=6, loadtype='t')

# get subsample data for the German tension sample in different ways
get_subsample_definitions(country='de', loadtype='t')
get_subsample_definitions(country=c(de=1), loadtype='t')
get_subsample_definitions(country=c(de=6), loadtype='t')

# bending samples from Sweden (both samples), Poland, and France, equally
```

```

# weighted
get_subsample_definitions(c('se', 'se_1', 'pl', 'fr'))
get_subsample_definitions(c(se=1, se_1=1, pl=1, fr=1))
get_subsample_definitions(c(se=5, se_1=5, pl=5, fr=5))

# four tension samples from Romania, two from Ukraine and one from Slovakia,
# weighted so that each country contributes equally
get_subsample_definitions(c(ro=1, ro=1, ro=1, ro=1, ua=2, ua=2, sk=4), loadtype='t')

# non-existent subsample names get replaced by random values (which are based
# on the range of stored values for the respective loadtype)
get_subsample_definitions(c('xx', 'yy', 'zz'))
get_subsample_definitions(c('xx', 'yy', 'zz'), loadtype='t')

# subsample names are case-sensitive!
get_subsample_definitions(c('at', 'aT', 'At', 'AT'), loadtype='t')

```

---

get\_transform\_names     *Return labels for given transforms*

---

## Description

The function [simbase\\_covar](#) allows the specification of a transform for one or more variables. The present function creates short names for such transforms for use in labelling (by default, the labelling is done by [simbase\\_labeler](#)).

## Usage

```

get_transform_names(
  transforms,
  prefer_primitive = c("if_shorter", "never", "always")
)

```

## Arguments

transforms	A named list of objects of class <code>trans</code> or class <code>transform</code> (see function <code>trans_new</code> in package <code>scales</code> )
prefer_primitive	If "never", the function always returns the value of the field name (except if this is missing). If "always", the name of the called function is returned unless it cannot be identified (in many cases, the transform will not be primitive). If "if_shorter", the shorter option of the two above is returned if both can be retrieved.

## Details

The label of a transform could be the value of the field name from each object of class `trans` (or `transform`), but also the name of the transform function itself, if it is a primitive function or just calls one function.

Each object of class `trans` (or `transform`) should have a field name which can be returned by the present function.

The function examines the field `transform`. If this field contains a primitive function (see [is\\_primitive](#)), or if there is just one function call in the body of this `transform` function, we can also return the name of this called function.

If there is no field name and no single function is called from the function defined in the field `transform`, a generic function name "f." is returned.

### Value

A named vector of transforms names.

### Examples

```
get_transform_names(list(a = scales::log_trans(), b = scales::boxcox_trans(0)));
get_transform_names(list(x = list(name = 'a very long name', transform = log, inverse = exp)))
```

---

simbase

*Predefined simbases in WoodSimulatR*

---

### Description

Predefined simbases in WoodSimulatR

### Usage

```
ws_t
ws_t_tr
ws_t_te
ws_t_logf
ws_t_tr_logf
ws_t_te_logf
ws_be
ws_be_tr
ws_be_te
ws_be_logf
ws_be_tr_logf
ws_be_te_logf
```

### Format

For statistical simulation of datasets in WoodSimulatR, one needs a [simbase\\_covar](#) object. WoodSimulatR contains a set of such predefined simbases for Norway spruce (*Picea abies*) grown in Austria.

The names of the simbases follow the following schema – the different parts are separated by "\_":

- "ws" – abbreviation of "WoodSimulatR simbase"
- loadtype – can either be "t" for material tested in tension, or "be" for material tested in bending

- subsample – empty for the full dataset, "tr" for the part of the dataset that was used for training, "te" for the part that was used for testing. The latter two can be used to more closely simulate independent training and test samples
- transformation – empty for no transformation, "logf" if the strength has been log-transformed prior to calculation of the simbase – see also the argument transforms in [simbase\\_covar](#).

The simbases contain the basis for simulating the following variables:

**f** Bending or tension strength, in N/mm<sup>2</sup>.

**E** Static modulus of elasticity in bending or tension, in N/mm<sup>2</sup>.

**rho** Density of a small clear sample, in kg/m<sup>3</sup>.

**E\_dyn** Dynamic modulus of elasticity of the timber after drying to a moisture content of about 12%, in N/mm<sup>2</sup>.

**E\_dyn\_u** Dynamic modulus of elasticity of the timber in the green state, with moisture contents mostly above fibre saturation point, in N/mm<sup>2</sup>.

**ip\_rho** An "indicating property" (IP) for density, established by measuring the weight of each board and dividing by its volume, in kg/m<sup>3</sup>.

**ip\_f** An "indicating property" (IP) for strength, established by linear regression on E\_dyn, ip\_rho and a knot parameter called "total knot area ratio" (tKAR), in N/mm<sup>2</sup>.

### Source

The simbases were created based on data from the research project SiOSiP of Holzforschung Austria. "SiOSiP" is short for "simulation-based optimization of sawn timber production" and ran from 2014 to 2017.

---

simbase_covar	<i>Calculate reference data for simulating values based on a covariance matrix approach</i>
---------------	---

---

### Description

Given the covariance matrix and the means of a set of variables, we can simulate not only the distribution of the variables, but also their correlations. The present function calculates the basic values required for the simulation and returns them packed into an object of class `simbase_covar`.

### Usage

```
simbase_covar(
  data,
  variables = NULL,
  transforms = list(),
  label = simbase_labeler,
  ...
)
```

## Arguments

data	The dataset for the calculation of the reference data for simulation; for grouped datasets (see <a href="#">group_by</a> ), the reference data is calculated for each group separately (see also <a href="#">simbase_list</a> ).
variables	Character vector containing the names in data which should be included in the simulation. If missing, all numeric variables in data are used.
transforms	A named list of objects of class <code>trans</code> or class <code>transform</code> (see function <code>trans_new</code> in package <code>scales</code> ); the name of each list entry <b>must</b> correspond to a variable name in <code>variables</code> .
label	Either a string describing the data and the simulation approach, or a labelling function which returns a label string and takes as input the data, a string giving the class of the <code>simbase</code> object (here <code>"simbase_covar"</code> ) and the <code>transforms</code> list.
...	Arguments to be passed on to <a href="#">simbase_list</a> (if it is called).

## Details

If some of the variables are non-normally distributed, a transform may improve the prediction. The transforms are passed to the function as a named list, where the name of a list entry must correspond to the name of the variable in the data which is to be transformed.

Predefined transforms can be found in the package `scales`, where they are used for axis transformations as a preparation for plotting. The package `scales` also contains a function `trans_new` which can be used to define new transforms.

In the context of destructively measured sawn timber properties, the type of destructive test applied is of interest. If the dataset `data` contains a variable `loadtype` which consistently throughout the dataset has either the value `"t"` (i.e. all sawn timber has been tested in tension) or the value `"be"` (i.e. all sawn timber has been tested in bending, edgewise), then the returned object also has a field `loadtype` with that value.

One can also calculate a `simbase` under the assumption that the correlations are different for different subgroups of the data. This is done by grouping the dataset `data` prior to passing it to the function, using [group\\_by](#). In this case, several objects of class `simbase_covar` are created and joined together in a `tibble` – see also [simbase\\_list](#).

## Value

An S3 object of class `simbase_list` if `data` is grouped, and an object of class `simbase_covar` otherwise.

## Examples

```
# obtain a dataset for demonstration
dataset <- simulate_dataset();

# calculate a simbase without transforms
simbase_covar(dataset, c('f', 'E', 'rho', 'E_dyn'));

# calculate a simbase with log-transformed f
simbase_covar(dataset, c('f', 'E', 'rho', 'E_dyn'), list(f = scales::log_trans()));
```

```
# if we group the dataset, we get a simbase_list object
simbase_covar(dplyr::group_by(dataset, country), c('f', 'E', 'rho', 'E_dyn'));
```

---

simbase_labeler	<i>Default labelling function for simbase objects</i>
-----------------	---

---

### Description

Each simbase object should have a label which can be used for differentiating different simulations. This function tries to simplify the label generation.

### Usage

```
simbase_labeler(data, simbase_class, transforms)
```

### Arguments

data	The dataset for the calculation of the basic simulation data.
simbase_class	The class of the simbase object for which the label is to be generated. Currently, only "simbase_covar" is supported.
transforms	The transforms applied to variables in the dataset. Must be objects of class trans or class transform (see function trans_new in package scales).

### Details

Primarily, this function is intended to be called as a default from `simbase_covar`. It can also serve as a template for creating custom labelling functions.

### Value

A string for labelling a simbase object.

---

simbase_list	<i>Wrapper for the simbase_* functions for grouped data</i>
--------------	---

---

### Description

If a function of the simbase\_\* family encounters grouped data (as caused by `group_by`), it should invoke `simbase_list` to create a collection of separate simbases for each group.

### Usage

```
simbase_list(data, simbase_constructor, ..., suffix = "_lst")
```

**Arguments**

data	A grouped dataset (see <a href="#">group_by</a> )
simbase_constructor	A function which returns a simbase_* object, like <a href="#">simbase_covar</a>
...	Further arguments passed to the simbase_* function.
suffix	Suffix to be added to the individual simbase labels if they are all the same (see details).

**Value**

A simbase\_list object; this is essentially a [tibble](#) with the grouping columns of data and a column .simbase which contains the simbase\_\* objects.

**Technical details**

Currently, the "simbase\_\* family" only consists of [simbase\\_covar](#) (although, in a broader sense, simbase\_list can also be thought to be part of this "family"). It is planned to add further simulation types in a later release.

The functions of the simbase\_\* family support label generation (see e.g. [simbase\\_covar](#)). These functions should generate the label *before* invoking simbase\_list, so that there is a common label for all of the simbases; simbase\_list adds a suffix suffix. A warning is issued if the labels of the different simbases are not all equal; no suffix is added in this case.

---

simulate\_conditionally

*Add simulated values to a dataset conditionally, based on a simbase\_\* object*

---

**Description**

Add simulated values to a dataset conditionally, based on a simbase\_\* object

**Usage**

```
simulate_conditionally(data, simbase, force_positive = TRUE, ...)
```

**Arguments**

data	The dataset where simulated values are added to. The dataset has to contain at least one variable which is also included in the simbase_* object.
simbase	Basic data object for the simulation, as calculated e.g. by <a href="#">simbase_covar</a> or <a href="#">simbase_list</a> .
force_positive	If TRUE, the resulting values are forced to be $\geq 0$ .
...	further arguments passed to or from other methods.

## Details

Given a `simbase_*` object, this function adds simulated values to a dataset, conditional on the values of some of the variables already contained in the dataset.

## Value

The modified dataset data with simulated values.

## Examples

```
# add simulated tension data based on a simbase stored in WoodSimulatR
dataset <- data.frame(E_dyn = rnorm(n = 100, mean = 12500, sd = 2200));
dataset_t <- simulate_conditionally(dataset, ws_t)

# add simulated bending data
dataset_be <- simulate_conditionally(dataset, ws_be)
```

---

```
simulate_conditionally.simbase_list
      Add simulated values to a dataset conditionally, based on a
      simbase_list object
```

---

## Description

Add simulated values to a dataset conditionally, based on a `simbase_list` object

## Usage

```
## S3 method for class 'simbase_list'
simulate_conditionally(
  data,
  simbase,
  force_positive = TRUE,
  ...,
  error_when_groups_missing = TRUE
)
```

## Arguments

<code>data</code>	The dataset where simulated values are added to.
<code>simbase</code>	Basic data object for the simulation, as calculated by <a href="#">simbase_list</a> .
<code>force_positive</code>	If TRUE, the resulting values are forced to be $\geq 0$ .
<code>...</code>	further arguments passed to or from other methods.
<code>error_when_groups_missing</code>	Whether to raise an error if for a certain value combination in the grouping variables no dedicated <code>simbase</code> exists (see details).

## Details

Simulating values based on a `simbase_list` object has some special aspects compared to that of other `simbase_*` objects, (see `simulate_conditionally`).

In particular, a `simbase_list` object stores `simbases` for specific value combinations within the grouping variables.

These grouping variables must also be present in data.

If there is a value combination in these grouping variables for which no dedicated `simbase` object exists, this will lead to NA values in the columns to be simulated and either to an error (if `error_when_groups_missing = TRUE`) or to a warning.

Due to the internal call to `nest` and subsequent call to `unnest`, the returned dataset will be ordered according to the grouping variables in the `simbase`, with any grouping variable combinations missing in the `simbase` coming last.

## Value

The modified dataset data with simulated values.

## Examples

```
# create a simbase_list object for the values of subsets = c('AT', 'DE')
dataset_0 <- simulate_dataset(subsets = c('AT', 'DE'));
simbase <- simbase_covar(dplyr::group_by(dataset_0, country), c('f', 'E', 'E_dyn'));

# simulate on another dataset
dataset <- data.frame(E_dyn = rnorm(n = 100, mean = 12500, sd = 2200), country = 'AT');
dataset_1 <- simulate_conditionally(dataset, simbase);
head(dataset_1);

# warning if for some value of country we don't have an entry in the simbase
dataset$country <- 'CH';
dataset_2 <- simulate_conditionally(dataset, simbase, error_when_groups_missing = FALSE);
head(dataset_2);
```

---

`simulate_dataset`

*Generate an artificial dataset with correlated variables*

---

## Description

Generate an artificial dataset with correlated variables and defined means and standard deviations.

## Usage

```
simulate_dataset(
  n = 5000,
  subsets = 4,
  random_seed = NULL,
```

```

simbase = WoodSimulatR::ws_t_logf,
loadtype = NULL,
...,
RNGversion = "3.6.0"
)

```

## Arguments

n	Number of rows in the dataset
subsets	Either NULL, or a <code>data.frame</code> describing the subsets (see details) or a character vector or named numeric vector suitable for argument <code>country</code> in <a href="#">get_subsample_definitions</a> .
random_seed	Allows to set an integer seed value for the random number generator to achieve reproducible results (see also <a href="#">set.seed</a> ).
simbase	An object of class <a href="#">simbase_covar</a> or <a href="#">simbase_list</a> . In particular, one of the simbases stored in WoodSimulatR may be used – see <a href="#">simbase</a> .
loadtype	For passing on to <a href="#">get_subsample_definitions</a> . A string with either "t" (for material tested in tension) or "be" (for material tested in edgewise bending). Is only used if the <code>simbase</code> doesn't contain a field <code>loadtype</code> or if the <code>loadtype</code> is ambiguous or not equal to "t" or "be".
...	arguments passed on to <a href="#">get_subsample_definitions</a> .
RNGversion	In WoodSimulatR 0.5, the <code>RNGversion</code> had been fixed to <code>RNGversion = "3.5.0"</code> , but this setting now causes a warning because the random number generator was changed in R version 3.6.0 (see <a href="#">RNGversion</a> ). For perfect reproducibility of results from WoodSimulatR 0.5, one should set <code>RNGversion = "3.5.0"</code> .

## Details

In the package WoodSimulatR, a number of predefined base values for simulation are stored – see [simbase](#).

Using a character vector for the argument `subsets` leads to subsets as equal in size as possible.

The argument `subsets` enables differing means and standard deviations for different subsamples. There are several possible usages:

- If `subsets = NULL`, the information about means and standard deviations is taken from the `simbase`. There can still be different means and standard deviations if `simbase` is an object of class [simbase\\_list](#).
- If a numeric vector or a character vector, it is used as argument `country` in an internal call to [get\\_subsample\\_definitions](#).
- If a dataset, there are the following requirements:
  - *identifier columns*: The dataset has to have one or more discrete-valued *identifier columns* (usually character vectors or factors) which uniquely identify each row. These *identifier columns* are named "country" and "subsample" in the standard case as yielded by [get\\_subsample\\_definitions](#). In the general case, the identifier columns are detected as those columns which are not named `share`, `species`, `loadtype` or `literature` and which do not end in `_mean` or `_sd`. If the argument `simbase` is of class [simbase\\_list](#), further restrictions apply (see below).

- *means and standard deviations*: For at least one of the variables defined in the `simbase`, also the mean *and* the standard deviation need to be given in each row; the column names for this data must be the name of the respective variable(s) from the `simbase`, suffixed by `_mean` and `_sd`, respectively.
- *optional*: A column `share` can be used to create subsamples of different sizes proportional to the values in `share`.

The argument `simbase` can be either an object of class `simbase_covar` or of class `simbase_list`.

- various predefined `simbase_covar` objects are available in `WoodSimulatR` – see `simbase`.
- for objects of class `simbase_list`, additional restrictions apply:
  1. the object may only have grouping variable(s) which are also *identifier columns* according to the `subsets` definition above – if the `subsets` argument is *not* a data frame, the *identifier columns* are "country" and "subsample".
  2. The value combinations in the *identifier columns* have to match those which the `subsets` argument leads to (see also `get_subsample_definitions`).

Both the means and standard deviations in the subsample definitions (see `get_subsample_definitions`) as well as the values in the `simbase` depend on the way the destructive testing of the sawn timber was done. If the `simbase` has a field `loadtype` (see also `simbase_covar`), this value is used in the call to `get_subsample_definitions`. Otherwise, the `loadtype` has to be passed directly to the present function unless no call to `get_subsample_definitions` is necessary (this depends on the value of `subsets` – see above). If a `loadtype` has been defined, a variable `loadtype` is also created in the resulting dataset for reference.

Negative values in any numeric column of the result dataset are forced to zero.

If `random_seed` is not `NULL`, reproducibility of results is enforced by using `set.seed` with arguments `kind='Mersenne-Twister'` and `normal.kind='Inversion'`, and by calling `RNGversion` with argument `RNGversion`.

If `random_seed` is not `NULL`, the random number generator is reset at the end of the function using `set.seed(NULL)` and `RNGversion(toString(getRversion()))`.

## Examples

```
simulate_dataset(n = 10, subsets = 1, random_seed = 1)

# As the loadtype is defined in the simbase, the argument loadtype is ignored
# with a warning
simulate_dataset(n = 10, subsets = 1, random_seed = 1, loadtype = 'be')

# Two subsamples
simulate_dataset(n = 10, subsets = 2, random_seed = 1)

# Two subsamples from pre-defined countries
simulate_dataset(n = 10, subsets = c('at', 'de'), random_seed = 1)

# Two subsamples from pre-defined countries with different sample sizes
simulate_dataset(n = 10, subsets = c(at = 3, de = 2), random_seed = 1)
```

# Index

## \* datasets

- gdp\_data, [2](#)
  
- gdp\_data, [2](#), [4](#)
- get\_subsample\_definitions, [3](#), [13](#), [14](#)
- get\_transform\_names, [5](#)
- group\_by, [8–10](#)
  
- is\_primitive, [6](#)
  
- nest, [12](#)
  
- RNGversion, [13](#), [14](#)
  
- set.seed, [13](#), [14](#)
- simbase, [6](#), [13](#), [14](#)
- simbase\_covar, [5–7](#), [7](#), [9](#), [10](#), [13](#), [14](#)
- simbase\_labeler, [5](#), [9](#)
- simbase\_list, [8](#), [9](#), [10–14](#)
- simulate\_conditionally, [10](#), [12](#)
- simulate\_conditionally.simbase\_list,  
[11](#)
- simulate\_dataset, [3](#), [4](#), [12](#)
  
- tibble, [8](#), [10](#)
  
- unnest, [12](#)
  
- ws\_be(simbase), [6](#)
- ws\_be\_logf(simbase), [6](#)
- ws\_be\_te(simbase), [6](#)
- ws\_be\_te\_logf(simbase), [6](#)
- ws\_be\_tr(simbase), [6](#)
- ws\_be\_tr\_logf(simbase), [6](#)
- ws\_t(simbase), [6](#)
- ws\_t\_logf(simbase), [6](#)
- ws\_t\_te(simbase), [6](#)
- ws\_t\_te\_logf(simbase), [6](#)
- ws\_t\_tr(simbase), [6](#)
- ws\_t\_tr\_logf(simbase), [6](#)