

# Package ‘tidysynthesis’

May 8, 2026

**Title** A Common API for Synthesizing Data

**Version** 0.1.3

**Maintainer** Aaron R. Williams <awilliams@urban.org>

**Description** A system built on 'tidymodels' for generating synthetic tabular data. We provide tools for ordering a sequential synthesis, feature and target engineering, sampling, hyperparameter tuning, enforcing constraints, and adding extra noise during a synthesis.

**URL** <https://ui-research.github.io/tidysynthesis-documentation/>

**BugReports** <https://github.com/UrbanInstitute/tidysynthesis/issues>

**Depends** R (>= 4.1.0)

**Imports** dplyr, forcats, parsnip, pillar, purrr, progressr, recipes, rlang, rsample, stringr, tibble, tidyr (>= 1.0.0), tune, vctrs, workflows, yardstick, ExtDist, dapper

**Suggests** hardhat, palmerpenguins, poissonreg, randomForest, ranger, rpart, rpart.LAD (>= 0.1.2), testthat (>= 2.1.0), usethis, VGAM

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Aaron R. Williams [aut, cre] (ORCID:

<<https://orcid.org/0000-0001-5564-1938>>),

Kyle Ueyama [aut],

Gabe Morrison [aut] (ORCID: <<https://orcid.org/0009-0008-1815-5920>>),

Jeremy Seeman [aut] (ORCID: <<https://orcid.org/0000-0003-3526-3209>>),

Elyse McFalls [ctb],

Claire Morton [ctb],

Livia Mucciolo [ctb],

Madeline Pickens [ctb],

Noah Zwiefel [ctb],

The Urban Institute [cph]

Repository CRAN

Date/Publication 2026-03-16 20:50:02 UTC

## Contents

.name_to_inspect . . . . .	3
acs_conf . . . . .	4
acs_conf_nw . . . . .	5
acs_start . . . . .	6
acs_start_nw . . . . .	6
add_noise_cat_unif . . . . .	7
add_noise_disc_gaussian . . . . .	9
add_noise_disc_laplace . . . . .	10
add_noise_gaussian . . . . .	11
add_noise_kde . . . . .	12
add_noise_laplace . . . . .	14
add_sequence_factor . . . . .	15
add_sequence_manual . . . . .	16
add_sequence_numeric . . . . .	16
collapse_na . . . . .	18
constraints . . . . .	18
constraints_api . . . . .	19
construct_extractors . . . . .	21
construct_models . . . . .	23
construct_noise . . . . .	24
construct_recipes . . . . .	26
construct_samplers . . . . .	28
construct_tuners . . . . .	30
convert_level_to_na . . . . .	31
convert_na_to_level . . . . .	32
enforce_custom_na . . . . .	33
enforce_na . . . . .	33
enforce_schema . . . . .	34
example_na . . . . .	35
expand_na . . . . .	35
invert . . . . .	36
invert.step_BoxCox . . . . .	37
invert.step_log . . . . .	38
invert.step_YeoJohnson . . . . .	39
ks_distance . . . . .	40
noise . . . . .	41
postsynth_to_roadmap . . . . .	42
postsynth_to_synth_spec . . . . .	43
presynth . . . . .	44
print.constraints . . . . .	45
print.noise . . . . .	46
print.postsynth . . . . .	47

print.presynth . . . . . 48

print.replicates . . . . . 49

print.schema . . . . . 50

print.start\_method . . . . . 50

print.synth\_spec . . . . . 51

print.visit\_sequence . . . . . 52

replicates . . . . . 52

replicates\_api . . . . . 53

roadmap . . . . . 54

sample\_glm . . . . . 56

sample\_lm . . . . . 57

sample\_ranger . . . . . 58

sample\_rpart . . . . . 59

schema . . . . . 60

schema\_api . . . . . 61

start\_method . . . . . 63

start\_method\_api . . . . . 64

start\_resample . . . . . 65

synthesize . . . . . 66

synth\_spec . . . . . 67

synth\_spec\_extractor\_api . . . . . 70

synth\_spec\_is\_component . . . . . 71

synth\_spec\_model\_api . . . . . 72

synth\_spec\_noise\_api . . . . . 73

synth\_spec\_recipes\_api . . . . . 74

synth\_spec\_sampler\_api . . . . . 75

synth\_spec\_tuner\_api . . . . . 77

tune\_synthesis . . . . . 79

update\_presynth . . . . . 81

update\_synth\_spec . . . . . 82

visit\_sequence . . . . . 83

visit\_sequence\_api . . . . . 84

**Index** **86**

---

.name_to_inspect	<i>constant mapping between component names and inspections</i>
------------------	---

---

**Description**

constant mapping between component names and inspections

**Usage**

.name\_to\_inspect

**Format**

An object of class `list` of length 7.

---

acs_conf	<i>American Community Survey confidential microdata (with weights)</i>
----------	--

---

**Description**

An extract constructed from the 2019 American Community Survey containing a survey sample of  $n = 1500$  Nebraska respondents, with survey weights included.

**Usage**

```
acs_conf
```

**Format**

acs\_conf:

A data frame with 1,500 rows and 12 columns:

**county** fct, county

**gq** fct, group quarter kind

**sex** fct, sex

**marst** fct, marital status

**hcovany** fct, health insurance status

**empstat** fct, employment status; contains empty levels.

**classwkr** fct, employment kind (ex: self-employed, etc.); contains "N/A" levels.

**age** dbl, age (in years)

**famsize** dbl, household/family size

**transit\_time** dbl, transit time to work (in minutes)

**inctot** dbl, annual income; contains missing values

**wgt** dbl, survey weight

**Details**

Original data source: Steven Ruggles, Sarah Flood, Matthew Sobek, Daniel Backman, Annie Chen, Grace Cooper, Stephanie Richards, Renae Rogers, and Megan Schouweiler. IPUMS USA: Version 15.0 [dataset]. Minneapolis, MN: IPUMS, 2024. <https://doi.org/10.18128/D010.V15.0>

**Source**

<https://usa.ipums.org/usa/>

---

acs_conf_nw	<i>American Community Survey confidential microdata (without weights)</i>
-------------	---

---

### Description

An extract constructed from the 2019 American Community Survey containing a survey sample of n = 1500 Nebraska respondents, with survey weights included.

### Usage

acs\_conf\_nw

### Format

acs\_conf\_nw:

A data frame with 1,500 rows and 11 columns:

**county** fct, county

**gq** fct, group quarter kind

**sex** fct, sex

**marst** fct, marital status

**hcovany** fct, health insurance status

**empstat** fct, employment status; contains empty levels.

**classwkr** fct, employment kind (ex: self-employed, etc.); contains "N/A" levels.

**age** dbl, age (in years)

**famsize** dbl, household/family size

**transit\_time** dbl, transit time to work (in minutes)

**inctot** dbl, annual income; contains missing values

### Details

Original data source: Steven Ruggles, Sarah Flood, Matthew Sobek, Daniel Backman, Annie Chen, Grace Cooper, Stephanie Richards, Renae Rogers, and Megan Schouweiler. IPUMS USA: Version 15.0 [dataset]. Minneapolis, MN: IPUMS, 2024. <https://doi.org/10.18128/D010.V15.0>

### Source

<https://usa.ipums.org/usa/>

---

acs_start	<i>American Community Survey starting microdata (with weights)</i>
-----------	--

---

**Description**

An extract constructed from the 2019 American Community Survey containing a survey sample of n = 500 Nebraska respondents, with survey weights included.

**Usage**

acs\_start

**Format**

acs\_start:

A data frame with 500 rows and 5 columns:

**county** fct, county

**gq** fct, group quarter kind

**sex** fct, sex

**marst** fct, marital status

**wgt** dbl, survey weight

**Details**

Original data source: Steven Ruggles, Sarah Flood, Matthew Sobek, Daniel Backman, Annie Chen, Grace Cooper, Stephanie Richards, Renae Rogers, and Megan Schouweiler. IPUMS USA: Version 15.0 [dataset]. Minneapolis, MN: IPUMS, 2024. <https://doi.org/10.18128/D010.V15.0>

**Source**

<https://usa.ipums.org/usa/>

---

acs_start_nw	<i>American Community Survey starting microdata (without weights)</i>
--------------	---

---

**Description**

An extract constructed from the 2019 American Community Survey containing a survey sample of n = 500 Nebraska respondents, with survey weights included.

**Usage**

acs\_start\_nw

**Format**

acs\_start\_nw:

A data frame with 500 rows and 4 columns:

**county** fct, county

**gq** fct, group quarter kind

**sex** fct, sex

**marst** fct, marital status

**Details**

Original data source: Steven Ruggles, Sarah Flood, Matthew Sobek, Daniel Backman, Annie Chen, Grace Cooper, Stephanie Richards, Renae Rogers, and Megan Schouweiler. IPUMS USA: Version 15.0 [dataset]. Minneapolis, MN: IPUMS, 2024. <https://doi.org/10.18128/D010.V15.0>

**Source**

<https://usa.ipums.org/usa/>

---

add_noise_cat_unif	<i>Inject noise into a categorical random variable by mixing a sample of uniform records into the predictions.</i>
--------------------	--

---

**Description**

Inject noise into a categorical random variable by mixing a sample of uniform records into the predictions.

**Usage**

```
add_noise_cat_unif(  
  model,  
  new_data,  
  conf_model_data,  
  outcome_var,  
  col_schema,  
  pred,  
  unif_prop,  
  resample_props = NULL,  
  observed_levels = FALSE  
)
```

**Arguments**

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
unif_prop	A proportion of records to resample with uniform noise
resample_props	An optional named vector of probabilities for resampling, defaults to uniform over all levels supplied in col_schema.
observed_levels	An optional Boolean to only resample from observed levels in the confidential data.

**Value**

A numeric vector with noise added to each prediction

**Examples**

```
conf_model_data <- mtcars|>
  dplyr::mutate(gear = factor(.data[["gear"]]))

col_schema <- list(
  "dtype" = "fct",
  "levels" = c("3", "4", "5"),
  "na_prop" = 0
)

add_noise_cat_unif(
  model = conf_model_data,
  new_data = NULL,
  conf_model_data = NULL,
  outcome_var = "gear",
  col_schema = col_schema,
  pred = factor(c(rep("3", 10), rep("4", 10), rep("5", 10))),
  unif_prop = 0.5
)
```

---

 add\_noise\_disc\_gaussian

*Add discrete normal noise with mean 0 to predicted values with constant variance*

---

### Description

Add discrete normal noise with mean 0 to predicted values with constant variance

### Usage

```
add_noise_disc_gaussian(
  model,
  new_data,
  conf_model_data,
  outcome_var,
  col_schema,
  pred,
  variance = NULL,
  rho = NULL,
  sensitivity = NULL,
  increment = 1
)
```

### Arguments

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
variance	float, sampling variance for additive noise
rho	float, alternative privacy loss budget prescribed by the Gaussian mechanism under rho-zero-concentrated differential privacy.
sensitivity	float, alternative sample sensitivity prescribed by the Gaussian mechanism under rho-zero-concentrated differential privacy.
increment	Numeric indicating space between discrete noise samples, defaults to 1. Note that this does not impact the noise sampling variance, as the increment rescales noise distributions specified by sampling variance.

### Value

A numeric vector with noise added to each prediction

**Examples**

```

add_noise_disc_gaussian(
  model = NULL,
  new_data = NULL,
  conf_model_data = NULL,
  outcome_var = NULL,
  col_schema = NULL,
  pred = 1:100,
  variance = 3
)

```

---

```

add_noise_disc_laplace

```

*Add discrete Laplace noise with mean 0 to predicted values with constant variance*

---

**Description**

Add discrete Laplace noise with mean 0 to predicted values with constant variance

**Usage**

```

add_noise_disc_laplace(
  model,
  new_data,
  conf_model_data,
  outcome_var,
  col_schema,
  pred,
  variance = NULL,
  epsilon = NULL,
  sensitivity = NULL,
  increment = 1
)

```

**Arguments**

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
variance	float, sampling variance for additive noise

epsilon	float, alternative privacy loss budget prescribed by the Laplace mechanism under epsilon differential privacy.
sensitivity	float, alternative sample sensitivity prescribed by the Laplace mechanism under epsilon differential privacy.
increment	Numeric indicating space between discrete noise samples, defaults to 1. Note that this does not impact the noise sampling variance, as the increment rescales noise distributions specified by sampling variance.

**Value**

A numeric vector with noise added to each prediction

**Examples**

```
add_noise_disc_laplace(
  model = NULL,
  new_data = NULL,
  conf_model_data = NULL,
  outcome_var = NULL,
  col_schema = NULL,
  pred = 1:100,
  variance = 3
)
```

---

add_noise_gaussian	<i>Add normal noise with mean 0 to predicted values with constant variance</i>
--------------------	--

---

**Description**

Add normal noise with mean 0 to predicted values with constant variance

**Usage**

```
add_noise_gaussian(
  model,
  new_data,
  conf_model_data,
  outcome_var,
  col_schema,
  pred,
  variance = NULL,
  rho = NULL,
  sensitivity = NULL
)
```

**Arguments**

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
variance	Sampling variance for additive noise
rho	Alternative privacy loss budget prescribed by the Gaussian mechanism under rho-zero-concentrated differential privacy.
sensitivity	Alternative sample sensitivity prescribed by the Gaussian mechanism under rho-zero-concentrated differential privacy.

**Value**

A numeric vector with noise added to each prediction

**Examples**

```
add_noise_gaussian(
  model = NULL,
  new_data = NULL,
  conf_model_data = NULL,
  outcome_var = NULL,
  col_schema = NULL,
  pred = 1:100,
  variance = 3
)
```

---

add_noise_kde	<i>Add normal noise to predicted values with variances calculated for ntiles using Gaussian kernel density estimators</i>
---------------	---

---

**Description**

Add normal noise to predicted values with variances calculated for ntiles using Gaussian kernel density estimators

**Usage**

```
add_noise_kde(
  model,
  new_data,
  conf_model_data,
  outcome_var,
  col_schema,
  pred,
  exclusions = NULL,
  n_ntiles = NULL,
  obs_per_ntile = NULL,
  ties_method = "collapse",
  sd_scale = 1
)
```

**Arguments**

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
exclusions	Numeric values that should not receive extra noise
n_ntiles	The number of ntiles
obs_per_ntile	A numeric for the minimum number of observations to be in an ntile. Cannot be used in conjunction with the n_ntiles argument.
ties_method	The ntiles approach to adding noise requires a one-to-one mapping from model-generated values to ntiles in the original data. The methods "collapse", "random", and "exclusions" deal with situations where the ntiles lack unique bounds. "collapse" collapses ntile breaks to preserve the one-to-one relationship; "random" adds a small random perturbation to the derived boundaries; finally, "exclusions" treats ntile tie values as derived exclusions.
sd_scale	float, a positive number to scale the estimated KDE variance. Defaults to 1.0

**Value**

A numeric vector with noise added to each prediction

**Examples**

```
add_noise_kde(
  model = NULL,
  new_data = tibble::tibble(x = 1:100),
  conf_model_data = tibble::tibble(x = 1:100),
```

```

outcome_var = "x",
col_schema = NULL,
pred = 1:100,
n_ntiles = 4
)

```

---

add_noise_laplace	<i>Add Laplace noise with mean 0 to predicted values with constant variance</i>
-------------------	---

---

### Description

Add Laplace noise with mean 0 to predicted values with constant variance

### Usage

```

add_noise_laplace(
  model,
  new_data,
  conf_model_data,
  outcome_var,
  col_schema,
  pred,
  variance = NULL,
  epsilon = NULL,
  sensitivity = NULL
)

```

### Arguments

model	A model_spec or a list of model_specs from library(parsnip)
new_data	A data frame used to generate predictions
conf_model_data	A data frame for estimating the predictive model
outcome_var	A string name representing the outcome variable
col_schema	A list of column schema specifications for the new variable
pred	A vector of values predicted by the model
variance	Sampling variance for additive noise
epsilon	Alternative privacy loss budget prescribed by the Laplace mechanism under epsilon differential privacy.
sensitivity	Alternative sample sensitivity prescribed by the Laplace mechanism under epsilon differential privacy.

**Value**

A numeric vector with noise added to each prediction

**Examples**

```
add_noise_laplace(
  model = NULL,
  new_data = NULL,
  conf_model_data = NULL,
  outcome_var = NULL,
  col_schema = NULL,
  pred = 1:100,
  variance = 3
)
```

---

add\_sequence\_factor     *Add to visit sequence for factor variables*

---

**Description**

Add to visit sequence for factor variables

**Usage**

```
add_sequence_factor(roadmap, ..., method = c("entropy", "gini"))
```

**Arguments**

roadmap	A roadmap object
...	<b>tidy-select</b> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.
method	A quoted name for the method used to sort the visit_sequence. Current methods include "entropy" and "gini".

**Value**

An updated visit\_sequence

**Examples**

```
roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
) |>
  add_sequence_factor(dplyr::where(is.factor), method = "gini")
```

---

add\_sequence\_manual *Add to visit sequence using a manual method*

---

**Description**

Add to visit sequence using a manual method

**Usage**

```
add_sequence_manual(roadmap, ...)
```

**Arguments**

roadmap	A roadmap object.
...	<a href="#">tidy-select</a> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.

**Value**

An updated roadmap object.

**Examples**

```
roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
) |>  
  add_sequence_manual(  
    c("inctot", "hcovany", "empstat", "classwkr", "age", "famsize",  
      "transit_time")  
  )
```

---

add\_sequence\_numeric *Add to visit sequence for numeric variables*

---

**Description**

Add to visit sequence for numeric variables

**Usage**

```
add_sequence_numeric(
  roadmap,
  ...,
  method = c("correlation", "proportion", "weighted total", "absolute weighted total",
             "weighted absolute total"),
  cor_var = NULL,
  na.rm = FALSE,
  cor_use = "everything"
)
```

**Arguments**

roadmap	A roadmap object
...	<a href="#">tidy-select</a> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like <code>x:y</code> can be used to select a range of variables.
method	A quoted name for the method used to sort the <code>visit_sequence</code>
cor_var	A numeric variable for the correlation method
na.rm	Boolean that if TRUE, removes NA values from computations
cor_use	A string correlation data method passed to <code>stats::cor</code> if using. If <code>na.rm == TRUE</code> then defaults to <code>complete.obs</code> . See <code>?stats::cor</code> for more options.

**Value**

An updated `visit_sequence`

**Examples**

```
roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
) |>
  add_sequence_numeric(
    dplyr::where(is.numeric),
    method = "correlation",
    cor_var = "age",
    na.rm = TRUE
  )
```

---

<code>collapse_na</code>	<i>Collapse data frames with <code>_NA</code> variables to coerce related variables to include NA</i>
--------------------------	---

---

**Description**

Collapse data frames with `_NA` variables to coerce related variables to include NA

**Usage**

```
collapse_na(data)
```

**Arguments**

`data` A data frame with columns ending in `_NA`

**Value**

A data frame with no `_NA` columns and NA values

**Examples**

```
example_na_expanded <- expand_na(data = example_na)

collapse_na(data = example_na_expanded)
```

---

<code>constraints</code>	<i>Create a constraints object</i>
--------------------------	------------------------------------

---

**Description**

Create a constraints object

**Usage**

```
constraints(
  schema,
  constraints_df_num = NULL,
  constraints_df_cat = NULL,
  max_z_num = 0,
  max_z_cat = 0
)
```

**Arguments**

schema	A schema object
constraints_df_num	A specially formatted data frame with constraints to be imposed during the synthesis process. See examples for formatting.
constraints_df_cat	A specifically formatted data frame with constraints to be imposed during the synthesis process.
max_z_num	Numeric vector(s) for the number of times a value should be resampled before hardbounding if it violates a constraint.
max_z_cat	Numeric vector(s) for the number of times a value should be resampled before hardbounding if it violates a constraint.

**Value**

A constraints object.

**Examples**

```
constraints(
  schema = schema(
    conf_data = mtcars |> dplyr::mutate(vs = factor(vs)),
    start_data = dplyr::select(mtcars, cyl)
  ),
  constraints_df_num = tibble::tribble(
    ~var, ~min, ~max, ~conditions,
    # ensure all mpg values are greater than 0
    "mpg", 0, Inf, "TRUE",
    # ensure when cyl == 6, mpg is less than 15
    "mpg", -Inf, 15, "cyl == 6",
    # ensure disp is always between 0 and 150
    "disp", 0, 150, "TRUE"
  ),
  constraints_df_cat = tibble::tribble(
    ~var, ~allowed, ~forbidden, ~conditions,
    # ensure vs != 1 when gear >= 4
    "vs", NA, 1, "gear >= 5",
    # ensure vs == 1 when gear >= 4
    "vs", 0, NA, "gear == 4"
  )
)
```

---

constraints_api	<i>Add, update, or reset a constraints object within an existing roadmap.</i>
-----------------	---

---

**Description**

Add, update, or reset a constraints object within an existing roadmap.

**Usage**

```
add_constraints(roadmap, constraints)
```

```
update_constraints(roadmap, ...)
```

```
reset_constraints(roadmap)
```

**Arguments**

roadmap	A roadmap object
constraints	A constraints object.
...	Optional named parameters passed to constraints().

**Value**

A new roadmap object.

A roadmap object with added constraints.

A roadmap object with updated constraints.

A roadmap object with reset constraints.

**Examples**

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
constraints_df_num <-  
  tibble::tribble(~var, ~min, ~max, ~conditions,  
                 "transit_time", 0, 300, "TRUE")  
  
constraints <- constraints(  
  schema = rm[["schema"]],  
  constraints_df_num = constraints_df_num,  
  max_z_num = 0  
)  
  
rm |>  
  add_constraints(constraints)  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)
```

```
constraints_df_num <-  
  tibble::tribble(~var, ~min, ~max, ~conditions,  
                 "transit_time", 0, 300, "TRUE")  
  
constraints <- constraints(  
  schema = rm[["schema"]],  
  constraints_df_num = constraints_df_num,  
  max_z_num = 0  
)  
  
rm |>  
  update_constraints(constraints)  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
constraints_df_num <-  
  tibble::tribble(~var, ~min, ~max, ~conditions,  
                 "transit_time", 0, 300, "TRUE")  
  
constraints <- constraints(  
  schema = rm[["schema"]],  
  constraints_df_num = constraints_df_num,  
  max_z_num = 0  
)  
  
rm <- rm |>  
  add_constraints(constraints)  
  
reset_constraints(rm)
```

---

construct\_extractors *Construct a list of extractors for parsnip models*

---

## Description

Construct a list of extractors for parsnip models

## Usage

```
construct_extractors(  
  roadmap,  
  default_extractor = NULL,  
  custom_extractors = NULL  
)
```

**Arguments**

roadmap            A roadmap object  
default\_extractor        An extractor from library(parsnip)  
custom\_extractors       A formatted list of extractors

**Value**

A named list of extractors

**Examples**

```
# construct_extractors() can create a sequence of extractors using a fully-default  
# approach, a hybrid approach, or a fully-customized approach. All approaches  
# require a roadmap and extractors.
```

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)
```

```
# Fully-default approach
```

```
construct_extractors(  
  roadmap = rm,  
  default_extractor = parsnip::extract_fit_engine  
)
```

```
# Hybrid approach
```

```
construct_extractors(  
  roadmap = rm,  
  default_extractor = parsnip::extract_fit_engine,  
  custom_extractors = list(  
    list(vars = "hcovany", extractor = parsnip::extract_parameter_dials)  
  )  
)
```

```
# Fully-customized approach
```

```
construct_extractors(  
  roadmap = rm,  
  custom_extractors = list(  
    list(  
      vars = c("hcovany", "empstat", "classwkr"),  
      extractor = parsnip::extract_fit_engine  
    ),  
    list(  
      vars = c("age", "famsize", "transit_time", "inctot"),  
      extractor = parsnip::extract_parameter_dials  
    )  
  )  
)
```

```

    )
  )
)

```

---

construct_models	<i>Construct a list of models for synthesis</i>
------------------	---

---

## Description

Construct a list of models for synthesis

## Usage

```

construct_models(
  roadmap,
  default_regression_model = NULL,
  default_classification_model = NULL,
  custom_models = NULL
)

```

## Arguments

roadmap	A roadmap object
default_regression_model	A parsnip model object used for regression in numeric outcome variables
default_classification_model	A parsnip model object used for classification in categorical outcome variables
custom_models	A formatted list with parsnip model objects explicitly paired with every variable in the visit_sequence

## Value

A named list of models

## Examples

```

# construct_models() can create a sequence of models using a fully-default
# approach, a hybrid approach, or a fully-customized approach. All approaches
# require a roadmap and model objects.

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>

```

```

parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")

lm_mod <- parsnip::linear_reg() |>
  parsnip::set_engine("lm") |>
  parsnip::set_mode(mode = "regression")

# Fully-default approach

construct_models(
  roadmap = rm,
  default_regression_model = lm_mod,
  default_classification_model = rpart_mod_class
)

# Hybrid approach

construct_models(
  roadmap = rm,
  default_regression_model = lm_mod,
  default_classification_model = rpart_mod_class,
  custom_models = list(
    list(vars = "age", model = lm_mod)
  )
)

# Fully-customized approach

construct_models(
  roadmap = rm,
  custom_models = list(
    list(vars = c("hcovany", "empstat", "classwkr"), model = rpart_mod_class),
    list(vars = c("age", "famsize", "transit_time", "inctot"), model = rpart_mod_reg)
  )
)

```

---

construct\_noise

*Construct a list of noise objects for synthesis*


---

## Description

Construct a list of noise objects for synthesis

## Usage

```
construct_noise(
```

```
roadmap,  
default_regression_noise = NULL,  
default_classification_noise = NULL,  
custom_noise = NULL  
)
```

### Arguments

roadmap	A roadmap object
default_regression_noise	A noise function for regression models
default_classification_noise	A noise function for classification models
custom_noise	A formatted list of noise functions

### Value

A named list of noise

### Examples

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
noise_defaults <- construct_noise(  
  roadmap = rm,  
  default_regression_noise = noise(),  
  default_classification_noise = noise()  
)  
  
# construct_noise() can create a sequence of noise objects using a  
# fully-default approach, a hybrid approach, or a fully-customized approach.  
# All approaches require a roadmap and noise objects.  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
noise_reg <- noise(  
  add_noise = TRUE,  
  mode = "regression",  
  noise_fun = add_noise_gaussian  
)  
  
noise_class <- noise(  
  add_noise = TRUE,  
  mode = "classification",
```

```

    noise_fun = add_noise_cat_unif
  )

# Fully-default approach

construct_noise(
  roadmap = rm,
  default_regression_noise = noise_reg,
  default_classification_noise = noise_class
)

# Hybrid approach

noise_reg2 <- noise(
  add_noise = TRUE,
  mode = "regression",
  noise_fun = add_noise_disc_gaussian
)

construct_noise(
  roadmap = rm,
  default_regression_noise = noise_reg,
  default_classification_noise = noise_class,
  custom_noise = list(
    list(vars = "age", noise = noise_reg2)
  )
)

# Fully-customized approach

construct_noise(
  roadmap = rm,
  custom_noise = list(
    list(vars = c("hcovany", "empstat", "classwkr"), noise = noise_class),
    list(vars = c("age", "famsize", "transit_time", "inctot"), noise = noise_reg)
  )
)

```

---

construct\_recipes

*Construct a sequence of model recipes for sequential synthesis*


---

### Description

Construct a sequence of model recipes for sequential synthesis

### Usage

```

construct_recipes(
  roadmap,

```

```

    default_regression_steps = NULL,
    default_classification_steps = NULL,
    custom_steps = NULL
  )

```

### Arguments

```

roadmap      A roadmap object
default_regression_steps
              A list containing one or more recipes::step_*( )
default_classification_steps
              A list containing one or more recipes::step_*( )
custom_steps A list of lists containing one or more recipes::step_*( )

```

### Value

A list of formulas

### Examples

```

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

```

```
construct_recipes(rm)
```

# construct\_recipes() can create a sequence of recipes using a fully-default approach, a hybrid approach, or a fully-customized approach. All approaches require a roadmap and steps.

```

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

```

```

step1 <- function(x) {
  x |>
  recipes::step_center(recipes::all_predictors(), id = "center")
}

```

# Fully-default approach

```

construct_recipes(
  roadmap = rm,
  default_regression_steps = step1,
  default_classification_steps = step1
)

```

# Hybrid approach



```
custom_samplers
      A formatted list of sampler functions
```

**Value**

A named list of samplers

**Examples**

```
# construct_samplers() can create a sequence of samplers using a fully-default
# approach, a hybrid approach, or a fully-customized approach. All approaches
# require a roadmap and samplers.

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

# Fully-default approach

construct_samplers(
  roadmap = rm,
  default_regression_sampler = sample_lm,
  default_classification_sampler = sample_rpart
)

# Hybrid approach

construct_samplers(
  roadmap = rm,
  default_regression_sampler = sample_lm,
  default_classification_sampler = sample_rpart,
  custom_samplers = list(
    list(vars = "hcovany", sampler = sample_rpart)
  )
)

# Fully-customized approach

construct_samplers(
  roadmap = rm,
  custom_samplers = list(
    list(vars = c("hcovany", "empstat", "classwkr"), sampler = sample_rpart),
    list(vars = c("age", "famsize", "transit_time", "inctot"), sampler = sample_lm)
  )
)
```

---

construct_tuners	<i>Construct a list of tuning grids for hyperparameter tuning predictive models</i>
------------------	---

---

### Description

Construct a list of tuning grids for hyperparameter tuning predictive models

### Usage

```
construct_tuners(  
  roadmap,  
  default_regression_tuner = NULL,  
  default_classification_tuner = NULL,  
  custom_tuners = NULL  
)
```

### Arguments

roadmap	A roadmap object
default_regression_tuner	A tuner.
default_classification_tuner	A tuner.
custom_tuners	A formatted list of tuners.

### Value

A named list of tuners

### Examples

```
# construct_tuners() can create a sequence of tuners using a fully-default  
# approach, a hybrid approach, or a fully-customized approach. All approaches  
# require a roadmap and tuners.  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
tuner_reg <- list(  
  v = 3,  
  grid = 3,  
  metrics = yardstick::metric_set(yardstick::rmse)  
)  
  
tuner_cat <- list(  
  v = 3,  
  grid = 3,  
  metrics = yardstick::metric_set(yardstick::rmse)  
)
```

```

    v = 3,
    grid = 3,
    metrics = yardstick::metric_set(yardstick::roc_auc)
  )

# Fully-default approach

construct_tuners(
  roadmap = rm,
  default_regression_tuner = tuner_reg,
  default_classification_tuner = tuner_cat
)

# Hybrid approach

tuner_cat2 <- list(
  v = 3,
  grid = 3,
  metrics = yardstick::metric_set(yardstick::precision)
)

construct_tuners(
  roadmap = rm,
  default_regression_tuner = tuner_reg,
  default_classification_tuner = tuner_cat,
  custom_tuners = list(
    list(vars = "hcovany", tuner = tuner_cat2)
  )
)

# Fully-customized approach

construct_tuners(
  roadmap = rm,
  custom_tuners = list(
    list(vars = c("hcovany", "empstat", "classwkr"), tuner = tuner_reg),
    list(vars = c("age", "famsize", "transit_time", "inctot"), tuner = tuner_cat)
  )
)

```

---

convert\_level\_to\_na    *Convert "NA" values to NA for categorical variables*

---

### Description

Convert "NA" values to NA for categorical variables

### Usage

```
convert_level_to_na(data)
```

**Arguments**

data                    A data frame or tibble

**Value**

A data frame or tibble with "NA" converted to NA

**Examples**

```
data <- data.frame(  
  x1 = c(1, 2, NA),  
  x2 = c("1", "2", "NA"),  
  x3 = factor(c("1", "2", "NA")),  
  x4 = factor(c("b", "NA", "a"), ordered = TRUE)  
)  
  
convert_level_to_na(data)
```

---

convert\_na\_to\_level    *Convert NA values to "NA" for categorical variables*

---

**Description**

Convert NA values to "NA" for categorical variables

**Usage**

```
convert_na_to_level(data)
```

**Arguments**

data                    A data frame or tibble

**Value**

A data frame or tibble with NA converted to "NA"

**Examples**

```
data <- data.frame(  
  x1 = c(1, 2, NA),  
  x2 = c("1", "2", NA),  
  x3 = factor(c("1", "2", NA)),  
  x4 = factor(c("b", NA, "a"), levels = c("b", NA, "a"), ordered = TRUE)  
)  
  
convert_na_to_level(data)
```

---

enforce_custom_na	<i>Redefine NA value for a dataset.</i>
-------------------	---

---

**Description**

Redefine NA value for a dataset.

**Usage**

```
enforce_custom_na(data, col_schema)
```

**Arguments**

data	A data.frame object
col_schema	A col_schema from a schema object

**Value**

A data.frame

**Examples**

```
# create custom NA filter
example_na_custom <- example_na |>
  tidyr::replace_na(
    list("wages" = -999)
  )

example_na_expanded_custom <- enforce_custom_na(
  data = example_na_custom,
  col_schema = list(
    "wages" = list(
      dtype = "dbl",
      na_value = -999
    )
  )
)
```

---

enforce_na	<i>Add missing values where values should be missing according to _NA variables</i>
------------	---

---

**Description**

Add missing values where values should be missing according to \_NA variables

**Usage**

```
enforce_na(data)
```

**Arguments**

data                    A synthetic data frame with `_NA` columns

**Value**

A synthetic data frame with `_NA` columns that converts values that are labelled missing in an `_NA` variable to missing in the corresponding variable

**Examples**

```
example_na_expanded <- expand_na(data = example_na)
enforce_na(data = example_na_expanded)
```

---

enforce_schema	<i>Enforce a roadmap's schema on its existing data</i>
----------------	--

---

**Description**

Enforce a roadmap's schema on its existing data

**Usage**

```
enforce_schema(roadmap)
```

**Arguments**

roadmap                A roadmap object

**Value**

A roadmap object with modified `conf_data`, `start_data`, and schema information.

**Examples**

```
rm <- roadmap(conf_data = acs_conf, start_data = acs_start) |>
  update_schema(na_numeric_to_ind = TRUE)
enforce_schema(rm)
```

---

example_na	<i>A df with different types of missingness</i>
------------	---

---

**Description**

A df with different types of missingness

**Usage**

```
example_na
```

**Format**

A tibble with 200 observations and 6 variables:

**age** Age of respondent

**sex** Sex of respondent with missingness at random

**labor\_force** Labor force status of respondent with structural missingness

**hours** Hours work of respondent with missingness at random

**wages** Wages earned with structural missingness

---

expand_na	<i>Add new variables that indicate if a value is "missing" or "not missing" for original variables that contain NA</i>
-----------	--

---

**Description**

Add new variables that indicate if a value is "missing" or "not missing" for original variables that contain NA

**Usage**

```
expand_na(
  data,
  types = c("chr", "dbl", "fct", "lgl", "int", "ord"),
  skip_vars = NULL
)
```

**Arguments**

**data** A data frame

**types** A vector of variables types to expand

**skip\_vars** A character vector of variables that shouldn't be expanded

**Value**

An augmented data frame with the original variables and new variables that contain the missingness patterns of variables with NA

**Examples**

```
expand_na(data = example_na, type = c("dbl", "int"))
```

---

 invert

*An S3 method for inverting a step*


---

**Description**

An S3 method for inverting a step

**Usage**

```
invert(object, predictions, ...)
```

**Arguments**

object	A recipe after fitting a model
predictions	A data frame with .pred
...	Other arguments

**Value**

A tibble with inverted model-generated values

**Examples**

```
data <- tibble::tibble(
  y = rlnorm(n = 1000, meanlog = 0, sdlog = 1),
  x = rnorm(n = 1000)
)

adj <- recipes::recipe(y ~ x, data = data) |>
  recipes::step_BoxCox(recipes::all_outcomes()) |>
  recipes::prep()

invert(
  object = adj$steps[[1]],
  predictions = tibble::tibble(.pred = adj[["template"]][["y"]])
)
```

---

invert.step\_BoxCox      *Invert a Box-Cox transformation*

---

## Description

Invert a Box-Cox transformation

## Usage

```
## S3 method for class 'step_BoxCox'  
invert(object, predictions, ...)
```

## Arguments

object	A recipe after fitting a model
predictions	A data frame with .pred
...	Other arguments

## Value

A tibble with the Box-Cox transformation inverted for .pred

## Examples

```
data <- tibble::tibble(  
  y = rlnorm(n = 1000, meanlog = 0, sdlog = 1),  
  x = rnorm(n = 1000)  
)  
  
adj <- recipes::recipe(y ~ x, data = data) |>  
  recipes::step_BoxCox(recipes::all_outcomes()) |>  
  recipes::prep()  
  
invert(  
  object = adj$steps[[1]],  
  predictions = tibble::tibble(.pred = adj[["template"]][["y"]])  
)
```

---

invert.step_log	<i>Invert a log transformation</i>
-----------------	------------------------------------

---

## Description

Invert a log transformation

## Usage

```
## S3 method for class 'step_log'  
invert(object, predictions, ...)
```

## Arguments

object	A recipe after fitting a model
predictions	A data frame with .pred
...	Other arguments

## Value

A tibble with the log transformation inverted for .pred

## Examples

```
data <- tibble::tibble(  
  y = rlnorm(n = 1000, meanlog = 0, sdlog = 1),  
  x = rnorm(n = 1000)  
)  
  
adj <- recipes::recipe(y ~ x, data = data) |>  
  recipes::step_log(recipes::all_outcomes()) |>  
  recipes::prep()  
  
invert(  
  object = adj$steps[[1]],  
  predictions = tibble::tibble(.pred = adj[["template"]][["y"]])  
)
```

---

`invert.step_YeoJohnson`*Invert a Yeo-Johnson transformation*

---

## Description

Invert a Yeo-Johnson transformation

## Usage

```
## S3 method for class 'step_YeoJohnson'  
invert(object, predictions, ...)
```

## Arguments

<code>object</code>	A recipe after fitting a model
<code>predictions</code>	A data frame with <code>.pred</code>
<code>...</code>	Other arguments

## Value

A tibble with the Yeo\_johnson transformation inverted for `.pred`

## Examples

```
data <- tibble::tibble(  
  y = rlnorm(n = 1000, meanlog = 0, sdlog = 1),  
  x = rnorm(n = 1000)  
)  
  
adj <- recipes::recipe(y ~ x, data = data) |>  
  recipes::step_YeoJohnson(recipes::all_outcomes()) |>  
  recipes::prep()  
  
invert(  
  object = adj$steps[[1]],  
  predictions = tibble::tibble(.pred = adj[["template"]][["y"]])  
)
```

---

ks_distance	<i>Kolmogorov-Smirnov distance</i>
-------------	------------------------------------

---

**Description**

Kolmogorov-Smirnov distance

**Usage**

```
ks_distance(data, ...)
```

```
## S3 method for class 'data.frame'
```

```
ks_distance(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

```
ks_distance_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Not currently used.
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For <code>_vec()</code> functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For <code>_vec()</code> functions, a numeric vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	This is a placeholder for now and will be added when case_weights are added to tidysynthesis.

**Value**

For `ks_distance_vec()`, a single numeric value (or NA).

A single numeric value (or NA).

A single numeric value (or NA).

**Examples**

```
ks1 <- data.frame(x = 1:100, y = 101:200)
```

```
ks_distance(data = ks1, truth = x, estimate = y)
```

```
ks1 <- data.frame(x = 1:100, y = 101:200)

ks_distance(data = ks1, truth = x, estimate = y)

ks1 <- data.frame(x = 1:100, y = 101:200)

ks_distance_vec(truth = ks1$x, estimate = ks1$y)
```

---

noise *Create a noise object*

---

### Description

Create a noise object

### Usage

```
noise(add_noise = FALSE, mode = "regression", noise_func = NULL, ...)
```

### Arguments

add_noise	Boolean, TRUE if adding noise
mode	String, one of "regression" or "classification"
noise_func	A function that adds noise to
...	Optional named additional arguments to pass to noise_func(...)

### Value

A noise object

### Examples

```
# create default noise object
noise()

# create noise object for classification
noise(
  add_noise = TRUE,
  mode = "classification",
  noise_func = add_noise_cat_unif
)

# create noise object for regression
noise(
  add_noise = TRUE,
  mode = "regression",
```

```

noise_func = add_noise_kde,
n_ntiles = 10
)

```

---

`postsynth_to_roadmap` *Convert a incompletely synthesized postsynth into a new roadmap using the same roadmap settings from the incomplete synthesis.*

---

### Description

In the new roadmap, `start_data` will now include the initial starting data along with the previously synthesized variables. The `visit_sequence` will consist of, in the same order as the original visit sequence, all unsynthesized variables. The schema will reflect the original schema but with an updated set of `synth_vars`. The constraints will be similar to those initially inputted, but all rows in the constraints data frames for already synthesized variables will be excluded.

### Usage

```
postsynth_to_roadmap(postsynth)
```

### Arguments

`postsynth`      A postsynth object generated with `keep_workflows == TRUE`

### Details

To update components of the roadmap use the `reset_*` and `add_*` functions. For example, to change the `visit_sequence` which by default follows the same order as the initial visit sequence, apply `reset_visit_sequence()` and then `add_sequence_*` to the newly created roadmap.

### Value

A roadmap object.

### Examples

```

# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>

```

```

parsnip::set_mode(mode = "classification")

synth_spec1 <- synth_spec(
  default_regression_model = rpart_mod_reg,
  default_regression_sampler = sample_rpart,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)

# create a presynth object
# use defaults for noise, constraints, and replicates
presynth1 <- presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)

# synthesize with keep_workflows = TRUE
set.seed(1)
postsynth1 <- synthesize(
  presynth = presynth1,
  keep_workflows = TRUE
)

# return roadmap (will return original roadmap if synthesis completes).
new_rmap <- postsynth_to_roadmap(postsynth1)

```

---

postsynth\_to\_synth\_spec

*Convert a incompletely synthesized postsynth into a new synth\_spec using the same synth\_spec settings from the incomplete synthesis.*

---

### Description

Convert a incompletely synthesized postsynth into a new synth\_spec using the same synth\_spec settings from the incomplete synthesis.

### Usage

```
postsynth_to_synth_spec(postsynth)
```

### Arguments

postsynth      A postsynth object generated with keep\_workflows == TRUE

### Value

A synth\_spec object that is similar to the initial synth\_spec but updated to remove information about already-synthesized variables.

## Examples

```
# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")

synth_spec1 <- synth_spec(
  default_regression_model = rpart_mod_reg,
  default_regression_sampler = sample_rpart,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)

# create a presynth object
# use defaults for noise, constraints, and replicates
presynth1 <- presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)

# synthesize with keep_workflows = TRUE
set.seed(1)
postsynth1 <- synthesize(
  presynth = presynth1,
  keep_workflows = TRUE
)

# return roadmap (will return original roadmap if synthesis completes).
new_spec <- postsynth_to_synth_spec(postsynth1)
```

---

presynth

*Create a presynth object*

---

## Description

Create a presynth object

## Usage

```
presynth(roadmap, synth_spec)
```

**Arguments**

roadmap            A roadmap object from roadmap().  
synth\_spec        A synth\_spec object from synth\_spec().

**Value**

A presynth object.

**Examples**

```
# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")

synth_spec1 <- synth_spec(
  default_regression_model = rpart_mod_reg,
  default_regression_sampler = sample_rpart,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)

# create a presynth object
# use defaults for noise, constraints, and replicates
presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)
```

---

print.constraints        *Print the constraints object to the console with formatting*

---

**Description**

Print the constraints object to the console with formatting

**Usage**

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

**Arguments**

x                    A constraints object  
 ...                  further arguments passed to or from other methods (not currently used).

**Value**

Invisibly returns the input constraints object.

**Examples**

```
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

constraints_df_num <-
  tibble::tribble(~var, ~min, ~max, ~conditions,
                 "transit_time", 0, 300, "TRUE")

constraints <- constraints(
  schema = rm[["schema"]],
  constraints_df_num = constraints_df_num,
  max_z_num = 0
)

print(constraints)
```

---

```
print.noise
```

*Print the noise object to the console with formatting*

---

**Description**

Print the noise object to the console with formatting

**Usage**

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

**Arguments**

x                    A noise object  
 ...                  further arguments passed to or from other methods (not currently used).

**Value**

Invisibly returns the input noise object.

**Examples**

```
print(noise())
```

---

```
print.postsynth      Print the postsynth object to the console with formatting
```

---

**Description**

Print the postsynth object to the console with formatting

**Usage**

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

**Arguments**

x	A postsynth object
...	further arguments passed to or from other methods (not currently used).

**Value**

Invisibly returns the input postsynth object.

**Examples**

```
# create roadmap  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rpart_mod_reg <- parsnip::decision_tree() |>  
  parsnip::set_engine(engine = "rpart") |>  
  parsnip::set_mode(mode = "regression")  
  
rpart_mod_class <- parsnip::decision_tree() |>  
  parsnip::set_engine(engine = "rpart") |>  
  parsnip::set_mode(mode = "classification")  
  
synth_spec1 <- synth_spec(  
  default_regression_model = rpart_mod_reg,  
  default_regression_sampler = sample_rpart,  
  default_classification_model = rpart_mod_class,  
  default_classification_sampler = sample_rpart  
)  
  
# create a presynth object
```

```
# use defaults for noise, constraints, and replicates
presynth1 <- presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)

# synthesize!
set.seed(1)
postsynth1 <- synthesize(presynth = presynth1)

print(postsynth1)
```

---

print.presynth      *print method for presynth objects*

---

## Description

print method for presynth objects

## Usage

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

## Arguments

x                    A presynth object  
...                  further arguments passed to or from other methods (not currently used).

## Value

A presynth object

## Examples

```
# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")
```

```
synth_spec1 <- synth_spec(  
  default_regression_model = rpart_mod_reg,  
  default_regression_sampler = sample_rpart,  
  default_classification_model = rpart_mod_class,  
  default_classification_sampler = sample_rpart  
)  
  
# create a presynth object  
# use defaults for noise, constraints, and replicates  
presynth <- presynth(  
  roadmap = rm,  
  synth_spec = synth_spec1  
)  
  
print(presynth)
```

---

print.replicates	<i>Print the replicates object to the console with formatting</i>
------------------	---

---

## Description

Print the replicates object to the console with formatting

## Usage

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

## Arguments

x	A replicates object
...	further arguments passed to or from other methods (not currently used).

## Value

Invisibly returns the input replicates object.

## Examples

```
rep <- replicates(  
  start_data_replicates = 2,  
  model_sample_replicates = 2,  
  end_to_end_replicates = 2  
)  
  
print(rep)
```

---

<code>print.schema</code>	<i>Print the schema object to the console with formatting</i>
---------------------------	---

---

**Description**

Print the schema object to the console with formatting

**Usage**

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

**Arguments**

<code>x</code>	A schema object
<code>...</code>	further arguments passed to or from other methods (not currently used).

**Value**

Invisibly returns the input schema object.

**Examples**

```
# default inferred schema  
schema1 <- schema(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
print(schema1)
```

---

<code>print.start_method</code>	<i>Print the start_method object to the console with formatting</i>
---------------------------------	---

---

**Description**

Print the start\_method object to the console with formatting

**Usage**

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

**Arguments**

x                    A start\_method object  
...                   further arguments passed to or from other methods (not currently used).

**Value**

A start\_method object

**Examples**

```
print(start_method())
```

---

```
print.synth_spec            Print the replicates object to the console with formatting
```

---

**Description**

Print the replicates object to the console with formatting

**Usage**

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

**Arguments**

x                    A replicates object  
...                   further arguments passed to or from other methods (not currently used).

**Value**

A synth\_spec object

**Examples**

```
synth_spec <- synth_spec()  
print(synth_spec)
```

---

```
print.visit_sequence
```

*Print method for visit\_sequence objects*

---

**Description**

Print method for visit\_sequence objects

**Usage**

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

**Arguments**

x                    A visit\_sequence object  
...                   further arguments passed to or from other methods (not currently used).

**Value**

Invisibly returns the input visit\_sequence object.

**Examples**

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
print(rm[["visit_sequence"]])
```

---

```
replicates
```

*Create a replicates object*

---

**Description**

Create a replicates object

**Usage**

```
replicates(  
  start_data_replicates = 1,  
  model_sample_replicates = 1,  
  end_to_end_replicates = 1  
)
```

**Arguments**

- `start_data_replicates`  
The number of starting data replicates to use. Note that if no `start_method` is provided, all start data replicates will be identical.
- `model_sample_replicates`  
The number of replicates for the conditional modeling process, including modeling and sampling new synthetic values.
- `end_to_end_replicates`  
The number of replicates for the entire synthesis process, including all previously specified steps.

**Value**

A new replicates object.

**Examples**

```
replicates(
  start_data_replicates = 2,
  model_sample_replicates = 2,
  end_to_end_replicates = 2
)
```

---

<code>replicates_api</code>	<i>Add, update, or reset a replicates object within an existing roadmap.</i>
-----------------------------	--

---

**Description**

Add, update, or reset a replicates object within an existing roadmap.

**Usage**

```
add_replicates(roadmap, replicates)
update_replicates(roadmap, ...)
reset_replicates(roadmap)
```

**Arguments**

- `roadmap` A roadmap object
- `replicates` A replicates object.
- `...` Optional named parameters passed to `replicates()`.

**Value**

- A new roadmap object.
- A new roadmap object with the added replicates.
- A new roadmap object with updated replicates.
- A new roadmap object with reset replicates.

**Examples**

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
new_replicates <- replicates(end_to_end_replicates = 2)  
  
rm |>  
  add_replicates(new_replicates)  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rm |>  
  update_replicates(start_data_replicates = 3)  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rm <- rm |>  
  add_replicates(replicates(start_data_replicates = 3))  
  
reset_replicates(roadmap = rm)
```

---

roadmap

*Create a roadmap*

---

**Description**

A roadmap is a container object that aggregates information required to specify the order of operations for synthesis modeling and sampling steps.

**Usage**

```
roadmap(  
  conf_data,  
  start_data,  
  start_method = NULL,  
  schema = NULL,  
  visit_sequence = NULL,  
  replicates = NULL,  
  constraints = NULL  
)
```

**Arguments**

<code>conf_data</code>	A data.frame of confidential data.
<code>start_data</code>	A data.frame of starting data used to initialize the process.
<code>start_method</code>	An optional <code>start_method</code> object.
<code>schema</code>	An optional schema object.
<code>visit_sequence</code>	An optional <code>visit_sequence</code> object.
<code>replicates</code>	An optional <code>replicates</code> object.
<code>constraints</code>	An optional <code>constraints</code> object.

**Details**

Users initiate a roadmap object with `conf_data` and `start_data`. All other objects will either be completed with defaults or specified interactively via the provided API.

**Value**

A new roadmap object.

**Examples**

```
roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw,  
  start_method = start_method(  
    start_func = start_resample, n = 1000  
  )  
)
```

---

sample_glm	<i>Sample the conditional distribution created by a generalized linear model</i>
------------	--

---

## Description

Currently, logistic and poisson regression are supported using `parsnip` and the standard `glm` engine. Note that poisson regression requires the suggested `poissonreg` library.

## Usage

```
sample_glm(model, new_data, conf_data)
```

## Arguments

model	A "model_fit" object created by <code>parsnip</code>
new_data	A data frame with predictors
conf_data	A data frame with original confidential predictors

## Value

A numeric vector of predictions

## Examples

```
acs_conf <- acs_conf |>
  tidyr::drop_na()

logistic_mod <- parsnip::logistic_reg() |>
  parsnip::set_engine("glm") |>
  parsnip::set_mode(mode = "classification")

classification_rec <- recipes::recipe(hcovany ~ ., data = acs_conf)

model_class <- workflows::workflow() |>
  workflows::add_model(spec = logistic_mod) |>
  workflows::add_recipe(recipe = classification_rec) |>
  parsnip::fit(data = acs_conf)

set.seed(1)
sample1 <- sample_glm(
  model = model_class,
  new_data = acs_conf[1:3, ],
  conf_data = acs_conf
)
```

---

sample_lm	<i>Sample the conditional distribution created by a linear model</i>
-----------	--

---

## Description

Sample the conditional distribution created by a linear model

## Usage

```
sample_lm(model, new_data, conf_data)
```

## Arguments

model	A "model_fit" object created by <code>parsnip::linear_reg()</code>
new_data	A data frame with predictors
conf_data	A data frame with original confidential predictors

## Value

A numeric vector of predictions

## Examples

```
lm_mod <- parsnip::linear_reg() |>
  parsnip::set_engine("lm") |>
  parsnip::set_mode(mode = "regression")

regression_rec <- recipes::recipe(inctot ~ ., data = acs_conf)

model_reg <- workflows::workflow() |>
  workflows::add_model(spec = lm_mod) |>
  workflows::add_recipe(recipe = regression_rec) |>
  parsnip::fit(data = acs_conf)

set.seed(1)
sample1 <- sample_lm(
  model = model_reg,
  new_data = acs_conf[1:3, ],
  conf_data = acs_conf
)
```

---

`sample_ranger`*Sample the conditional distribution created by a ranger rf model*

---

## Description

Sample the conditional distribution created by a ranger rf model

## Usage

```
sample_ranger(model, new_data, conf_data)
```

## Arguments

<code>model</code>	A "model_fit" object created by <code>parsnip::ranger()</code>
<code>new_data</code>	A data frame with predictors
<code>conf_data</code>	A data frame with original confidential predictors

## Value

A numeric vector of predictions

## Examples

```
rf_mod_regression <- parsnip::rand_forest(trees = 500, min_n = 1) |>
  parsnip::set_engine(engine = "ranger") |>
  parsnip::set_mode(mode = "regression") |>
  parsnip::set_args(quantreg = TRUE)

regression_rec <- recipes::recipe(age ~ ., data = acs_conf)

model_reg <- workflows::workflow() |>
  workflows::add_model(spec = rf_mod_regression) |>
  workflows::add_recipe(recipe = regression_rec) |>
  parsnip::fit(data = acs_conf)

set.seed(1)
sample1 <- sample_ranger(
  model = model_reg,
  new_data = acs_conf[1:3, ],
  conf_data = acs_conf
)
```

---

sample_rpart	<i>Sample the conditional distribution created by a CART model</i>
--------------	--

---

## Description

Sample the conditional distribution created by a CART model

## Usage

```
sample_rpart(model, new_data, conf_data, ignore_zeros = TRUE)
```

## Arguments

model	A "model_fit" object created by rpart
new_data	A data frame with predictors
conf_data	A data frame with original confidential predictors
ignore_zeros	Should a vector of all 0 observations return NA for the I-diversity calculation. Defaults to TRUE.

## Value

A numeric vector of predictions

## Examples

```
rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine("rpart") |>
  parsnip::set_mode(mode = "regression")

regression_rec <- recipes::recipe(inctot ~ ., data = acs_conf)

model_reg <- workflows::workflow() |>
  workflows::add_model(spec = rpart_mod_reg) |>
  workflows::add_recipe(recipe = regression_rec) |>
  parsnip::fit(data = acs_conf)

set.seed(1)
sample1 <- sample_rpart(
  model = model_reg,
  new_data = acs_conf[1:3, ],
  conf_data = acs_conf
)

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine("rpart") |>
  parsnip::set_mode(mode = "classification")
```

```

classification_rec <- recipes::recipe(hcovany ~ ., data = acs_conf)

model_reg <- workflows::workflow() |>
  workflows::add_model(spec = rpart_mod_class) |>
  workflows::add_recipe(recipe = classification_rec) |>
  parsnip::fit(data = acs_conf)

set.seed(1)
sample1 <- sample_rpart(
  model = model_reg,
  new_data = acs_conf[1:10, ],
  conf_data = acs_conf
)

```

---

schema

*Generate a schema object.*

---

## Description

Generate a schema object.

## Usage

```

schema(
  conf_data,
  start_data,
  col_schema = NULL,
  enforce = TRUE,
  coerce_to_factors = FALSE,
  coerce_to_doubles = FALSE,
  na_factor_to_level = TRUE,
  na_numeric_to_ind = TRUE
)

```

## Arguments

<code>conf_data</code>	A data frame to be synthesized.
<code>start_data</code>	A data frame with starting variables.
<code>col_schema</code>	An optional named list of columns in the confidential data with their properties, including data type and factor levels. If NULL or only partially specified, <code>col_schema</code> will be inferred from the confidential data. See example code for formatting.
<code>enforce</code>	Boolean that if true, will preprocess both <code>conf_data</code> and <code>start_data</code> to align with <code>col_schema</code> and the arguments below.
<code>coerce_to_factors</code>	Boolean that if true, coerces categorical data types (chr, fct, ord) to base R factors when <code>enforce_schema</code> is called.

`coerce_to_doubles`  
Boolean that if true, coerces columns specified as `dbl` in `col_schema` to base R doubles when `enforce_schema` is called.

`na_factor_to_level`  
Boolean that if true, applies `convert_level_to_na()` to factor variables when `enforce_schema` is called.

`na_numeric_to_ind`  
Boolean that if true, applies `expand_na()` to numeric data to create logical missingness indicators when `enforce_schema` is called.

### Value

A schema object.

### Examples

```
conf_data <- data.frame(
  var1 = c("1", "1", "2"),
  var2 = c(1L, 2L, 3L),
  var3 = c(1.1, 2.2, 3.3)
)

start_data <- dplyr::select(conf_data, var1)

# default inferred schema
schema(
  conf_data = conf_data,
  start_data = start_data
)

# overwriting factor levels
schema(
  conf_data = conf_data,
  start_data = start_data,
  col_schema = list(
    "var1" = list(
      "dtype" = "fct",
      "levels" = c("1", "2", "3")
    )
  ),
  coerce_to_factors = TRUE
)
```

**Description**

Add, update, or reset a schema object within an existing roadmap.

**Usage**

```
add_schema(roadmap, schema)
```

```
update_schema(roadmap, ...)
```

```
reset_schema(roadmap)
```

**Arguments**

roadmap	A roadmap object
schema	A schema object.
...	Optional named parameters passed to schema().

**Value**

A new roadmap object.

A roadmap object with added schema.

A roadmap object with updated schema.

A roadmap object with reset schema.

**Examples**

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
acs_schema <- schema(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw,  
  na_numeric_to_ind = TRUE  
)  
  
rm |>  
  add_schema(schema = acs_schema)  
  
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rm |>  
  update_schema(na_numeric_to_ind = TRUE)
```

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rm <- rm |>  
  update_schema(na_numeric_to_ind = TRUE)  
  
reset_schema(roadmap = rm)
```

---

start_method	<i>Create a start_method object.</i>
--------------	--------------------------------------

---

### Description

A `start_method` gets executed prior to running a synthesis. This modifies the `start_data`, typically randomly, to provide greater disclosure risk protections.

### Usage

```
start_method(start_func = NULL, ...)
```

### Arguments

<code>start_func</code>	A function that accepts and returns a <code>data.frame</code> . If none provided <code>.identity_start()</code> is used.
<code>...</code>	Optional keyword arguments passed to <code>start_func(...)</code>

### Value

A `start_method` object

### Examples

```
# basic usage  
start_method(start_func = start_resample)  
  
# adjust the number of observations  
start_method(  
  start_func = start_resample,  
  start_data = acs_start_nw,  
  n = 10  
)  
  
# adjust the number of observations and use all combinations as support  
start_method(  
  start_func = start_resample,
```

```
start_data = acs_start_nw,  
n = 10,  
inv_noise_scale = 1,  
support = "all"  
)
```

---

start_method_api	<i>Add, update, or reset a start method within an existing roadmap.</i>
------------------	---

---

### Description

Add, update, or reset a start method within an existing roadmap.

### Usage

```
add_start_method(roadmap, start_method)
```

```
update_start_method(roadmap, ...)
```

```
remove_start_method(roadmap)
```

### Arguments

roadmap	A roadmap object
start_method	A start_method object.
...	Optional named parameters passed to start_method()

### Value

A new roadmap object.  
A new roadmap object with added start\_method.  
A new roadmap object with updated start\_method.  
A new roadmap object with removed start\_method.

### Examples

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw,  
)  
  
add_start_method(  
  roadmap = rm,  
  start_method = start_method()  
)
```

```

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

update_start_method(
  roadmap = rm,
  start_method = start_method()
)

rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw,
  start_method = start_method()
)

remove_start_method(
  roadmap = rm
)

```

---

start_resample	<i>Specify a resampling scheme for start_data</i>
----------------	---

---

## Description

Specify a resampling scheme for start\_data

## Usage

```

start_resample(
  start_data,
  n = NULL,
  inv_noise_scale = NULL,
  support = c("observed", "all")
)

```

## Arguments

start_data	A data.frame
n	An optional integer sample size. If unspecified, n = nrow(start_data)
inv_noise_scale	An optional parameter to set randomized noise to the proportions of records with different start_data characteristics. Corresponds to a privacy loss budget under epsilon differential privacy.
support	A string that specifies the method of resampling from the start_data domain.

**Value**

A `start_method` object for resampling starting data

**Examples**

```
start_method(  
  start_func = start_resample, n = 1000  
)
```

---

`synthesize`

*Synthesize a data set*

---

**Description**

Synthesize a data set

**Usage**

```
synthesize(  
  presynth,  
  progress = FALSE,  
  keep_workflows = FALSE,  
  keep_partial = FALSE  
)
```

**Arguments**

<code>presynth</code>	A <code>presynth</code> object created by <code>presynth()</code> .
<code>progress</code>	Logical, if TRUE, displays progress. Default is FALSE, no progress displayed.
<code>keep_workflows</code>	Logical, if TRUE, returns derived roadmap components in <code>postsynth(s)</code> . Default is FALSE.
<code>keep_partial</code>	Logical, if TRUE, saves partial synthesis to output. Default is FALSE, no intermediate outputs saved.

**Value**

A `postsynth` object.

**Examples**

```

# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")

synth_spec1 <- synth_spec(
  default_regression_model = rpart_mod_reg,
  default_regression_sampler = sample_rpart,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)

# create a presynth object
# use defaults for noise, constraints, and replicates
presynth1 <- presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)

# synthesize!
set.seed(1)
postsynth1 <- synthesize(presynth = presynth1)

```

---

synth\_spec

*Create a synth\_spec object*


---

**Description**

The `synth_spec` object holds specifications for modeling and sampling components for sequential synthetic data generation. Each component has an associated `construct_*` function called when creating a `presynth` object.

**Usage**

```

synth_spec(
  default_regression_model = NULL,
  default_classification_model = NULL,
  custom_models = NULL,

```

```

default_regression_steps = NULL,
default_classification_steps = NULL,
custom_steps = NULL,
default_regression_sampler = NULL,
default_classification_sampler = NULL,
custom_samplers = NULL,
default_regression_noise = NULL,
default_classification_noise = NULL,
custom_noise = NULL,
default_regression_tuner = NULL,
default_classification_tuner = NULL,
custom_tuners = NULL,
default_extractor = NULL,
custom_extractors = NULL,
invert_transformations = TRUE,
enforce_na = TRUE
)

```

### Arguments

`default_regression_model`  
A `model_spec` object from `library(parsnip)` for use in regression models.

`default_classification_model`  
A `model_spec` object from `library(parsnip)` for use in classification models.

`custom_models` A list of named lists each with two elements: `vars` for variable names, and `model` for their associated model. from `library(parsnip)`.

`default_regression_steps`  
A list of `recipe::step_function(s)` from `library(recipes)` for use in regression models.

`default_classification_steps`  
A list of `recipe::step_function(s)` from `library(recipes)` for use in classification models.

`custom_steps` A list of named lists each with two elements: `vars` for variable names, and `steps` for their associated recipe.

`default_regression_sampler`  
A sampling function for drawing new values from regression models.

`default_classification_sampler`  
A sampling function for drawing new values from classification models.

`custom_samplers`  
A list of named lists each with two elements: `vars` for variable names, and `sampler` for their associated sampler

`default_regression_noise`  
A noise function for adding noise to numeric values.

`default_classification_noise`  
A noise function for adding noise to classification values.

`custom_noise` A list of named lists each with two elements: `vars` for variable names, and `noise` for their associated noise

default_regression_tuner	A tuner from <code>library(tune)</code> for use in regression models.
default_classification_tuner	A tuner from <code>library(tune)</code> for use in classification models.
custom_tuners	A list of named lists each with two elements: <code>vars</code> for variable names, and <code>tuner</code> for their associated tuner
default_extractor	An optional method for extracting workflows or extracts from workflows.
custom_extractors	A list of named lists each with two elements: <code>vars</code> for variable names, and <code>extractor</code> for their associated extractor
invert_transformations	A Boolean for if outcome variable transformations applied through recipes should be inverted during synthesis. recipes need ids that begin with "outcome".
enforce_na	A Boolean for if NA values should be added into the synthetic data with <code>enforce_na()</code> during synthesis. An alternative approach is to add the NA values after synthesis

**Value**

A `synth_spec` object

**Examples**

```
rpart_mod <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

lm_mod <- parsnip::linear_reg() |>
  parsnip::set_engine("lm") |>
  parsnip::set_mode(mode = "regression")

step1 <- function(x) {
  x |>
  recipes::step_center(recipes::all_predictors(), id = "center")
}

step2 <- function(x) {
  x |>
  recipes::step_scale(recipes::all_predictors(), id = "scale")
}

step3 <- function(x) { x |> step1() |> step2() }

synth_spec(
  default_regression_model = rpart_mod,
  custom_models = list(
    list("vars" = c("var1", "var2"),
        "model" = lm_mod)
  ),
)
```

```

default_regression_steps = step1,
custom_steps = list(
  list("vars" = c("var2", "var3"),
    "steps" = step2),
  list("vars" = c("var4"),
    "steps" = step3)
),
default_regression_sampler = sample_rpart,
custom_samplers = list(
  list("vars" = c("var1", "var2"),
    "sampler" = sample_lm)
)
)

```

---

synth\_spec\_extractor\_api

*Add, update, or remove extractors from a synth\_spec object*

---

### Description

Add, update, or remove extractors from a synth\_spec object

### Usage

```
add_custom_extractors(synth_spec, ...)
```

```
update_custom_extractors(synth_spec, ...)
```

```
remove_custom_extractors(synth_spec)
```

### Arguments

synth_spec	A synth_spec object
...	Optional named lists with two elements, vars and extractor, mapping variable names to extractors.

### Value

A new synth\_spec object.

A new synth\_spec object with added custom extractors.

A new synth\_spec object with updated custom extractors.

A new synth\_spec object with removed custom extractors.

**Examples**

```
synth_spec <- synth_spec()

add_custom_extractors(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "extractor" = parsnip::extract_fit_engine)
)

synth_spec <- synth_spec()

update_custom_extractors(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "extractor" = parsnip::extract_fit_engine)
)

synth_spec <- synth_spec()

synth_spec <- add_custom_extractors(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "extractor" = parsnip::extract_fit_engine)
)

remove_custom_extractors(synth_spec = synth_spec)
```

---

synth\_spec\_is\_component

*Inspections for synth\_spec components*

---

**Description**

Inspections for synth\_spec components

**Arguments**

z                    Object

**Value**

Boolean if matches class type

---

synth\_spec\_model\_api *Add, update, or remove custom models from a synth\_spec object*

---

### Description

Add, update, or remove custom models from a synth\_spec object

### Usage

```
add_custom_models(synth_spec, ...)  
  
update_custom_models(synth_spec, ...)  
  
remove_custom_models(synth_spec)
```

### Arguments

synth_spec	A synth_spec object
...	Optional named lists with two elements, vars and model, mapping variable names to model_spec objects from library(parsnip).

### Value

A new synth\_spec object.  
A new synth\_spec object with added custom models.  
A new synth\_spec object with updated custom models.  
A new synth\_spec object with removed custom models.

### Examples

```
synth_spec <- synth_spec()  
  
dt_reg_mod <- parsnip::decision_tree() |>  
  parsnip::set_engine("rpart") |>  
  parsnip::set_mode("regression")  
  
add_custom_models(  
  synth_spec = synth_spec,  
  list("vars" = c("a", "b", "c"), "model" = dt_reg_mod)  
)  
  
synth_spec <- synth_spec()  
  
dt_reg_mod <- parsnip::decision_tree() |>  
  parsnip::set_engine("rpart") |>  
  parsnip::set_mode("regression")
```

```
update_custom_models(  
  synth_spec = synth_spec,  
  list("vars" = c("a", "b", "c"), "model" = dt_reg_mod)  
)  
  
synth_spec <- synth_spec()  
  
dt_reg_mod <- parsnip::decision_tree() |>  
  parsnip::set_engine("rpart") |>  
  parsnip::set_mode("regression")  
  
synth_spec <- update_custom_models(  
  synth_spec = synth_spec,  
  list("vars" = c("a", "b", "c"), "model" = dt_reg_mod)  
)  
  
remove_custom_models(synth_spec = synth_spec)
```

---

synth\_spec\_noise\_api *Add, update, or remove noise from a synth\_spec object*

---

## Description

Add, update, or remove noise from a synth\_spec object

## Usage

```
add_custom_noise(synth_spec, ...)
```

## Arguments

synth_spec	A synth_spec object
...	Optional named lists with two elements, vars and noise, mapping variable names to samplers.

## Value

A new synth\_spec object.

A new synth\_spec object with added custom noise.

**Examples**

```
synth_spec <- synth_spec()

noise1 <- noise(
  add_noise = TRUE,
  noise_func = add_noise_kde,
  noise_params = list(
    n_tiles = 2
  )
)

add_custom_noise(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "noise" = noise1)
)
```

---

synth\_spec\_recipes\_api

*Add, update, or remove recipe recipes from a synth\_spec object*

---

**Description**

Add, update, or remove recipe recipes from a synth\_spec object

**Usage**

```
add_custom_steps(synth_spec, ...)

update_custom_steps(synth_spec, ...)

remove_custom_steps(synth_spec)
```

**Arguments**

synth_spec	A synth_spec object
...	Optional named arguments mapping variables to lists of recipe::recipe_function(s) from library(recipes).

**Value**

A new synth\_spec object.

A new synth\_spec object with added custom steps.

A new synth\_spec object with updated custom steps.

A new synth\_spec object with removed custom steps.

**Examples**

```
synth_spec <- synth_spec()

step1 <- function(x) {
  x |> recipes::step_center(recipes::all_predictors(), id = "center")
}

add_custom_steps(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "steps" = step1)
)

synth_spec <- synth_spec()

step1 <- function(x) {
  x |> recipes::step_center(recipes::all_predictors(), id = "center")
}

update_custom_steps(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "steps" = step1)
)

synth_spec <- synth_spec()

step1 <- function(x) {
  x |> recipes::step_center(recipes::all_predictors(), id = "center")
}

synth_spec <- add_custom_steps(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "steps" = step1)
)

remove_custom_steps(synth_spec = synth_spec)
```

---

synth\_spec\_sampler\_api

*Add, update, or remove samplers from a synth\_spec object*

---

**Description**

Add, update, or remove samplers from a synth\_spec object

**Usage**

```
add_custom_samplers(synth_spec, ...)  
  
update_custom_samplers(synth_spec, ...)  
  
remove_custom_samplers(synth_spec)  
  
update_custom_noise(synth_spec, ...)  
  
remove_custom_noise(synth_spec)
```

**Arguments**

synth_spec	A synth_spec object
...	Optional named lists with two elements, vars and sampler, mapping variable names to samplers.

**Value**

A new synth\_spec object.  
A new synth\_spec object with added custom samplers.  
A new synth\_spec object with updated custom samplers.  
A new synth\_spec object with removed custom samplers.  
A new synth\_spec object with updated custom noise.  
A new synth\_spec object with removed custom noise.

**Examples**

```
synth_spec <- synth_spec()  
  
add_custom_samplers(  
  synth_spec = synth_spec,  
  list("vars" = c("a", "b", "c"), "sampler" = sample_rpart)  
)  
  
synth_spec <- synth_spec()  
  
update_custom_samplers(  
  synth_spec = synth_spec,  
  list("vars" = c("a", "b", "c"), "sampler" = sample_rpart)  
)  
  
synth_spec <- synth_spec()  
  
synth_spec <- add_custom_samplers(  
  synth_spec = synth_spec,
```

```
list("vars" = c("a", "b", "c"), "sampler" = sample_rpart)
)

remove_custom_samplers(synth_spec = synth_spec)

synth_spec <- synth_spec()

noise1 <- noise(
  add_noise = TRUE,
  noise_func = add_noise_kde,
  noise_params = list(
    n_ntiles = 2
  )
)

update_custom_noise(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "noise" = noise1)
)

synth_spec <- synth_spec()

noise1 <- noise(
  add_noise = TRUE,
  noise_func = add_noise_kde,
  noise_params = list(
    n_ntiles = 2
  )
)

synth_spec <- add_custom_noise(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "noise" = noise1)
)

remove_custom_noise(synth_spec = synth_spec)
```

---

synth\_spec\_tuner\_api *Add, update, or remove tuners from a synth\_spec object*

---

## Description

Add, update, or remove tuners from a synth\_spec object

## Usage

```
add_custom_tuners(synth_spec, ...)
```

```
update_custom_tuners(synth_spec, ...)
```

```
remove_custom_tuners(synth_spec)
```

### Arguments

synth_spec	A synth_spec object
...	Optional named lists with two elements, vars and tuner, mapping variable names to tuners.

### Value

A new synth\_spec object.  
A new synth\_spec object with added custom tuners.  
A new synth\_spec object with updated custom tuners.  
A new synth\_spec object with removed custom tuners.

### Examples

```
synth_spec <- synth_spec()

tuner1 <- list(
  v = 3,
  grid = 3,
  metrics = yardstick::metric_set(yardstick::rmse)
)

add_custom_tuners(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "tuner" = tuner1)
)

synth_spec <- synth_spec()

tuner1 <- list(
  v = 3,
  grid = 3,
  metrics = yardstick::metric_set(yardstick::rmse)
)

update_custom_tuners(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "tuner" = tuner1)
)

synth_spec <- synth_spec()
```

```

tuner1 <- list(
  v = 3,
  grid = 3,
  metrics = yardstick::metric_set(yardstick::rmse)
)

synth_spec <- add_custom_tuners(
  synth_spec = synth_spec,
  list("vars" = c("a", "b", "c"), "tuner" = tuner1)
)

remove_custom_tuners(synth_spec = synth_spec)

```

---

tune_synthesis	<i>Generate syntheses from multiple presynth objects.</i>
----------------	---

---

### Description

Generate syntheses from multiple presynth objects.

### Usage

```

tune_synthesis(
  presynths,
  postprocessing_func,
  metadata_func = NULL,
  simplify_post = FALSE,
  seed = NULL
)

```

### Arguments

presynths	A list of presynth objects
postprocessing_func	A function with arguments "synth_id", "synth_name", and "postsynth" that performs any desired postprocessing operations, like writing
metadata_func	An optional function with argument "presynth" that extracts specified information from each presynth object and returns a list. Each list element becomes an additional column in the output metadata.
simplify_post	Boolean that, if true, expects postprocessing_func to return a list corresponding to the row of the output dataframe (one per synthesis).
seed	A RNG seed to pass to set.seed()

### Value

A post\_tunesynth object.

**Examples**

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
dt_mod_reg <- parsnip::decision_tree() |>  
  parsnip::set_engine(engine = "rpart") |>  
  parsnip::set_mode(mode = "regression")  
  
dt_mod_reg_cp <- parsnip::decision_tree(cost_complexity = 0.01) |>  
  parsnip::set_engine(engine = "rpart") |>  
  parsnip::set_mode(mode = "regression")  
  
dt_mod_class <- parsnip::decision_tree() |>  
  parsnip::set_engine(engine = "rpart") |>  
  parsnip::set_mode(mode = "classification")  
  
# synth specs  
synth_spec1 <- synth_spec(  
  default_regression_model = dt_mod_reg,  
  default_regression_sampler = sample_rpart,  
  default_classification_model = dt_mod_class,  
  default_classification_sampler = sample_rpart  
)  
  
synth_spec2 <- synth_spec(  
  default_regression_model = dt_mod_reg_cp,  
  default_regression_sampler = sample_rpart,  
  default_classification_model = dt_mod_class,  
  default_classification_sampler = sample_rpart  
)  
  
presynth1 <- presynth(  
  roadmap = rm,  
  synth_spec = synth_spec1  
)  
  
presynth2 <- presynth(  
  roadmap = rm,  
  synth_spec = synth_spec2  
)  
  
postproc_f_null <- function(synth_id, synth_name, postsynth) {  
  return(postsynth[["synthetic_data"]])  
}  
  
tune_synthesis(  
  presynths = list(presynth1, presynth2),  
  postprocessing_func = postproc_f_null,  
  seed = 12345
```

```
)
```

---

update_presynth	<i>Update presynth object</i>
-----------------	-------------------------------

---

### Description

Update presynth object

### Usage

```
update_presynth(presynth, roadmap = NULL, synth_spec = NULL)
```

### Arguments

presynth	A presynth object
roadmap	An optional roadmap object
synth_spec	An optional synth_spec object

### Value

A presynth object.

### Examples

```
# create roadmap
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rpart_mod_reg <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "regression")

rpart_mod_class <- parsnip::decision_tree() |>
  parsnip::set_engine(engine = "rpart") |>
  parsnip::set_mode(mode = "classification")

synth_spec1 <- synth_spec(
  default_regression_model = rpart_mod_reg,
  default_regression_sampler = sample_rpart,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)
```

```

# create a presynth object
# use defaults for noise, constraints, and replicates
presynth <- presynth(
  roadmap = rm,
  synth_spec = synth_spec1
)

lm_mod <- parsnip::linear_reg() |>
  parsnip::set_engine(engine = "lm") |>
  parsnip::set_mode(mode = "regression")

synth_spec2 <- synth_spec(
  default_regression_model = lm_mod,
  default_regression_sampler = sample_lm,
  default_classification_model = rpart_mod_class,
  default_classification_sampler = sample_rpart
)

```

---

update\_synth\_spec      *Tidy API calls* —————

---

## Description

Update non-custom synth\_spec arguments

## Usage

```
update_synth_spec(synth_spec, ...)
```

## Arguments

synth_spec	A synth_spec object
...	Optional named keywords in synth_spec, with the exception of any custom_*arguments

## Value

A synth\_spec

## Examples

```

synth_spec <- synth_spec()

lm_mod <- parsnip::linear_reg() |>
  parsnip::set_engine("lm") |>
  parsnip::set_mode(mode = "regression")

update_synth_spec(
  synth_spec,

```

```

    default_regression_model = lm_mod
  )

```

---

visit_sequence	<i>Generate a visit sequence.</i>
----------------	-----------------------------------

---

### Description

Generate a visit sequence.

### Usage

```
visit_sequence(schema, weight_var = NULL, synthesize_weight = TRUE)
```

### Arguments

**schema**            A schema object.  
**weight\_var**        A numeric weight for the weighted total ordering.  
**synthesize\_weight**  
                     Boolean for if `weight_var` should be included in the visit sequence.

### Value

A `visit_sequence` object.

### Examples

```

df <- data.frame(
  factor_var = c("1", "1", "2"),
  vara = c(10000, 20000, 100000),
  varb = c(300, 200, 100),
  var_loss = c(1999999, 0, -1000000),
  weight = c(1000, 1000, 2000)
)

start_df <- dplyr::select(df, factor_var)

schema1 <- schema(
  conf_data = dplyr::select(df, -weight),
  start_data = start_df
)

vs1 <- visit_sequence(
  schema = schema1
)

schema2 <- schema(
  conf_data = df,

```

```
    start_data = start_df
  )

vs2 <- visit_sequence(
  schema = schema2,
  weight_var = weight,
  synthesize_weight = TRUE
)
```

---

visit\_sequence\_api      *Add or reset a visit\_sequence object within an existing roadmap.*

---

### Description

Add or reset a visit\_sequence object within an existing roadmap.

### Usage

```
update_visit_sequence(roadmap, ...)

reset_visit_sequence(roadmap)
```

### Arguments

roadmap	A roadmap object
...	Optional additional parameters.

### Value

A new roadmap object.  
A roadmap with an updated visit\_sequence.  
A new roadmap object with reset visit\_sequence.

### Examples

```
rm <- roadmap(
  conf_data = acs_conf_nw,
  start_data = acs_start_nw
)

rm |>
  update_visit_sequence(
    weight_var = wgt,
    synthesize_weight = TRUE
  )
```

```
rm <- roadmap(  
  conf_data = acs_conf_nw,  
  start_data = acs_start_nw  
)  
  
rm <- rm |>  
  update_visit_sequence(  
    weight_var = wgt,  
    synthesize_weight = TRUE  
  )  
  
reset_visit_sequence(roadmap = rm)
```

# Index

- \* **datasets**
  - .name\_to\_inspect, 3
  - acs\_conf, 4
  - acs\_conf\_nw, 5
  - acs\_start, 6
  - acs\_start\_nw, 6
  - example\_na, 35
- \* **postprocessing**
  - invert, 36
  - .name\_to\_inspect, 3
- acs\_conf, 4
- acs\_conf\_nw, 5
- acs\_start, 6
- acs\_start\_nw, 6
- add\_constraints (constraints\_api), 19
- add\_custom\_extractors
  - (synth\_spec\_extractor\_api), 70
- add\_custom\_models
  - (synth\_spec\_model\_api), 72
- add\_custom\_noise
  - (synth\_spec\_noise\_api), 73
- add\_custom\_samplers
  - (synth\_spec\_sampler\_api), 75
- add\_custom\_steps
  - (synth\_spec\_recipes\_api), 74
- add\_custom\_tuners
  - (synth\_spec\_tuner\_api), 77
- add\_noise\_cat\_unif, 7
- add\_noise\_disc\_gaussian, 9
- add\_noise\_disc\_laplace, 10
- add\_noise\_gaussian, 11
- add\_noise\_kde, 12
- add\_noise\_laplace, 14
- add\_replicates (replicates\_api), 53
- add\_schema (schema\_api), 61
- add\_sequence\_factor, 15
- add\_sequence\_manual, 16
- add\_sequence\_numeric, 16
- add\_start\_method (start\_method\_api), 64
- collapse\_na, 18
- constraints, 18
- constraints\_api, 19
- construct\_extractors, 21
- construct\_models, 23
- construct\_noise, 24
- construct\_recipes, 26
- construct\_samplers, 28
- construct\_tuners, 30
- convert\_level\_to\_na, 31
- convert\_na\_to\_level, 32
- enforce\_custom\_na, 33
- enforce\_na, 33
- enforce\_schema, 34
- example\_na, 35
- expand\_na, 35
- invert, 36
- invert.step\_BoxCox, 37
- invert.step\_log, 38
- invert.step\_YeoJohnson, 39
- ks\_distance, 40
- ks\_distance\_vec (ks\_distance), 40
- noise, 41
- postsynth\_to\_roadmap, 42
- postsynth\_to\_synth\_spec, 43
- presynth, 44
- print.constraints, 45
- print.noise, 46
- print.postsynth, 47
- print.presynth, 48
- print.replicates, 49
- print.schema, 50
- print.start\_method, 50
- print.synth\_spec, 51
- print.visit\_sequence, 52

remove\_custom\_extractors  
    (synth\_spec\_extractor\_api), 70  
remove\_custom\_models  
    (synth\_spec\_model\_api), 72  
remove\_custom\_noise  
    (synth\_spec\_sampler\_api), 75  
remove\_custom\_samplers  
    (synth\_spec\_sampler\_api), 75  
remove\_custom\_steps  
    (synth\_spec\_recipes\_api), 74  
remove\_custom\_tuners  
    (synth\_spec\_tuner\_api), 77  
remove\_start\_method (start\_method\_api),  
    64  
replicates, 52  
replicates\_api, 53  
reset\_constraints (constraints\_api), 19  
reset\_replicates (replicates\_api), 53  
reset\_schema (schema\_api), 61  
reset\_visit\_sequence  
    (visit\_sequence\_api), 84  
roadmap, 54  
  
sample\_glm, 56  
sample\_lm, 57  
sample\_ranger, 58  
sample\_rpart, 59  
schema, 60  
schema\_api, 61  
start\_method, 63  
start\_method\_api, 64  
start\_resample, 65  
synth\_spec, 67  
synth\_spec\_extractor\_api, 70  
synth\_spec\_is\_component, 71  
synth\_spec\_model\_api, 72  
synth\_spec\_noise\_api, 73  
synth\_spec\_recipes\_api, 74  
synth\_spec\_sampler\_api, 75  
synth\_spec\_tuner\_api, 77  
synthesize, 66  
  
tune\_synthesis, 79  
  
update\_constraints (constraints\_api), 19  
update\_custom\_extractors  
    (synth\_spec\_extractor\_api), 70  
update\_custom\_models  
    (synth\_spec\_model\_api), 72  
update\_custom\_noise  
    (synth\_spec\_sampler\_api), 75  
update\_custom\_samplers  
    (synth\_spec\_sampler\_api), 75  
update\_custom\_steps  
    (synth\_spec\_recipes\_api), 74  
update\_custom\_tuners  
    (synth\_spec\_tuner\_api), 77  
update\_presynth, 81  
update\_replicates (replicates\_api), 53  
update\_schema (schema\_api), 61  
update\_start\_method (start\_method\_api),  
    64  
update\_synth\_spec, 82  
update\_visit\_sequence  
    (visit\_sequence\_api), 84  
  
visit\_sequence, 83  
visit\_sequence\_api, 84