Package ‘recipes’

April 16, 2021

**Title**  Preprocessing Tools to Create Design Matrices

**Version**  0.1.16

**Description**  An extensible framework to create and preprocess design matrices. Recipes consist of one or more data manipulation and analysis `steps`. Statistical parameters for the steps can be estimated from an initial data set and then applied to other data sets. The resulting design matrices can then be used as inputs into statistical or machine learning models.


**BugReports**  https://github.com/tidymodels/recipes/issues

**Depends**  R (>= 3.1), dplyr

**Imports**  ellipsis, generics (>= 0.1.0), glue, gower, ipred, lifecycle, lubridate, magrittr, Matrix, purrr (>= 0.2.3), rlang (>= 0.4.0), stats, tibble, tidyrr (>= 1.0.0), tidyselect (>= 1.1.0), timeDate, utils, withr

**Suggests**  covr, ddaLpha, dimRed (>= 0.2.2), fastICA, ggplot2, igraph, kernlab, knitr, modeldata, RANN, RcppRoll, rmarkdown, rpart, rsample, RSpectra, testthat (>= 2.1.0), xml2

**License**  MIT + file LICENSE

**VignetteBuilder**  knitr

**Encoding**  UTF-8

**RoxygenNote**  7.1.1.9001

**RdMacros**  lifecycle

**NeedsCompilation**  no

**Author**  Max Kuhn [aut, cre], Hadley Wickham [aut], RStudio [cph]

**Maintainer**  Max Kuhn <max@rstudio.com>

**Repository**  CRAN

**Date/Publication**  2021-04-16 15:30:02 UTC
R topics documented:

add_step ................................................................. 4
bake ................................................................. 4
check_class .......................................................... 6
check_cols ........................................................... 8
check_missing ........................................................ 9
check_new_values .................................................... 11
check_range .......................................................... 13
detect_step ............................................................ 15
discretize .............................................................. 16
formula.recipe .......................................................... 18
fully_trained .......................................................... 18
has_role ............................................................... 19
juice ................................................................. 20
names0 ................................................................. 21
prep ................................................................. 22
prepper .............................................................. 24
print.recipe ............................................................ 25
recipe ................................................................. 25
recipes ............................................................... 28
roles ................................................................. 29
selections ............................................................. 31
step_arrange .......................................................... 32
step_bin2factor ........................................................ 34
step_BoxCox .......................................................... 36
step_bs ............................................................... 38
step_center .......................................................... 40
step_classdist ........................................................ 42
step_corr ............................................................. 44
step_count ............................................................ 46
step_cut ............................................................... 48
step_date ............................................................. 50
step_depth ............................................................ 52
step_discretize ....................................................... 54
step_dummy ........................................................... 55
step_factor2string ................................................... 58
step_filter ............................................................ 60
step_geodist .......................................................... 62
step_holiday ............................................................ 64
step_hyperbolic ........................................................ 65
step_ica .............................................................. 67
step_impute_bag ....................................................... 69
step_impute_knn ....................................................... 72
step_impute_linear ................................................... 75
step_impute_lower .................................................... 77
step_impute_mean ..................................................... 79
step_impute_median .................................................. 82
R topics documented:

- step_impute_mode
- step_impute_roll
- step_indicate_na
- step_interact
- step_intercept
- step_inverse
- step_invlogit
- step_isomap
- step_kpca
- step_kpca_poly
- step_kpca_rbf
- step_lag
- step_lincomb
- step_log
- step_logit
- step_mutate
- step_mutate_at
- step_n Naomi
- step_nnmf
- step_normalize
- step_novel
- step_ns
- step_num2factor
- step_nzv
- step_ordinalscore
- step_other
- step_pca
- step_pls
- step_poly
- step_profile
- step_range
- step_ratio
- step_regex
- step_relevel
- step_relu
- step_rename
- step_rename_at
- step_rm
- step_sample
- step_scale
- step_select
- step_shuffle
- step_slice
- step_spatialsign
- step_sqrt
- step_string2factor
- step_unknown
add_step        Add a New Operation to the Current Recipe

Description
add_step adds a step to the last location in the recipe. add_check does the same for checks.

Usage
add_step(rec, object)
add_check(rec, object)

Arguments
rec        A recipe().
object     A step or check object.

Value
A updated recipe() with the new operation in the last slot.

bake        Apply a Trained Data Recipe

Description
For a recipe with at least one preprocessing operation that has been trained by prep.recipe(),
apply the computations to new data.

Usage
bake(object, ...)

## S3 method for class 'recipe'
bake(object, new_data, ..., composition = "tibble")
Arguments

**object**
A trained object such as a `recipe()` with at least one preprocessing operation.

... One or more selector functions to choose which variables will be returned by
the function. See `selections()` for more details. If no selectors are given, the
default is to use `everything()`.

**new_data**
A data frame or tibble for whom the preprocessing will be applied. If `NULL` is
given to `new_data`, the pre-processed *training data* will be returned (assuming
that `prep(retain = TRUE)` was used).

**composition**
Either "tibble", "matrix", "data.frame", or "dgCMatrix" for the format of the
processed data set. Note that all computations during the baking process are
done in a non-sparse format. Also, note that this argument should be called
after any selectors and the selectors should only resolve to numeric columns
(otherwise an error is thrown).

Details

*bake()* takes a trained recipe and applies the operations to a data set to create a design matrix.

If the data set is not too large, time can be saved by using the `retain = TRUE` option of `prep()`.
This stores the processed version of the training set. With this option set, `bake(object,new_data
=NULL)` will return it for free.

Also, any steps with `skip = TRUE` will not be applied to the data when `bake()` is invoked with a
data set in `new_data`. `bake(object,new_data = NULL)` will always have all of the steps applied.

Value

A tibble, matrix, or sparse matrix that may have different columns than the original columns in
new_data.

Author(s)

Max Kuhn

See Also

`recipe()`, `prep()`

Examples

data(ames, package = "modeldata")

ames <- mutate(ames, Sale_Price = log10(Sale_Price))

ames_rec <-
  recipe(Sale_Price ~ ., data = ames[-(1:6), ]) %>%
  step_other(Neighborhood, threshold = 0.05) %>%
  step_dummy(all_nominal()) %>%
  step_interact(~ starts_with("Central_Air") : Year_Built) %>%
  step_ns(Longitude, Latitude, deg_free = 2) %>%
```r
class_rec <- recipe(hm_price ~ ., data = ames)

# build the recipe
class_rec <- step_zv(all_predictors()) %>%
  prep()

# return the training set (already embedded in ames_rec)
ames_train <- bake(class_rec, new_data = NULL)

# apply processing to other data:
ames_new <- bake(class_rec, new_data = head(ames))
```

---

**check_class**

**Check Variable Class**

**Description**

Check class creates a specification of a recipe check that will check if a variable is of a designated class.

**Usage**

```r
check_class(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  class_nm = NULL,
  allow_additional = FALSE,
  skip = FALSE,
  class_list = NULL,
  id = rand_id("class")
)
```

```r
## S3 method for class 'check_class'
tidy(x, ...)
```

**Arguments**

- **recipe**: A recipe object. The check will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the check. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this check since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **class_nm**: A character vector that will be used in `inherits` to check the class. If NULL the classes will be learned in `prep`. Can contain more than one class.
check_class

allow_additional
If TRUE a variable is allowed to have additional classes to the one(s) that are checked.

skip
A logical. Should the check be skipped when the recipe is baked by `bake.recipe()`?
While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

class_list
A named list of column classes. This is NULL until computed by `prep.recipe()`.

id
A character string that is unique to this step to identify it.

x
A check_class object.

Details
This function can check the classes of the variables in two ways. When the class argument is provided it will check if all the variables specified are of the given class. If this argument is NULL, the check will learn the classes of each of the specified variables in prep. Both ways will break bake if the variables are not of the requested class. If a variable has multiple classes in prep, all the classes are checked. Please note that in prep the argument strings_as_factors defaults to TRUE. If the train set contains character variables the check will be break bake when strings_as_factors is TRUE.

Value
An updated version of recipe with the new check added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and value (the type).

See Also
`recipe()` `prep.recipe()` `bake.recipe()`

Examples

```
library(dplyr)
library(modeldata)
data(okc)

# Learn the classes on the train set
train <- okc[1:1000, ]
test <- okc[1001:2000, ]
recipe(train, age ~ . ) %>%
  check_class(everything()) %>%
  prep(train, strings_as_factors = FALSE) %>%
bake(test)

# Manual specification
recipe(train, age ~ . ) %>%
  check_class(age, class_nm = "integer") %>%
  check_class(diet, location, class_nm = "character") %>%
```
check_class(date, class_nm = "Date") %>%
prep(train, strings_as_factors = FALSE) %>%
bake(test)

# By default only the classes that are specified
# are allowed.
x_df <- tibble(time = c(Sys.time() - 60, Sys.time()))
x_df$time %>% class()

## Not run:
recipe(x_df) %>%
  check_class(time, class_nm = "POSIXt") %>%
  prep(x_df) %>%
  bake_(x_df)

## End(Not run)

# Use allow_additional = TRUE if you are fine with it
recipe(x_df) %>%
  check_class(time, class_nm = "POSIXt", allow_additional = TRUE) %>%
  prep(x_df) %>%
  bake(x_df)

---

check_cols  
Check if all Columns are Present

Description

check_cols creates a specification of a recipe step that will check if all the columns of the training frame are present in the new data.

Usage

check_cols(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  skip = FALSE,
  id = rand_id("cols")
)

## S3 method for class 'check_cols'
tidy(x, ...)

Arguments

    recipe  A recipe object. The check will be added to the sequence of operations for this recipe.
check_missing

... One or more selector functions to choose which variables are checked in the check. See selections() for more details. For the tidy method, these are not currently used.

role Not used by this check since no new variables are created.

trained A logical for whether the selectors in ... have been resolved by prep().

skip A logical. Should the check be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A check_cols object.

Details

This check will break the bake function if any of the specified columns is not present in the data. If the check passes, nothing is changed to the data.

Examples

```r
library(modeldata)
data(biomass)

biomass_rec <- recipe(HHV ~ ., data = biomass) %>%
  step_rm(sample, dataset) %>%
  check_cols(contains("gen")) %>%
  step_center(all_numeric_predictors())

## Not run:
bake(biomass_rec, biomass[, c("carbon", "HHV")])

## End(Not run)
```

Table

<table>
<thead>
<tr>
<th>check_missing</th>
<th>Check for Missing Values</th>
</tr>
</thead>
</table>

Description

check_missing creates a specification of a recipe operation that will check if variables contain missing values.
Usage

check_missing(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("missing")
)

## S3 method for class 'check_missing'
tidy(x, ...)

Arguments

recipe A recipe object. The check will be added to the sequence of operations for this recipe.

... One or more selector functions to choose which variables are checked in the check See `selections()` for more details. For the tidy method, these are not currently used.

role Not used by this check since no new variables are created.

trained A logical for whether the selectors in ... have been resolved by `prep()`.

columns A character string of variable names that will be populated (eventually) by the terms argument.

skip A logical. Should the check be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A check_missing object.

Details

This check will break the bake function if any of the checked columns does contain `NA` values. If the check passes, nothing is changed to the data.

Value

An updated version of `recipe` with the new check added to the sequence of existing operations (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected).

Examples

library(modeldata)
data(credit_data)
is.na(credit_data) %>% colSums()

# If the test passes, 'new_data' is returned unaltered
recipe(credit_data) %>%
  check_missing(Age, Expenses) %>%
  prep() %>%
  bake(credit_data)

# If your training set doesn't pass, prep() will stop with an error
## Not run:
recipe(credit_data) %>%
  check_missing(Income) %>%
  prep()
## End(Not run)

# If 'new_data' contain missing values, the check will stop bake()

train_data <- credit_data %>% dplyr::filter(Income > 150)

## Not run:
test_data <- credit_data %>% dplyr::filter(Income <= 150 | is.na(Income))

rp <- recipe(train_data) %>%
  check_missing(Income) %>%
  prep()

bake(rp, train_data)
## Not run:
bake(rp, test_data)
## End(Not run)

---

**check_new_values**  
**Check for New Values**

**Description**

`check_new_values` creates a specification of a recipe operation that will check if variables contain new values.

**Usage**

```r
check_new_values(
  recipe, 
  ..., 
  role = NA, 
  trained = FALSE, 
  columns = NULL, 
  ignore_NA = TRUE,
```
values = NULL,
skip = FALSE,
id = rand_id("new_values")
)

Arguments

recipe  
A recipe object. The check will be added to the sequence of operations for this recipe.

...  
One or more selector functions to choose which variables are checked in the check. See selections() for more details. For the tidy method, these are not currently used.

role  
Not used by this check since no new variables are created.

trained  
A logical for whether the selectors in ... have been resolved by prep().

columns  
A character string of variable names that will be populated (eventually) by the terms argument.

ignore_NA  
A logical that indicates if we should consider missing values as value or not. Defaults to TRUE.

values  
A named list with the allowed values. This is NULL until computed by prep.recipe().

skip  
A logical. Should the check be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id  
A character string that is unique to this step to identify it.

Details

This check will break the bake function if any of the checked columns does contain values it did not contain when prep was called on the recipe. If the check passes, nothing is changed to the data.

Value

An updated version of recipe with the new check added to the sequence of existing operations (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected).

Examples

library(modeldata)
data(credit_data)

# If the test passes, 'new_data' is returned unaltered
recipe(credit_data) %>%
  check_new_values(Home) %>%
  prep() %>%
  bake(new_data = credit_data)

# If 'new_data' contains values not in 'x' at the 'prep()' function,
check_range

# the `bake()` function will break.
## Not run:
recipe(credit_data %>% dplyr::filter(Home != "rent")) %>%
  check_new_values(Home) %>%
  prep() %>%
  bake(new_data = credit_data)
## End(Not run)

# By default missing values are ignored, so this passes.
recipe(credit_data %>% dplyr::filter(!is.na(Home))) %>%
  check_new_values(Home) %>%
  prep() %>%
  bake(credit_data)

# Use `ignore_NA = FALSE` if you consider missing values as a value,
# that should not occur when not observed in the train set.
## Not run:
recipe(credit_data %>% dplyr::filter(!is.na(Home))) %>%
  check_new_values(Home, ignore_NA = FALSE) %>%
  prep() %>%
  bake(credit_data)
## End(Not run)

---

check_range  

Check Range Consistency

**Description**

check_range creates a specification of a recipe check that will check if the range of a numeric variable changed in the new data.

**Usage**

```r
check_range(
  recipe,
  ..., 
  role = NA,
  skip = FALSE,
  trained = FALSE,
  slack_prop = 0.05,
  warn = FALSE,
  lower = NULL,
  upper = NULL,
  id = rand_id("range_check_")
)
```

## S3 method for class 'check_range'

```r
tidy(x, ...)
```
check_range

Arguments

- **recipe**: A recipe object. The check will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the check. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this check since no new variables are created.
- **skip**: A logical. Should the check be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **slack_prop**: The allowed slack as a proportion of the range of the variable in the train set.
- **warn**: If `TRUE` the check will throw a warning instead of an error when failing.
- **lower**: A named numeric vector of minimum values in the train set. This is `NULL` until computed by `prep.recipe()`.
- **upper**: A named numeric vector of maximum values in the train set. This is `NULL` until computed by `prep.recipe()`.
- **id**: A character string that is unique to this step to identify it.
- **x**: A check_range object.

Details

The amount of slack that is allowed is determined by the `slack_prop`. This is a numeric of length one or two. If of length one, the same proportion will be used at both ends of the train set range. If of length two, its first value is used to compute the allowed slack at the lower end, the second to compute the allowed slack at the upper end.

Value

An updated version of `recipe` with the new check added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `value` (the means).

See Also

- `recipe()`
- `prep.recipe()`
- `bake.recipe()`

Examples

```r
slack_df <- data.frame(x = 0:100)
slack_new_data <- data.frame(x = -10:110)
# this will fail the check both ends
## Not run:
```
detect_step

Detect if a particular step or check is used in a recipe

Description

Detect if a particular step or check is used in a recipe

Usage

detect_step(recipe, name)

Arguments

- recipe: A recipe to check.
- name: Character name of a step or check, omitted the prefix. That is, to check if step_intercept is present, use name = intercept.

Value

Logical indicating if recipes contains given step.
Examples

rec <- recipe(Species ~ ., data = iris) %>%
  step_intercept()

detect_step(rec, "step_intercept")

discretize       Discretize Numeric Variables

Description

discretize converts a numeric vector into a factor with bins having approximately the same number of data points (based on a training set).

Usage

discretize(x, ...)

## Default S3 method:
discretize(x, ...)

## S3 method for class 'numeric'
discretize(
  x,
  cuts = 4,
  labels = NULL,
  prefix = "bin",
  keep_na = TRUE,
  infs = TRUE,
  min_unique = 10,
  ...
)

## S3 method for class 'discretize'
predict(object, new_data, ...)

Arguments

x                  A numeric vector

...                Options to pass to stats::quantile() that should not include x or probs.
cuts               An integer defining how many cuts to make of the data.
labels             A character vector defining the factor levels that will be in the new factor (from smallest to largest). This should have length cuts+1 and should not include a level for missing (see keep_na below).
prefix             A single parameter value to be used as a prefix for the factor levels (e.g. bin1, bin2, ...). If the string is not a valid R name, it is coerced to one.
discretize

keep_na A logical for whether a factor level should be created to identify missing values in x.
infs A logical indicating whether the smallest and largest cut point should be infinite.
min_unique An integer defining a sample size line of dignity for the binning. If (the number of unique values)/(cuts+1) is less than min_unique, no discretization takes place.
object An object of class discretize.
new_data A new numeric object to be binned.

Details

discretize estimates the cut points from x using percentiles. For example, if cuts = 3, the function estimates the quartiles of x and uses these as the cut points. If cuts = 2, the bins are defined as being above or below the median of x.

The predict method can then be used to turn numeric vectors into factor vectors.
If keep_na = TRUE, a suffix of "_missing" is used as a factor level (see the examples below).
If infs = FALSE and a new value is greater than the largest value of x, a missing value will result.

Value
discretize returns an object of class discretize and predict.discretize returns a factor vector.

Examples
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

median(biomass_tr$carbon)
discretize(biomass_tr$carbon, cuts = 2)
discretize(biomass_tr$carbon, cuts = 2, infs = FALSE)
discretize(biomass_tr$carbon, cuts = 2, infs = FALSE, keep_na = FALSE)
discretize(biomass_tr$carbon, cuts = 2, prefix = "maybe a bad idea to bin")

carbon_binned <- discretize(biomass_tr$carbon)
table(predict(carbon_binned, biomass_tr$carbon))

carbon_no_infs <- discretize(biomass_tr$carbon, infs = FALSE)
predict(carbon_no_infs, c(50, 100))

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, data = biomass_tr)
rec <- rec %>% step_discretize(carbon, hydrogen)
rec <- prep(rec, biomass_tr)
binned_te <- bake(rec, biomass_te)
table(binned_te$carbon)
Create a Formula from a Prepared Recipe

**Description**

In case a model formula is required, the formula method can be used on a recipe to show what predictors and outcome(s) could be used.

**Usage**

```r
## S3 method for class 'recipe'
formula(x, ...)
```

**Arguments**

- `x`: A recipe object that has been prepared.
- `...`: Note currently used.

**Value**

A formula.

**Examples**

```r
formula(recipe(Species + Sepal.Length ~ ., data = iris) %>% prep())
iris_rec <- recipe(Species ~ ., data = iris) %>%
  step_center(all_numeric()) %>%
  prep()
formula(iris_rec)
```

---

**fully_trained**

Check to see if a recipe is trained/prepared

**Description**

Check to see if a recipe is trained/prepared

**Usage**

```r
fully_trained(x)
```

**Arguments**

- `x`: A recipe
Value
A logical which is true if all of the recipe steps have been run through prep. If no steps have been added to the recipe, TRUE is returned only if the recipe has been prepped.

Examples
```r
rec <- recipe(Species ~ ., data = iris) %>%
  step_center(all_numeric())

rec %>% fully_trained()

rec %>% prep(training = iris) %>% fully_trained()
```

has_role

Description
has_role(), all_predictors(), and all_outcomes() can be used to select variables in a formula that have certain roles.

Similarly, has_type(), all_numeric(), and all_nominal() are used to select columns based on their data type. Nominal variables include both character and factor.

In most cases, the selectors all_numeric_predictors() and all_nominal_predictors(), which select on role and type, will be the right approach for users.

See selections for more details.

current_info() is an internal function.

All of these functions have have limited utility outside of column selection in step functions.

Usage
```r
has_role(match = "predictor")
all_predictors()
all_numeric_predictors()
all_nominal_predictors()
all_outcomes()

has_type(match = "numeric")
all_numeric()
all_nominal()
```
current_info()

Arguments

match  A single character string for the query. Exact matching is used (i.e. regular expressions won’t work).

Value

Selector functions return an integer vector.

current_info() returns an environment with objects vars and data.

Examples

library(modeldata)
data(biomass)

rec <- recipe(biomass) %>%
  update_role(
    carbon, hydrogen, oxygen, nitrogen, sulfur,
    new_role = "predictor"
  ) %>%
  update_role(HHV, new_role = "outcome") %>%
  update_role(sample, new_role = "id variable") %>%
  update_role(dataset, new_role = "splitting indicator")

recipe_info <- summary(rec)
recipe_info

# Centering on all predictors except carbon
rec %>%
  step_center(all_predictors(), -carbon) %>%
  prep(training = biomass) %>%
  bake(new_data = NULL)

juice  Extract Finalized Training Set

Description

As of recipes version 0.1.14, juice() is superseded in favor of bake(object, new_data = NULL).

Usage

juice(object, ..., composition = "tibble")
names0

Arguments

object: A recipe object that has been prepared with the option retain = TRUE.

...: One or more selector functions to choose which variables will be returned by the function. See selections() for more details. If no selectors are given, the default is to use everything().

composition: Either "tibble", "matrix", "data.frame", or "dgCMatrix" for the format of the processed data set. Note that all computations during the baking process are done in a non-sparse format. Also, note that this argument should be called after any selectors and the selectors should only resolve to numeric columns (otherwise an error is thrown).

Details

As steps are estimated by prep, these operations are applied to the training set. Rather than running bake() to duplicate this processing, this function will return variables from the processed training set.

When preparing a recipe, if the training data set is retained using retain = TRUE, there is no need to bake() the recipe to get the preprocessed training set.

juice() will return the results of a recipe where all steps have been applied to the data, irrespective of the value of the step's skip argument.

See Also

recipe() prep.recipe() bake.recipe()

dummy_names(var, lvl, ordinal = FALSE, sep = "_")
Arguments

- **num**: A single integer for how many elements are created.
- **prefix**: A character string that will start each name.
- **var**: A single string for the original factor name.
- **lvl**: A character vectors of the factor levels (in order). When used with `step_dummy()`, `lvl` would be the suffixes that result after `model.matrix` is called (see the example below).
- **ordinal**: A logical; was the original factor ordered?
- **sep**: A single character value for the separator between the names and levels.

Value

`names0` returns a character string of length `num` and `dummy_names` generates a character vector the same length as `lvl`.

Examples

```r
names0(9, "x")
names0(10, "x")

e <- data.frame(y = ordered(letters[1:5]),
                 z = factor(LETTERS[1:5]))

dummy_names("z", levels(e)$z)[-1])
after_mm <- colnames(model.matrix(~y, data = e))[-1]
after_mm
levels(e)$y

dummy_names("y", substring(after_mm, 2), ordinal = TRUE)
```

---

**prep** 

Train a Data Recipe

Description

For a recipe with at least one preprocessing operation, estimate the required parameters from a training set that can be later applied to other data sets.

Usage

```r
prep(x, ...)

## S3 method for class 'recipe'
prep(
  x,
```
prep

training = NULL,
fresh = FALSE,
verbose = FALSE,
retain = TRUE,
log_changes = FALSE,
strings_as_factors = TRUE,
...
)

Arguments

x an object

... further arguments passed to or from other methods (not currently used).

training A data frame or tibble that will be used to estimate parameters for preprocessing.

fresh A logical indicating whether already trained operation should be re-trained. If TRUE, you should pass in a data set to the argument training.

verbose A logical that controls whether progress is reported as operations are executed.

retain A logical: should the preprocessed training set be saved into the template slot of the recipe after training? This is a good idea if you want to add more steps later but want to avoid re-training the existing steps. Also, it is advisable to use retain = TRUE if any steps use the option skip = FALSE. Note that this can make the final recipe size large. When verbose = TRUE, a message is written with the approximate object size in memory but may be an underestimate since it does not take environments into account.

log_changes A logical for printing a summary for each step regarding which (if any) columns were added or removed during training.

strings_as_factors A logical: should character columns be converted to factors? This affects the preprocessed training set (when retain = TRUE) as well as the results of bake.recipe.

Details

Given a data set, this function estimates the required quantities and statistics required by any operations.

prep() returns an updated recipe with the estimates.

Note that missing data handling is handled in the steps; there is no global na.rm option at the recipe-level or in prep().

Also, if a recipe has been trained using prep() and then steps are added, prep() will only update the new operations. If fresh = TRUE, all of the operations will be (re)estimated.

As the steps are executed, the training set is updated. For example, if the first step is to center the data and the second is to scale the data, the step for scaling is given the centered data.

Value

A recipe whose step objects have been updated with the required quantities (e.g. parameter estimates, model objects, etc). Also, the term_info object is likely to be modified as the operations are executed.
Author(s)
Max Kuhn

Examples

data(ames, package = "modeldata")

library(dplyr)

ames <- mutate(ames, Sale_Price = log10(Sale_Price))

ames_rec <-
  recipe(
    Sale_Price ~ Longitude + Latitude + Neighborhood + Year_Built + Central_Air,
    data = ames
  ) %>%
  step_other(Neighborhood, threshold = 0.05) %>%
  step_dummy(all_nominal()) %>%
  step_interact(~ starts_with("Central_Air") : Year_Built) %>%
  step_ns(Longitude, Latitude, deg_free = 5)

prep(ames_rec, verbose = TRUE)

prep(ames_rec, log_changes = TRUE)

---

prepper

Wrapper function for preparing recipes within resampling

Description

When working with the rsample package, a simple recipe must be prepared using the prep function first. When using recipes with rsample it is helpful to have a function that can prepare a recipe across a series of split objects that are produced in this package. prepper is a wrapper function around prep that can be used to do this. See the vignette on "Recipes and rsample" for an example.

Usage

prepper(split_obj, recipe, ...)

Arguments

  split_obj    An rsplit object
  recipe      An untrained recipe object.
  ...          Arguments to pass to prep such as verbose or retain.

Details

prepper() sets the underlying prep() argument fresh to TRUE.
print.recipe  

**Print a Recipe**

**Description**

Print a Recipe

**Usage**

```r
## S3 method for class 'recipe'
print(x, form_width = 30, ...)
```

**Arguments**

- `x`: A recipe object
- `form_width`: The number of characters used to print the variables or terms in a formula
- `...`: Further arguments passed to or from other methods (not currently used).

**Value**

The original object (invisibly)

**Author(s)**

Max Kuhn

---

recipe  

**Create a Recipe for Preprocessing Data**

**Description**

A recipe is a description of what steps should be applied to a data set in order to get it ready for data analysis.

**Usage**

```r
recipe(x, ...)
```

```r
## Default S3 method:
recipe(x, ...)
```

```r
## S3 method for class 'data.frame'
recipe(x, formula = NULL, ..., vars = NULL, roles = NULL)
```

```r
## S3 method for class 'formula'
```
recipe(formula, data, ...)

## S3 method for class 'matrix'
recipe(x, ...)

### Arguments

- **x, data**
  A data frame or tibble of the template data set (see below).

- **...**
  Further arguments passed to or from other methods (not currently used).

- **formula**
  A model formula. No in-line functions should be used here (e.g. \( \log(x) \), \( x:y \), etc.) and minus signs are not allowed. These types of transformations should be enacted using step functions in this package. Dots are allowed as are simple multivariate outcome terms (i.e. no need for `cbind`; see Examples). A model formula may not be the best choice for high-dimensional data with many columns, because of problems with memory.

- **vars**
  A character string of column names corresponding to variables that will be used in any context (see below)

- **roles**
  A character string (the same length of `vars`) that describes a single role that the variable will take. This value could be anything but common roles are "outcome", "predictor", "case_weight", or "ID"

### Details

Recipes are alternative methods for creating design matrices and for preprocessing data.

Variables in recipes can have any type of role in subsequent analyses such as: outcome, predictor, case weights, stratification variables, etc.

*recipe* objects can be created in several ways. If the analysis only contains outcomes and predictors, the simplest way to create one is to use a simple formula (e.g. \( y \sim x1 + x2 \)) that does not contain inline functions such as \( \log(x3) \). An example is given below.

Alternatively, a *recipe* object can be created by first specifying which variables in a data set should be used and then sequentially defining their roles (see the last example). This alternative is an excellent choice when the number of variables is very high, as the formula method is memory-inefficient with many variables.

There are two different types of operations that can be sequentially added to a recipe. **Steps** can include common operations like logging a variable, creating dummy variables or interactions and so on. More computationally complex actions such as dimension reduction or imputation can also be specified. **Checks** are operations that conduct specific tests of the data. When the test is satisfied, the data are returned without issue or modification. Otherwise, any error is thrown.

Once a recipe has been defined, the `prep()` function can be used to estimate quantities required for the operations using a data set (a.k.a. the training data). `prep()` returns another recipe.

To apply the recipe to a data set, the `bake()` function is used in the same manner as `predict` would be for models. This applies the steps to any data set.

Note that the data passed to `recipe` need not be the complete data that will be used to train the steps (by `prep()`). The recipe only needs to know the names and types of data that will be used. For large data sets, `head` could be used to pass the recipe a smaller data set to save time and memory.
Value

An object of class recipe with sub-objects:

- **var_info** A tibble containing information about the original data set columns
- **term_info** A tibble that contains the current set of terms in the data set. This initially defaults to the same data contained in var_info.
- **steps** A list of step or check objects that define the sequence of preprocessing operations that will be applied to data. The default value is NULL
- **template** A tibble of the data. This is initialized to be the same as the data given in the data argument but can be different after the recipe is trained.

Author(s)

Max Kuhn

Examples

```r
# simple example:
library(modeldata)
data(biomass)

# split data
biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

# When only predictors and outcomes, a simplified formula can be used.
rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

# Now add preprocessing steps to the recipe.
sp_signed <- rec %>%
  step_normalize(all_numeric_predictors()) %>%
  step_spatialsign(all_numeric_predictors())
sp_signed

# now estimate required parameters
sp_signed_trained <- prep(sp_signed, training = biomass_tr)
sp_signed_trained

# apply the preprocessing to a data set
test_set_values <- bake(sp_signed_trained, new_data = biomass_te)

# or use pipes for the entire workflow:
rec <- biomass_tr %>%
  recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_spatialsign(all_numeric_predictors())
```
# multivariate example

# no need for cbind(carbon, hydrogen) for left-hand side
multi_y <- recipe(carbon + hydrogen ~ oxygen + nitrogen + sulfur,
                   data = biomass_tr)

multi_y <- multi_y %>%
  step_center(all_numeric_predictors()) %>%
  step_scale(all_numeric_predictors())

multi_y_trained <- prep(multi_y, training = biomass_tr)

results <- bake(multi_y_trained, biomass_te)

# example with manually updating different roles

# best choice for high-dimensional data:
rec <- recipe(biomass_tr) %>%
  update_role(carbon, hydrogen, oxygen, nitrogen, sulfur,
              new_role = "predictor") %>%
  update_role(HHV, new_role = "outcome") %>%
  update_role(sample, new_role = "id variable") %>%
  update_role(dataset, new_role = "splitting indicator")
rec

---

recipes

recipes: A package for computing and preprocessing design matrices.

Description

The recipes package can be used to create design matrices for modeling and to conduct preprocessing of variables. It is meant to be a more extensive framework that R’s formula method. Some differences between simple formula methods and recipes are that

1. Variables can have arbitrary roles in the analysis beyond predictors and outcomes.
2. A recipe consists of one or more steps that define actions on the variables.
3. Recipes can be defined sequentially using pipes as well as being modifiable and extensible.

Basic Functions

The three main functions are recipe(), prep(), and bake().

recipe() defines the operations on the data and the associated roles. Once the preprocessing steps are defined, any parameters are estimated using prep(). Once the data are ready for transformation, the bake() function applies the operations.
Step Functions

These functions are used to add new actions to the recipe and have the naming convention "step_action". For example, `step_center()` centers the data to have a zero mean and `step_dummy()` is used to create dummy variables.

---

**Manually Alter Roles**

<table>
<thead>
<tr>
<th>roles</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_role()</td>
<td>alters an existing role in the recipe or assigns an initial role to variables that do not yet have a declared role.</td>
</tr>
<tr>
<td>add_role()</td>
<td>adds an additional role to variables that already have a role in the recipe. It does not overwrite old roles, as a single variable can have multiple roles.</td>
</tr>
<tr>
<td>remove_role()</td>
<td>eliminates a single existing role in the recipe.</td>
</tr>
</tbody>
</table>

**Usage**

```r
add_role(recipe, ..., new_role = "predictor", new_type = NULL)
update_role(recipe, ..., new_role = "predictor", old_role = NULL)
remove_role(recipe, ..., old_role)
```

**Arguments**

- `recipe`: An existing `recipe()`.
- `...`: One or more selector functions to choose which variables are being assigned a role. See `selections()` for more details.
- `new_role`: A character string for a single role.
- `new_type`: A character string for specific type that the variable should be identified as. If left as NULL, the type is automatically identified as the first type you see for that variable in `summary(recipe)`.
- `old_role`: A character string for the specific role to update for the variables selected by `...`. `update_role()` accepts a NULL as long as the variables have only a single role.

**Details**

Variables can have any arbitrary role (see the examples) but there are two special standard roles, "predictor" and "outcome". These two roles are typically required when fitting a model.

`update_role()` should be used when a variable doesn’t currently have a role in the recipe, or to replace an `old_role` with a `new_role`. `add_role()` only adds additional roles to variables that already have roles and will throw an error when the current role is missing (i.e. NA).
When using add_role(), if a variable is selected that already has the new_role, a warning is emitted and that variable is skipped so no duplicate roles are added.

Adding or updating roles is a useful way to group certain variables that don’t fall in the standard “predictor” bucket. You can perform a step on all of the variables that have a custom role with the selector has_role().

Value

An updated recipe object.

Examples

```r
library(recipes)
library(modeldata)
data(biomass)

# Using the formula method, roles are created for any outcomes and predictors:
recipe(HHV ~ ., data = biomass) %>%
  summary()

# However `sample` and `dataset` aren't predictors. Since they already have
# roles, `update_role()` can be used to make changes, to any arbitrary role:
recipe(HHV ~ ., data = biomass) %>%
  update_role(sample, new_role = "id variable") %>%
  update_role(dataset, new_role = "splitting variable") %>%
  summary()

# `update_role()` cannot set a role to NA, use `remove_role()` for that
## Not run:
recipe(HHV ~ ., data = biomass) %>%
  update_role(sample, new_role = NA_character_)
## End(Not run)

# ------------------------------------------------------------------------------
# Variables can have more than one role. `add_role()` can be used
# if the column already has at least one role:
recipe(HHV ~ ., data = biomass) %>%
  add_role(carbon, sulfur, new_role = "something") %>%
  summary()

# `update_role()` has an argument called `old_role` that is required to
# unambiguously update a role when the column currently has multiple roles.
recipe(HHV ~ ., data = biomass) %>%
  add_role(carbon, new_role = "something") %>%
  update_role(carbon, new_role = "something else", old_role = "something") %>%
  summary()

# `carbon` has two roles at the end, so the last `update_roles()` fails since
# `old_role` was not given.
## Not run:
```
recipe(HHV ~ ., data = biomass) %>%
  add_role(carbon, sulfur, new_role = "something") %>%
  update_role(carbon, new_role = "something else")

## End(Not run)

# To remove a role, `remove_role()` can be used to remove a single role.
recipe(HHV ~ ., data = biomass) %>%
  add_role(carbon, new_role = "something") %>%
  remove_role(carbon, old_role = "something") %>%
  summary()

# To remove all roles, call `remove_role()` multiple times to reset to `NA`
recipe(HHV ~ ., data = biomass) %>%
  add_role(carbon, new_role = "something") %>%
  remove_role(carbon, old_role = "something") %>%
  remove_role(carbon, old_role = "predictor") %>%
  summary()

# If the formula method is not used, all columns have a missing role:
recipe(biomass) %>%
  summary()

---

**selections**

**Methods for Selecting Variables in Step Functions**

**Description**

When selecting variables or model terms in step functions, dplyr-like tools are used. The selector functions can choose variables based on their name, current role, data type, or any combination of these. The selectors are passed as any other argument to the step. If the variables are explicitly stated in the step function, this might be similar to:

```r
recipe(~ ., data = USArrests) %>%
  step_pca(Murder, Assault, UrbanPop, Rape, num_comp = 3)
```

The first four arguments indicate which variables should be used in the PCA while the last argument is a specific argument to `step_pca()`.

Note that:

1. These arguments are not evaluated until the prep function for the step is executed.
2. The dplyr-like syntax allows for negative signs to exclude variables (e.g. `~ -Murder`) and the set of selectors will processed in order.
3. A leading exclusion in these arguments (e.g. ~Murder) has the effect of adding all variables to the list except the excluded variable(s).

Select helpers from the tidyselect package can also be used: tidyselect::starts_with(), tidyselect::ends_with(), tidyselect::contains(), tidyselect::matches(), tidyselect::num_range(), tidyselect::everything(), tidyselect::one_of(), tidyselect::all_of(), and tidyselect::any_of()

For example:

```r
recipe(Species ~ ., data = iris) %>%
  step_center(starts_with("Sepal"), -contains("Width"))
```

would only select Sepal.Length

Columns of the design matrix that may not exist when the step is coded can also be selected. For example, when using step_pca(), the number of columns created by feature extraction may not be known when subsequent steps are defined. In this case, using matches("^PC") will select all of the columns whose names start with "PC" once those columns are created.

There are sets of recipes-specific functions that can be used to select variables based on their role or type: has_role() and has_type(). For convenience, there are also functions that are more specific. The functions all_numeric() and all_nominal() select based on type, with nominal variables including both character and factor; the functions all_predictors() and all_outcomes() select based on role. Any can be used in conjunction with the previous functions described for selecting variables using their names:

```r
data(biomass)
recipe(HHV ~ ., data = biomass) %>%
  step_center(all_numeric(), -all_outcomes())
```

This results in all the numeric predictors: carbon, hydrogen, oxygen, nitrogen, and sulfur.

If a role for a variable has not been defined, it will never be selected using role-specific selectors.

Selectors can be used in step_interact() in similar ways but must be embedded in a model formula (as opposed to a sequence of selectors). For example, the interaction specification could be ~starts_with("Species")::Sepal.Width. This can be useful if Species was converted to dummy variables previously using step_dummy(). The implementation of step_interact() is special, and is more restricted than the other step functions. Only the selector functions from recipes and tidyselect are allowed. User defined selector functions will not be recognized. Additionally, the tidyselect domain specific language is not recognized here, meaning that &, |, !, and ~ will not work.
Usage

```r
step_arrange(  
  recipe,  
  ...,  
  role = NA,  
  trained = FALSE,  
  inputs = NULL,  
  skip = FALSE,  
  id = rand_id("arrange")
)
```

```r
## S3 method for class 'step_arrange'  
tidy(x, ...)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: Comma separated list of unquoted variable names. Use `desc()` to sort a variable in descending order. See `dplyr::arrange()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **inputs**: Quosure of values given by ... .
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_arrange` object

Details

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!!`) to embed the value of the object in the expression (to be portable between sessions). See the examples.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which contains the sorting variable(s) or expression(s). The expressions are text representations and are not parsable.
### Examples

```r
rec <- recipe(~ ., data = iris) %>%
  step_arrange(desc(Sepal.Length), 1/Petal.Length)

prepped <- prep(rec, training = iris %>% slice(1:75))
tidy(prepped, number = 1)

library(dplyr)

dplyr_train <-
  iris %>%
  as_tibble() %>%
  slice(1:75) %>%
  dplyr::arrange(desc(Sepal.Length), 1/Petal.Length)

rec_train <- bake(prepped, new_data = NULL)
all.equal(dplyr_train, rec_train)

dplyr_test <-
  iris %>%
  as_tibble() %>%
  slice(76:150) %>%
  dplyr::arrange(desc(Sepal.Length), 1/Petal.Length)
rec_test <- bake(prepped, iris %>% slice(76:150))
all.equal(dplyr_test, rec_test)

# When you have variables/expressions, you can create a list of symbols with `rlang::syms()` and splice them in the call with `!!!`. See https://tidyeval.tidyverse.org

sort_vars <- c("Sepal.Length", "Petal.Length")

qq_rec <-
  recipe(~ ., data = iris) %>%
  # Embed the `values` object in the call using `!!!`
  step_arrange(!!!syms(sort_vars)) %>%
  prep(training = iris)

tidy(qq_rec, number = 1)
```

### Description

`step_bin2factor` creates a specification of a recipe step that will create a two-level factor from a single dummy variable.
Usage

step_bin2factor(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  levels = c("yes", "no"),
  ref_first = TRUE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("bin2factor")
)

## S3 method for class 'step_bin2factor'
tidy(x, ...)

Arguments

recipe
A recipe object. The step will be added to the sequence of operations for this
recipe.

...  Selector functions that choose which variables will be converted. See selections()
for more details. For the tidy method, these are not currently used.

role  Not used by this step since no new variables are created.

trained  A logical to indicate if the quantities for preprocessing have been estimated.

levels  A length 2 character string that indicates the factor levels for the 1's (in the first
position) and the zeros (second)

ref_first  Logical. Should the first level, which replaces 1's, be the factor reference level?

columns  A vector with the selected variable names. This is NULL until computed by
prep.recipe().

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all
operations are baked when prep.recipe() is run, some operations may not be able to be conducted
on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the
computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_bin2factor object.

Details

This operation may be useful for situations where a binary piece of information may need to be
represented as categorical instead of numeric. For example, naive Bayes models would do better to
have factor predictors so that the binomial distribution is modeled instead of a Gaussian probability
density of numeric binary data. Note that the numeric data is only verified to be numeric (and does
not count levels).
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the columns that will be affected).

Examples

```r
library(modeldata)
data(covers)

rec <- recipe(~ description, covers) %>%
  step_regex(description, pattern = "(rock|stony)", result = "rocks") %>%
  step_regex(description, pattern = "(rock|stony)", result = "more_rocks") %>%
  step_bin2factor(rocks)

 tidy(rec, number = 3)

rec <- prep(rec, training = covers)
results <- bake(rec, new_data = covers)

table(results$rocks, results$more_rocks)

tidy(rec, number = 3)
```

---

**step_BoxCox**

*Box-Cox Transformation for Non-Negative Data*

Description

`step_BoxCox` creates a specification of a recipe step that will transform data using a simple Box-Cox transformation.

Usage

```r
step_BoxCox(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  lambdas = NULL,
  limits = c(-5, 5),
  num_unique = 5,
  skip = FALSE,
  id = rand_id("BoxCox")
)
```

```r
## S3 method for class 'step_BoxCox'
tidy(x, ...)
```
Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **lambdas**: A numeric vector of transformation values. This is NULL until computed by `prep.recipe()`.
- **limits**: A length 2 numeric vector defining the range to compute the transformation parameter lambda.
- **num_unique**: An integer where data that have less possible values will not be evaluated for a transformation.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_BoxCox` object.

Details

The Box-Cox transformation, which requires a strictly positive variable, can be used to rescale a variable to be more similar to a normal distribution. In this package, the partial log-likelihood function is directly optimized within a reasonable set of transformation values (which can be changed by the user).

This transformation is typically done on the outcome variable using the residuals for a statistical model (such as ordinary least squares). Here, a simple null model (intercept only) is used to apply the transformation to the predictor variables individually. This can have the effect of making the variable distributions more symmetric.

If the transformation parameters are estimated to be very close to the bounds, or if the optimization fails, a value of NA is used and no transformation is applied.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `value` (the lambda estimate).

References

See Also

step_YeoJohnson() recipe() prep.recipe() bake.recipe()

Examples

rec <- recipe(~ ., data = as.data.frame(state.x77))
bc_trans <- step_BoxCox(rec, all_numeric())
bc_estimates <- prep(bc_trans, training = as.data.frame(state.x77))
bc_data <- bake(bc_estimates, as.data.frame(state.x77))
plot(density(state.x77[, "Illiteracy"]), main = "before")
plot(density(bc_data$Illiteracy), main = "after")
tidy(bc_trans, number = 1)
tidy(bc_estimates, number = 1)

---

**step_bs**  
*B-Spline Basis Functions*

Description

*step_bs* creates a *specification* of a recipe step that will create new columns that are basis expansions of variables using B-splines.

Usage

```r
step_bs(  
  recipe,  
  ...,  
  role = "predictor",  
  trained = FALSE,  
  deg_free = NULL,  
  degree = 3,  
  objects = NULL,  
  options = list(),  
  skip = FALSE,  
  id = rand_id("bs")  
)
```

## S3 method for class 'step_bs'

tidy(x, ...)
```
step_bs

Arguments

recipe
A recipe object. The step will be added to the sequence of operations for this recipe.

... One or more selector functions to choose which variables are affected by the step. See selections() for more details. For the tidy method, these are not currently used.

role
For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new columns created from the original variables will be used as predictors in a model.

trained
A logical to indicate if the quantities for preprocessing have been estimated.

deg_free
The degrees of freedom for the spline. As the degrees of freedom for a spline increase, more flexible and complex curves can be generated. When a single degree of freedom is used, the result is a rescaled version of the original data.

degree
Degree of polynomial spline (integer).

objects
A list of splines::bs() objects created once the step has been trained.

options
A list of options for splines::bs() which should not include x, degree, or df.

skip
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id
A character string that is unique to this step to identify it.

x
A step_bs object.

Details

step_bs can create new features from a single variable that enable fitting routines to model this variable in a nonlinear manner. The extent of the possible nonlinearity is determined by the df, degree, or knot arguments of splines::bs(). The original variables are removed from the data and new columns are added. The naming convention for the new variables is varname_bs_1 and so on.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the columns that will be affected and holiday.

See Also

step_poly() recipe() step_ns() prep.recipe() bake.recipe()
Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

with_splines <- rec %>%
               step_bs(carbon, hydrogen)
with_splines <- prep(with_splines, training = biomass_tr)

expanded <- bake(with_splines, biomass_te)
expanded
```

---

**step_center**  
*Centering numeric data*

**Description**

`step_center` creates a *specification* of a recipe step that will normalize numeric data to have a mean of zero.

**Usage**

```r
step_center(
  recipe,
  ...,  
  role = NA,
  trained = FALSE,
  means = NULL,
  na_rm = TRUE,
  skip = FALSE,
  id = rand_id("center")
)
```

```r
## S3 method for class 'step_center'
tidy(x, ...)
```

**Arguments**

- `recipe`  
  A recipe object. The step will be added to the sequence of operations for this recipe.

- `...`  
  One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
**step_center**

Not used by this step since no new variables are created.

A logical to indicate if the quantities for preprocessing have been estimated.

A named numeric vector of means. This is NULL until computed by `prep.recipe()`.

A logical value indicating whether NA values should be removed during computations.

A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

A character string that is unique to this step to identify it.

A step_center object.

**Details**

Centering data means that the average of a variable is subtracted from the data. `step_center` estimates the variable means from the data used in the `training` argument of `prep.recipe`. `bake.recipe` then applies the centering to new data sets using these means.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `value` (the means).

**See Also**

`recipe() prep.recipe() bake.recipe()`

**Examples**

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

center_trans <- rec %>%
   step_center(carbon, contains("gen"), -hydrogen)

center_obj <- prep(center_trans, training = biomass_tr)

transformed_te <- bake(center_obj, biomass_te)

biomass_te[1:10, names(transformed_te)]
transformed_te
```
**Description**

*step_classdist* creates a *specification* of a recipe step that will convert numeric data into Mahalanobis distance measurements to the data centroid. This is done for each value of a categorical class variable.

**Usage**

```r
step_classdist(
  recipe,
  ...,
  class,
  role = "predictor",
  trained = FALSE,
  mean_func = mean,
  cov_func = cov,
  pool = FALSE,
  log = TRUE,
  objects = NULL,
  prefix = "classdist_",
  skip = FALSE,
  id = rand_id("classdist")
)
```

### S3 method for class 'step_classdist'

```r
tidy(x, ...)
```

**Arguments**

- **recipe**
  - A recipe object. The step will be added to the sequence of operations for this recipe.
  - One or more selector functions to choose which variables are affected by the step. See *selections()* for more details. For the tidy method, these are not currently used.

- **class**
  - A single character string that specifies a single categorical variable to be used as the class.

- **role**
  - For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that resulting distances will be used as predictors in a model.
trained
mean_func
cov_func
pool
log
objects
prefix
skip
id
x

Details

step_classdist will create a new column for every unique value of the class variable. The resulting variables will not replace the original values and by default have the prefix classdist_. The naming format can be changed using the prefix argument.

Note that, by default, the default covariance function requires that each class should have at least as many rows as variables listed in the terms argument. If pool = TRUE, there must be at least as many data points are variables overall.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected), value (the centroid of the class), and class.

Examples

# in case of missing data...
mean2 <- function(x) mean(x, na.rm = TRUE)

# define naming convention
rec <- recipe(Species ~ ., data = iris) %>%
  step_classdist(all_numeric_predictors(), class = "Species",
               pool = FALSE, mean_func = mean2, prefix = "centroid_")

# default naming
rec <- recipe(Species ~ ., data = iris) %>%
  step_classdist(all_numeric_predictors(), class = "Species",
               pool = FALSE, mean_func = mean2)
#### Description

`step_corr` creates a specification of a recipe step that will potentially remove variables that have large absolute correlations with other variables.

#### Usage

```r
corr <- step_corr(
  recipe, 
  ..., 
  role = NA, 
  trained = FALSE, 
  threshold = 0.9, 
  use = "pairwise.complete.obs", 
  method = "pearson", 
  removals = NULL, 
  skip = FALSE, 
  id = rand_id("corr")
)
```

```r
## S3 method for class 'step_corr'
tidy(x, ...)
```

#### Arguments

- `recipe`: A recipe object. The step will be added to the sequence of operations for this recipe.
- `...`: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the `tidy` method, these are not currently used.
- `role`: Not used by this step since no new variables are created.
- `trained`: A logical to indicate if the quantities for preprocessing have been estimated.
threshold  A value for the threshold of absolute correlation values. The step will try to remove the minimum number of columns so that all the resulting absolute correlations are less than this value.

use  A character string for the use argument to the stats::cor() function.

method  A character string for the method argument to the stats::cor() function.

removals  A character string that contains the names of columns that should be removed. These values are not determined until prep.recipe() is called.

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_corr object.

Details
This step attempts to remove variables to keep the largest absolute correlation between the variables less than threshold. When a column has a single unique value, that column will be excluded from the correlation analysis. Also, if the data set has sporadic missing values (and an inappropriate value of use is chosen), some columns will also be excluded from the filter.

Value
An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the columns that will be removed.

Author(s)
Original R code for filtering algorithm by Dong Li, modified by Max Kuhn. Contributions by Reynald Lescarbeau (for original in caret package). Max Kuhn for the step function.

See Also
step_nzv() recipe() prep.recipe() bake.recipe()

Examples
library(modeldata)
data(biomass)
set.seed(3535)
biomass$duplicate <- biomass$carbon + rnorm(nrow(biomass))

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]
```
rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen +
              sulfur + duplicate,
              data = biomass_tr)

corr_filter <- rec %>%
    step_corr(all_numeric_predictors(), threshold = .5)

filter_obj <- prep(corr_filter, training = biomass_tr)

filtered_te <- bake(filter_obj, biomass_te)
round(abs(cor(biomass_tr[, c(3:7, 9)])), 2)
round(abs(cor(filtered_te)), 2)

tidy(corr_filter, number = 1)
tidy(filter_obj, number = 1)
```

---

**step_count**

Create Counts of Patterns using Regular Expressions

**Description**

`step_count` creates a specification of a recipe step that will create a variable that counts instances of a regular expression pattern in text.

**Usage**

```r
step_count(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  pattern = ".",
  normalize = FALSE,
  options = list(),
  result = make.names(pattern),
  input = NULL,
  skip = FALSE,
  id = rand_id("count")
)
```

## S3 method for class 'step_count'
tidy(x, ...)

**Arguments**

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
A single selector functions to choose which variable will be searched for the pattern. The selector should resolve into a single variable. See `selections()` for more details. For the tidy method, these are not currently used.

For a variable created by this step, what analysis role should they be assigned?

By default, the function assumes that the new dummy variable column created by the original variable will be used as a predictor in a model.

A logical to indicate if the quantities for preprocessing have been estimated.

A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced by `as.character` to a character string if possible.

A logical; should the integer counts be divided by the total number of characters in the string?

A list of options to `gregexpr()` that should not include `x` or `pattern`.

A single character value for the name of the new variable. It should be a valid column name.

A single character value for the name of the variable being searched. This is NULL until computed by `prep.recipe()`.

A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations

A character string that is unique to this step to identify it.

A step_count object.

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `result` (the new column name).

```r
library(modeldata)
data(covers)

rec <- recipe(~ description, covers) %>%
  step_count(description, pattern = "(rock|stony)", result = "rocks") %>%
  step_count(description, pattern = "famil", normalize = TRUE)

rec2 <- prep(rec, training = covers)
rec2

count_values <- bake(rec2, new_data = covers)
count_values

tidy(rec, number = 1)
tidy(rec2, number = 1)
```
step_cut

*Cut a numeric variable into a factor*

**Description**

`step_cut()` creates a *specification* of a recipe step that cuts a numeric variable into a factor based on provided boundary values.

**Usage**

```r
step_cut(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  breaks,
  include_outside_range = FALSE,
  skip = FALSE,
  id = rand_id("cut")
)
```

```r
## S3 method for class 'step_cut'
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `breaks` A numeric vector with at least one cut point.
- `include_outside_range` Logical, indicating if values outside the range in the train set should be included in the lowest or highest bucket. Defaults to FALSE, values outside the original range will be set to NA.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- `id` A character string that is unique to this step to identify it.
- `x` A `step_cut` object.
**Details**

Unlike the `base::cut()` function there is no need to specify the min and the max values in the breaks. All values before the lowest break point will end up in the first bucket, all values after the last break points will end up in the last.

`step_cut()` will call `base::cut()` in the baking step with `include.lowest` set to TRUE.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

**Examples**

```r
df <- data.frame(x = 1:10, y = 5:14)
rec <- recipe(df)

# The min and max of the variable are used as boundaries
# if they exceed the breaks
rec %>%
  step_cut(x, breaks = 5) %>%
  prep() %>%
  bake(df)

# You can use the same breaks on multiple variables
# then for each variable the boundaries are set separately
rec %>%
  step_cut(x, y, breaks = c(6, 9)) %>%
  prep() %>%
  bake(df)

# You can keep the original variables using `step_mutate` or
# `step_mutate_at`, for transforming multiple variables at once
rec %>%
  step_mutate(x_orig = x) %>%
  step_cut(x, breaks = 5) %>%
  prep() %>%
  bake(df)

# It is up to you if you want values outside the
# range learned at prep to be included
new_df <- data.frame(x = 1:11)
rec %>%
  step_cut(x, breaks = 5, include_outside_range = TRUE) %>%
  prep() %>%
  bake(new_df)

rec %>%
  step_cut(x, breaks = 5, include_outside_range = FALSE) %>%
  prep() %>%
  bake(new_df)
```
Description

`step_date` creates a specification of a recipe step that will convert date data into one or more factor or numeric variables.

Usage

```r
step_date(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  features = c("dow", "month", "year"),
  abbr = TRUE,
  label = TRUE,
  ordinal = FALSE,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("date")
)
```

## S3 method for class 'step_date'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class Date or POSIXct. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **features**: A character string that includes at least one of the following values: month, dow (day of week), doy (day of year), week, month, decimal (decimal date, e.g. 2002.197), quarter, semester, year.
- **abbr**: A logical. Only available for features month or dow. FALSE will display the day of the week as an ordered factor of character strings, such as “Sunday”.
step_date

TRUE will display an abbreviated version of the label, such as "Sun". abbr is disregarded if label = FALSE.

label
A logical. Only available for features month or dow. TRUE will display the day of the week as an ordered factor of character strings, such as "Sunday." FALSE will display the day of the week as a number.

ordinal
A logical: should factors be ordered? Only available for features month or dow.

columns
A character string of variables that will be used as inputs. This field is a placeholder and will be populated once prep.recipe() is used.

keep_original_cols
A logical to keep the original variables in the output. Defaults to TRUE.

skip
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id
A character string that is unique to this step to identify it.

x
A step_date object.

Details
Unlike some other steps, step_date does not remove the original date variables by default. Set keep_original_cols to FALSE to remove them.

Value
For step_date, an updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected), value (the feature names), and ordinal (a logical).

See Also
step_holiday() step_rm() recipe() prep.recipe() bake.recipe()

Examples
library(lubridate)

touch: Do you have a list of variables that are dates in your dataset? If you do, you can create a new feature that is the day of the week using step_date. For example:

```
library(lubridate)

# Create a dataset
examples <- data.frame(Dan = ymd("2002-03-04") + days(1:10),
                       Stefan = ymd("2006-01-13") + days(1:10))

date_rec <- recipe(~ Dan + Stefan, examples) %>%
              step_date(all_predictors())

tidy(date_rec, number = 1)

date_rec <- prep(date_rec, training = examples)

date_values <- bake(date_rec, new_data = examples)

date_values
```
**Description**

`step_depth` creates a specification of a recipe step that will convert numeric data into measurement of data depth. This is done for each value of a categorical class variable.

**Usage**

```r
step_depth(
  recipe,
  ..., class,
  role = "predictor",
  trained = FALSE,
  metric = "halfspace",
  options = list(),
  data = NULL,
  prefix = "depth_",
  skip = FALSE,
  id = rand_id("depth")
)
```

```r
## S3 method for class 'step_depth'
tidy(x, ...)
```

**Arguments**

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be used to create the new features. See `selections()` for more details. For the tidy method, these are not currently used.
- **class**: A single character string that specifies a single categorical variable to be used as the class.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that resulting depth estimates will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **metric**: A character string specifying the depth metric. Possible values are "potential", "halfspace", "Mahalanobis", "simplicialVolume", "spatial", and "zonoid".
options  A list of options to pass to the underlying depth functions. See `ddalpha::depth.halfspace()`, `ddalpha::depth.Mahalanobis()`, `ddalpha::depth.potential()`, `ddalpha::depth.projection()`, `ddalpha::depth.simplicial()`, `ddalpha::depth.simplicialVolume()`, `ddalpha::depth.spatial()`, `ddalpha::depth.zonoid()`.

data  The training data are stored here once after `prep.recipe()` is executed.

prefix  A character string that defines the naming convention for new depth columns. Defaults to "depth_". See Details below.

skip  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A `step_depth` object.

Details

Data depth metrics attempt to measure how close data a data point is to the center of its distribution. There are a number of methods for calculating depth but a simple example is the inverse of the distance of a data point to the centroid of the distribution. Generally, small values indicate that a data point not close to the centroid. `step_depth` can compute a class-specific depth for a new data point based on the proximity of the new value to the training set distribution.

This step requires the `ddalpha` package. If not installed, the step will stop with a note about installing the package.

Note that the entire training set is saved to compute future depth values. The saved data have been trained (i.e. prepared) and baked (i.e. processed) up to the point before the location that `step_depth` occupies in the recipe. Also, the data requirements for the different step methods may vary. For example, using `metric = "Mahalanobis"` requires that each class should have at least as many rows as variables listed in the `terms` argument.

The function will create a new column for every unique value of the `class` variable. The resulting variables will not replace the original values and by default have the prefix `depth_`. The naming format can be changed using the `prefix` argument.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `class`.

Examples

```r
# halfspace depth is the default
rec <- recipe(Species ~ ., data = iris) %>%
  step_depth(all_numeric_predictors(), class = "Species")

# use zonoid metric instead
# also, define naming convention for new columns
```
```r
rec <- recipe(Species ~ ., data = iris) %>%
  step_depth(all_numeric_predictors(), class = "Species",
             metric = "zonoid", prefix = "zonoid_")

rec_dists <- prep(rec, training = iris)
dists_to_species <- bake(rec_dists, new_data = iris)
dists_to_species
tidy(rec, number = 1)
tidy(rec_dists, number = 1)
```

### step_discretize

**Discretize Numeric Variables**

**Description**

`step_discretize` creates a specification of a recipe step that will convert numeric data into a factor with bins having approximately the same number of data points (based on a training set).

**Usage**

```r
step_discretize(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  num_breaks = 4,
  min_unique = 10,
  objects = NULL,
  options = list(),
  skip = FALSE,
  id = rand_id("discretize")
)
```

```r
## S3 method for class 'step_discretize'
tidy(x, ...)
```

**Arguments**

- **recipe**
  - A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**
  - For `step_discretize`, the dots specify one or more selector functions to choose which variables are affected by the step. See `selections()` for more details.
  - For the `tidy` method, these are not currently used.
- **role**
  - Not used by this step since no new variables are created.
- **trained**
  - A logical to indicate if the quantities for preprocessing have been estimated.
**step_dummy**

`num_breaks` An integer defining how many cuts to make of the data.

`min_unique` An integer defining a sample size line of dignity for the binning. If (the number of unique values)/(cuts+1) is less than `min_unique`, no discretization takes place.

`objects` The `discretize()` objects are stored here once the recipe has been trained by `prep.recipe()`.

`options` A list of options to `discretize()`. A default is set for the argument `x`. Note that using the options `prefix` and `labels` when more than one variable is being transformed might be problematic as all variables inherit those values.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

`x` A `step_discretize` object

**Value**

`step_discretize` returns an updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables selected) and `value` (the breaks).

**Description**

`step_dummy()` creates a specification of a recipe step that will convert nominal data (e.g. character or factors) into one or more numeric binary model terms for the levels of the original data.

**Usage**

```r
step_dummy(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  one_hot = FALSE,
  preserve = deprecated(),
  naming = dummy_names,
  levels = NULL,
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("dummy")
)```

## S3 method for class 'step_dummy'
tidy(x, ...)

### Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which factor variables will be used to create the dummy variables. See `selections()` for more details. The selected variables must be factors. For the tidy() method, these are not currently used.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the binary dummy variable columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **one_hot**: A logical. For C levels, should C dummy variables be created rather than C-1?
- **preserve**: Use `keep_original_cols` to specify whether the selected column(s) should be retained (in addition to the new dummy variables).
- **naming**: A function that defines the naming convention for new dummy columns. See Details below.
- **levels**: A list that contains the information needed to create dummy variables for each variable contained in terms. This is NULL until the step is trained by `prep.recipe()`.
- **keep_original_cols**: A logical to keep the original variables in the output. Defaults to FALSE.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_dummy` object.

### Details

`step_dummy()` will create a set of binary dummy variables from a factor variable. For example, if an unordered factor column in the data set has levels of "red", "green", "blue", the dummy variable bake will create two additional columns of 0/1 data for two of those three values (and remove the original column). For ordered factors, polynomial contrasts are used to encode the numeric values.

By default, the excluded dummy variable (i.e. the reference cell) will correspond to the first level of the unordered factor being converted.

The function allows for non-standard naming of the resulting variables. For an unordered factor named `x`, with levels "a" and "b", the default naming convention would be to create a new variable called `x_b`. Note that if the factor levels are not valid variable names (e.g. "some text with spaces"), it will be changed by `base::make.names()` to be valid (see the example below). The naming
format can be changed using the naming argument and the function `dummy_names()` is the default. This function will also change the names of ordinal dummy variables. Instead of values such as ".L", ".Q", or "^4", ordinal dummy variables are given simple integer suffixes such as ".1", ".2", etc.

To change the type of contrast being used, change the global contrast option via `options`.

When the factor being converted has a missing value, all of the corresponding dummy variables are also missing. See `step_unknown()` for a solution.

When data to be processed contains novel levels (i.e., not contained in the training set), a missing value is assigned to the results. See `step_other()` for an alternative.

If no columns are selected (perhaps due to an earlier `step_zv()`), `bake()` will return the data as-is (e.g. with no dummy variables).

Note that, by default, the new dummy variable column names obey the naming rules for columns. If there are levels such as "0", `dummy_names()` will put a leading "X" in front of the level (since it uses `make.names()`). This can be changed by passing in a different function to the naming argument for this step.

The package vignette for dummy variables and interactions has more information.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or original variables selected) and `columns` (the list of corresponding binary columns).

**See Also**

`step_factor2string()`, `step_string2factor()`, `dummy_names()`, `step_regex()`, `step_count()`, `step_ordinalscores()`, `step_unorder()`, `step_other()` `step_novel()`

**Examples**

```r
library(modeldata)
data(okc)
okc <- okc[complete.cases(okc),]

# Original data: diet has 18 levels
length(unique(okc$diet))
unique(okc$diet) %>% sort()

rec <- recipe(~ diet + age + height, data = okc)

# Default dummy coding: 17 dummy variables
dummies <- rec %>%
  step_dummy(diet) %>%
  prep(training = okc)

dummy_data <- bake(dummies, new_data = NULL)

dummy_data %>%
  select(starts_with("diet")) %>%
```
names() # level "anything" is the reference level

# Obtain the full set of 18 dummy variables using `one_hot` option
dummies_one_hot <- rec %>%
  step_dummy(diet, one_hot = TRUE) %>%
  prep(training = okc)

dummy_data_one_hot <- bake(dummies_one_hot, new_data = NULL)

dummy_data_one_hot %>%
  select(starts_with("diet")) %>%
  names() # no reference level

tidy(dummies, number = 1)
tidy(dummies_one_hot, number = 1)

---

**step_factor2string**

*Convert Factors to Strings*

**Description**

`step_factor2string` will convert one or more factor vectors to strings.

**Usage**

```r
step_factor2string(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = FALSE,
  skip = FALSE,
  id = rand_id("factor2string")
)
```

## S3 method for class 'step_factor2string'
tidy(x, ...)

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables will be converted to strings. See `selections()` for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
step_factor2string

columns A character string of variables that will be converted. This is NULL until computed by prep.recipe().

skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A step_factor2string object.

Details

prep has an option strings_as_factors that defaults to TRUE. If this step is used with the default option, the string(s) produced by this step will be converted to factors after all of the steps have been prepped.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the columns that will be affected).

See Also

step_string2factor() step_dummy()

Examples

library(modeldata)
data(okc)

rec <- recipe(~ diet + location, data = okc)

rec <- rec %>%
  step_string2factor(diet)

factor_test <- rec %>%
  prep(training = okc, strings_as_factors = FALSE) %>%
  juice
# diet is a factor
class(factor_test$diet)

rec <- rec %>%
  step_factor2string(diet)

string_test <- rec %>%
  prep(training = okc, strings_as_factors = FALSE) %>%
  juice
# diet is a character string
class(string_test$diet)
tidy(rec, number = 1)

**step_filter**

*Filter rows using dplyr*

**Description**

`step_filter` creates a specification of a recipe step that will remove rows using `dplyr::filter()`.

**Usage**

```r
step_filter(
  recipe, 
  ..., 
  role = NA, 
  trained = FALSE, 
  inputs = NULL, 
  skip = TRUE, 
  id = rand_id("filter")
)
```

```r
## S3 method for class 'step_filter'
tidy(x, ...)
```

**Arguments**

- **recipe**
  - A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  - Logical predicates defined in terms of the variables in the data. Multiple conditions are combined with &. Only rows where the condition evaluates to TRUE are kept. See `dplyr::filter()` for more details. For the tidy method, these are not currently used.

- **role**
  - Not used by this step since no new variables are created.

- **trained**
  - A logical to indicate if the quantities for preprocessing have been estimated.

- **inputs**
  - Quosure of values given by ....

- **skip**
  - A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = FALSE`.

- **id**
  - A character string that is unique to this step to identify it.

- **x**
  - A `step_filter` object
Details

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!`) to embed the value of the object in the expression (to be portable between sessions). See the examples.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which contains the conditional statements. These expressions are text representations and are not parsable.

Row Filtering

This step can entirely remove observations (rows of data), which can have unintended and/or problematic consequences when applying the step to new data later via `bake.recipe()`. Consider whether `skip = TRUE` or `skip = FALSE` is more appropriate in any given use case. In most instances that affect the rows of the data being predicted, this step probably should not be applied at all; instead, execute operations like this outside and before starting a preprocessing `recipe()`.

See Also

`step_naomit()` `step_sample()` `step_slice()`

Examples

```r
rec <- recipe(~ ., data = iris) %>%
  step_filter(Sepal.Length > 4.5, Species == "setosa")

prepped <- prep(rec, training = iris %>% slice(1:75))

library(dplyr)

# library(dplyr)
dplyr_train <-
  iris %>%
  as_tibble() %>%
  slice(1:75) %>%
  dplyr::filter(Sepal.Length > 4.5, Species == "setosa")

rec_train <- bake(prepped, new_data = NULL)
all.equal(dplyr_train, rec_train)

# dplyr_test <-
#  iris %>%
#  as_tibble() %>%
#  slice(76:150) %>%
#  dplyr::filter(Sepal.Length > 4.5, Species != "setosa")
rec_test <- bake(prepped, iris %>% slice(76:150))
all.equal(dplyr_test, rec_test)

values <- c("versicolor", "virginica")
```
qq_rec <-
  recipe(~ ., data = iris) %>%
  # Embed the 'values' object in the call using !!
  step_filter(Sepal.Length > 4.5, Species %in% !!values)

 tidy(qq_rec, number = 1)

---

**step_geodist**

*Distance between two locations*

**Description**

*step_geodist* creates a specification of a recipe step that will calculate the distance between points on a map to a reference location.

**Usage**

```r
step_geodist(
  recipe,
  lat = NULL,
  lon = NULL,
  role = "predictor",
  trained = FALSE,
  ref_lat = NULL,
  ref_lon = NULL,
  log = FALSE,
  name = "geo_dist",
  columns = NULL,
  skip = FALSE,
  id = rand_id("geodist")
)
```

```r
## S3 method for class 'step_geodist'
tidy(x, ...)
```

**Arguments**

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **lon, lat**: Selector functions to choose which variables are affected by the step. See selections() for more details.
- **role**: or model term created by this step, what analysis role should be assigned?. By default, the function assumes that resulting distance will be used as a predictor in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **ref_lon, ref_lat**: Single numeric values for the location of the reference point.
**step_geodist**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>log</code></td>
<td>A logical: should the distance be transformed by the natural log function?</td>
</tr>
<tr>
<td><code>name</code></td>
<td>A single character value to use for the new predictor column. If a column exists with this name, an error is issued.</td>
</tr>
<tr>
<td><code>columns</code></td>
<td>A character string of variable names that will be populated (eventually) by the terms argument.</td>
</tr>
<tr>
<td><code>skip</code></td>
<td>A logical. Should the step be skipped when the recipe is baked by <code>bake.recipe()</code>? While all operations are baked when <code>prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = TRUE</code> as it may affect the computations for subsequent operations.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>A character string that is unique to this step to identify it.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>A <code>step_geodist</code> object.</td>
</tr>
<tr>
<td><code>...</code></td>
<td>One or more selector functions to choose which variables are affected by the step. See <code>selections()</code> for more details. For the tidy method, these are not currently used.</td>
</tr>
</tbody>
</table>

**Details**

`step_geodist` will create a

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns echoing the values of `lat`, `lon`, `ref_lat`, `ref_lon`, `name`, and `id`.

**Examples**

```r
library(modeldata)
data(Smithsonian)

# How close are the museums to Union Station?
near_station <- recipe(~ ., data = Smithsonian) %>%
  update_role(name, new_role = "location") %>%
  step_geodist(lat = latitude, lon = longitude, log = FALSE,
               ref_lat = 38.8986312, ref_lon = -77.0062457) %>%
  prep(training = Smithsonian)

bake(near_station, new_data = NULL) %>%
  arrange(geo_dist)

tidy(near_station, number = 1)
```
**Description**

`step_holiday` creates a specification of a recipe step that will convert date data into one or more binary indicator variables for common holidays.

**Usage**

```r
step_holiday(
  recipe,
  ...,  # One or more selector functions to choose which variables will be used to create the new variables. The selected variables should have class Date or POSIXct. See selections() for more details. For the tidy method, these are not currently used.
  role = "predictor",
  trained = FALSE,
  holidays = c("LaborDay", "NewYearsDay", "ChristmasDay"),  # A character string that includes at least one holiday supported by the timeDate package. See timeDate::listHolidays() for a complete list.
  columns = NULL,  # A character string of variables that will be used as inputs. This field is a placeholder and will be populated once prep.recipe() is used.
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("holiday")
)
```

```r
## S3 method for class 'step_holiday'
 tidy(x, ...)  # tidy method for class 'step_holiday'
```

**Arguments**

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables will be used to create the new variables. The selected variables should have class Date or POSIXct. See selections() for more details. For the tidy method, these are not currently used.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **holidays**: A character string that includes at least one holiday supported by the timeDate package. See timeDate::listHolidays() for a complete list.
- **columns**: A character string of variables that will be used as inputs. This field is a placeholder and will be populated once prep.recipe() is used.
- **keep_original_cols**: A logical to keep the original variables in the output. Defaults to TRUE.
## step_hyperbolic

### Description

*step_hyperbolic* creates a *specification* of a recipe step that will transform data using a hyperbolic function.

### Details

- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

  While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_holiday` object.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the columns that will be affected and `holiday`.

### See Also

- `step_date()`, `step_rm()`, `recipe()`, `prep.recipe()`, `bake.recipe()`, `timeDate::listHolidays()`

### Examples

```r
library(lubridate)

examples <- data.frame(someday = ymd("2000-12-20") + days(0:40))
holiday_rec <- recipe(~ someday, examples) %>%
  step_holiday(all_predictors())
holiday_rec <- prep(holiday_rec, training = examples)
holiday_values <- bake(holiday_rec, new_data = examples)
holiday_values
```

---

**step_hyperbolic**  

*Hyperbolic Transformations*

---

**Description**

- **step_hyperbolic** creates a *specification* of a recipe step that will transform data using a hyperbolic function.
step_hyperbolic

Usage

step_hyperbolic(
  recipe,
  ...,  
  role = NA,
  trained = FALSE,
  func = "sin",
  inverse = TRUE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("hyperbolic")
)

## S3 method for class 'step_hyperbolic'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See selections() for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **func**: A character value for the function. Valid values are "sin", "cos", or "tan".
- **inverse**: A logical: should the inverse function be used?
- **columns**: A character string of variable names that will be populated (eventually) by the terms argument.
- **skip**: A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A step_hyperbolic object.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the columns that will be affected), inverse, and func.

See Also

step_logit() step_invlogit() step_log() step_sqrt() recipe() prep.recipe() bake.recipe()
**Examples**

```r
set.seed(313)
examples <- matrix(rnorm(40), ncol = 2)
examples <- as.data.frame(examples)

rec <- recipe(~ V1 + V2, data = examples)

cos_trans <- rec %>%
  step_hyperbolic(all_numeric_predictors(),
                  func = "cos", inverse = FALSE)

cos_obj <- prep(cos_trans, training = examples)

transformed_te <- bake(cos_obj, examples)
plot(examples$V1, transformed_te$V1)
tidy(cos_trans, number = 1)
tidy(cos_obj, number = 1)
```

---

**step_ica**

*ICA Signal Extraction*

**Description**

*step_ica* creates a *specification* of a recipe step that will convert numeric data into one or more independent components.

**Usage**

```r
step_ica(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  options = list(method = "C"),
  res = NULL,
  prefix = "IC",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("ica")
)
```

```r
## S3 method for class 'step_ica'
tidy(x, ...)
```
Arguments

recipe  A recipe object. The step will be added to the sequence of operations for this recipe.

role  For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new independent component columns created by the original variables will be used as predictors in a model.

num_comp  The number of ICA components to retain as new predictors. If num_comp is greater than the number of columns or the number of possible components, a smaller value will be used.

options  A list of options to fastICA::fastICA(). No defaults are set here. Note that the arguments X and n.comp should not be passed here.

prefix  A character string that will be the prefix to the resulting new variables. See notes below.

keep_original_cols  A logical to keep the original variables in the output. Defaults to FALSE.

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_ica object.

Details

Independent component analysis (ICA) is a transformation of a group of variables that produces a new set of artificial features or components. ICA assumes that the variables are mixtures of a set of distinct, non-Gaussian signals and attempts to transform the data to isolate these signals. Like PCA, the components are statistically independent from one another. This means that they can be used to combat large inter-variables correlations in a data set. Also like PCA, it is advisable to center and scale the variables prior to running ICA.

This package produces components using the "FastICA" methodology (see reference below). This step requires the dimRed and fastICA packages. If not installed, the step will stop with a note about installing these packages.

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if num_comp < 10, their names will be IC1 - IC9. If num_comp = 101, the names would be IC001 - IC101.
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected), value (the loading), and component.

References


See Also

step_pca() step_kpca() step_isomap() recipe() prep.recipe() bake.recipe()

Examples

# from fastICA::fastICA
set.seed(131)
S <- matrix(runif(400), 200, 2)
A <- matrix(c(1, 1, -1, 3), 2, 2, byrow = TRUE)
X <- as.data.frame(S %*% A)

tr <- X[1:100,]
te <- X[101:200,]

rec <- recipe(~ ., data = tr)

ica_trans <- step_center(rec, V1, V2)
ica_trans <- step_scale(ica_trans, V1, V2)
ica_trans <- step_ica(ica_trans, V1, V2, num_comp = 2)

if (require(dimRed) & require(fastICA)) {
  ica_estimates <- prep(ica_trans, training = tr)
  ica_data <- bake(ica_estimates, te)

  plot(te$V1, te$V2)
  plot(ica_data$IC1, ica_data$IC2)

  tidy(ica_trans, number = 3)
  tidy(ica_estimates, number = 3)
}

---

**step_impute_bag**  

**Imputation via Bagged Trees**

**Description**

**step_impute_bag** creates a specification of a recipe step that will create bagged tree models to impute missing data.
step_impute_bag

Usage

step_impute_bag(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  impute_with = imp_vars(all_predictors()),
  trees = 25,
  models = NULL,
  options = list(keepX = FALSE),
  seed_val = sample.int(10^4, 1),
  skip = FALSE,
  id = rand_id("impute_bag")
)

step_bagimpute(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  impute_with = imp_vars(all_predictors()),
  trees = 25,
  models = NULL,
  options = list(keepX = FALSE),
  seed_val = sample.int(10^4, 1),
  skip = FALSE,
  id = rand_id("impute_bag")
)

imp_vars(...)

## S3 method for class 'step_impute_bag'
tidy(x, ...)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this recipe.

... One or more selector functions to choose variables. For step_impute_bag, this indicates the variables to be imputed. When used with imp_vars, the dots indicate which variables are used to predict the missing data in each variable. See selections() for more details. For the tidy method, these are not currently used.

role Not used by this step since no new variables are created.

trained A logical to indicate if the quantities for preprocessing have been estimated.

impute_with A call to imp_vars to specify which variables are used to impute the variables that can include specific variable names separated by commas or different selec-
step_impute_bag

tors (see selections()). If a column is included in both lists to be imputed and to be an imputation predictor, it will be removed from the latter and not used to impute itself.

trees
An integer for the number of bagged trees to use in each model.

models
The ipred::ipredbagg() objects are stored here once this bagged trees have be trained by prep.recipe().

options
A list of options to ipred::ipredbagg(). Defaults are set for the arguments nbagg and keepX but others can be passed in. Note that the arguments X and y should not be passed here.

seed_val
An integer used to create reproducible models. The same seed is used across all imputation models.

skip
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id
A character string that is unique to this step to identify it.

x
A step_impute_bag object.

Details
For each variable requiring imputation, a bagged tree is created where the outcome is the variable of interest and the predictors are any other variables listed in the impute_with formula. One advantage to the bagged tree is that is can accept predictors that have missing values themselves. This imputation method can be used when the variable of interest (and predictors) are numeric or categorical. Imputed categorical variables will remain categorical. Also, integers will be imputed to integer too.

Note that if a variable that is to be imputed is also in impute_with, this variable will be ignored.

It is possible that missing values will still occur after imputation if a large majority (or all) of the imputing variables are also missing.

As of recipes 0.1.16, this function name changed from step_bagimpute() to step_impute_bag().

Value
An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and model (the bagged tree object).

References

Examples
library(modeldata)
data("credit_data")
## missing data per column
vapply(credit_data, function(x) mean(is.na(x)), c(num = 0))

set.seed(342)
in_training <- sample(1:nrow(credit_data), 2000)

credit_tr <- credit_data[ in_training, ]
credit_te <- credit_data[-in_training, ]
missing_examples <- c(14, 394, 565)

rec <- recipe(Price ~ ., data = credit_tr)
# Not run:
impute_rec <- rec %>%
  step_impute_bag(Status, Home, Marital, Job, Income, Assets, Debt)

imp_models <- prep(impute_rec, training = credit_tr)

imputed_te <- bake(imp_models, new_data = credit_te, everything())

credit_te[missing_examples,]
imputed_te[missing_examples, names(credit_te)]
tidy(impute_rec, number = 1)
tidy(imp_models, number = 1)

## Specifying which variables to impute with
impute_rec <- rec %>%
  step_impute_bag(Status, Home, Marital, Job, Income, Assets, Debt,
  impute_with = imp_vars(Time, Age, Expenses),
  # for quick execution, nbagg lowered
  options = list(nbagg = 5, keepX = FALSE))

imp_models <- prep(impute_rec, training = credit_tr)

imputed_te <- bake(imp_models, new_data = credit_te, everything())

credit_te[missing_examples,]
imputed_te[missing_examples, names(credit_te)]
tidy(impute_rec, number = 1)
tidy(imp_models, number = 1)

## End(Not run)

---

**step_impute_knn**

Imputation via K-Nearest Neighbors
step_impute_knn

Description

step_impute_knn creates a specification of a recipe step that will impute missing data using nearest neighbors.

Usage

step_impute_knn(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  neighbors = 5,
  impute_with = imp_vars(all_predictors()),
  options = list(nthread = 1, eps = 1e-08),
  ref_data = NULL,
  columns = NULL,
  skip = FALSE,
  id = rand_id("impute_knn")
)

step_knnimpute(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  neighbors = 5,
  impute_with = imp_vars(all_predictors()),
  options = list(nthread = 1, eps = 1e-08),
  ref_data = NULL,
  columns = NULL,
  skip = FALSE,
  id = rand_id("impute_knn")
)

## S3 method for class 'step_impute_knn'
tidy(x, ...)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this recipe.
...
One or more selector functions to choose variables. For step_impute_knn, this indicates the variables to be imputed. When used with imp_vars, the dots indicate which variables are used to predict the missing data in each variable. See selections() for more details. For the tidy method, these are not currently used.
role Not used by this step since no new variables are created.
trained A logical to indicate if the quantities for preprocessing have been estimated.
neighbors  The number of neighbors.

impute_with  A call to imp.vars to specify which variables are used to impute the variables that can include specific variable names separated by commas or different selectors (see selections()). If a column is included in both lists to be imputed and to be an imputation predictor, it will be removed from the latter and not used to impute itself.

options  A named list of options to pass to gower::gower_topn(). Available options are currently nthread and eps.

ref_data  A tibble of data that will reflect the data preprocessing done up to the point of this imputation step. This is NULL until the step is trained by prep.recipe().

columns  The column names that will be imputed and used for imputation. This is NULL until the step is trained by prep.recipe().

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_impute_knn object.

Details

The step uses the training set to impute any other data sets. The only distance function available is Gower’s distance which can be used for mixtures of nominal and numeric data.

Once the nearest neighbors are determined, the mode is used to predict nominal variables and the mean is used for numeric data. Note that, if the underlying data are integer, the mean will be converted to an integer too.

Note that if a variable that is to be imputed is also in impute_with, this variable will be ignored.

It is possible that missing values will still occur after imputation if a large majority (or all) of the imputing variables are also missing.

As of recipes 0.1.16, this function name changed from step_knnimpute() to step_impute_knn().

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables for imputation), predictors (those variables used to impute), and neighbors.

References

Examples

library(recipes)
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training", ]
biomass_te <- biomass[biomass$dataset == "Testing", ]
biomass_te_whole <- biomass_te

# induce some missing data at random
set.seed(9039)
carb_missing <- sample(1:nrow(biomass_te), 3)
nitro_missing <- sample(1:nrow(biomass_te), 3)

biomass_te$carbon[carb_missing] <- NA
biomass_te$nitrogen[nitro_missing] <- NA

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

ratio_recipe <- rec %>%
               step_impute_knn(all_predictors(), neighbors = 3)
ratio_recipe2 <- prep(ratio_recipe, training = biomass_tr)
imputed <- bake(ratio_recipe2, biomass_te)

# how well did it work?
summary(biomass_te_whole$carbon)
cbind(before = biomass_te_whole$carbon[carb_missing],
       after = imputed$carbon[carb_missing])

summary(biomass_te_whole$nitrogen)
cbind(before = biomass_te_whole$nitrogen[nitro_missing],
       after = imputed$nitrogen[nitro_missing])

tidy(ratio_recipe, number = 1)
tidy(ratio_recipe2, number = 1)

---

step_impute_linear  Imputation of numeric variables via a linear model.

Description

step_impute_linear creates a specification of a recipe step that will create linear regression models to impute missing data.

Usage

step_impute_linear(
  recipe,
   ...)
...,
role = NA,
trained = FALSE,
impute_with = imp_vars(all_predictors()),
models = NULL,
skip = FALSE,
id = rand_id("impute_linear")
)

## S3 method for class 'step_impute_linear'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose variables. For step_impute_linear, this indicates the variables to be imputed; these variables must be of type numeric. When used with imp_vars, the dots indicates which variables are used to predict the missing data in each variable. See selections() for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **impute_with**: A call to imp_vars to specify which variables are used to impute the variables that can include specific variable names separated by commas or different selectors (see selections()). If a column is included in both lists to be imputed and to be an imputation predictor, it will be removed from the latter and not used to impute itself.
- **models**: The lm() objects are stored here once the linear models have been trained by prep.recipe().
- **skip**: A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A step_impute_linear object.

Details

For each variable requiring imputation, a linear model is fit where the outcome is the variable of interest and the predictors are any other variables listed in the impute_with formula. Note that if a variable that is to be imputed is also in impute_with, this variable will be ignored.

The variable(s) to be imputed must be of type numeric. The imputed values will keep the same type as their original data (i.e, model predictions are coerced to integer as needed).

Since this is a linear regression, the imputation model only uses complete cases for the training set predictors.
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and model (the bagged tree object).

References


Examples

```r
data(ames, package = "modeldata")
set.seed(393)
ames_missing <- ames
ames_missing$Longitude[sample(1:nrow(ames), 200)] <- NA

imputed_ames <-
  recipe(Sale_Price ~ ., data = ames_missing) %>%
  step_impute_linear(
    Longitude,
    impute_with = imp_vars(Latitude, Neighborhood, MS_Zoning, Alley)
  ) %>%
  prep(ames_missing)

imputed <-
  bake(imputed_ames, new_data = ames_missing) %>%
  dplyr::rename(imputed = Longitude) %>%
  bind_cols(ames %>% dplyr::select(original = Longitude)) %>%
  bind_cols(ames_missing %>% dplyr::select(Longitude)) %>%
  dplyr::filter(is.na(Longitude))

library(ggplot2)
ggplot(imputed, aes(x = original, y = imputed)) +
  geom_abline(col = "green") +
  geom_point(alpha = .3) +
  coord_equal() +
  labs(title = "Imputed Values")
```

---

**step_impute_lower**  
Impute Numeric Data Below the Threshold of Measurement

Description

step_impute_lower creates a specification of a recipe step designed for cases where the non-negative numeric data cannot be measured below a known value. In these cases, one method for imputing the data is to substitute the truncated value by a random uniform number between zero and the truncation point.
Usage

```r
step_impute_lower(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  threshold = NULL,
  skip = FALSE,
  id = rand_id("impute_lower")
)
```

```
step_lowerimpute( 
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  threshold = NULL,
  skip = FALSE,
  id = rand_id("impute_lower")
)
```

```r
## S3 method for class 'step_impute_lower'
 tidy(x, ...)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **threshold**: A named numeric vector of lower bounds. This is `NULL` until computed by `prep.recipe()`.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
  While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_impute_lower` object.

Details

`step_impute_lower` estimates the variable minimums from the data used in the training argument of `prep.recipe`. `bake.recipe` then simulates a value for any data at the minimum with a
random uniform value between zero and the minimum.

As of recipes 0.1.16, this function name changed from step_lowerimpute() to step_impute_lower().

**Value**

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and value for the estimated threshold.

**Examples**

```r
library(recipes)
library(modeldata)
data(biomass)

## Truncate some values to emulate what a lower limit of
## the measurement system might look like
biomass$carbon <- ifelse(biomass$carbon > 40, biomass$carbon, 40)
biomass$hydrogen <- ifelse(biomass$hydrogen > 5, biomass$carbon, 5)

biomass_tr <- biomass[biomass$dataset == "Training",]
bioas_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

impute_rec <- rec %>%
  step_impute_lower(carbon, hydrogen)

tidy(impute_rec, number = 1)

impute_rec <- prep(impute_rec, training = biomass_tr)

tidy(impute_rec, number = 1)

transformed_te <- bake(impute_rec, biomass_te)

plot(transformed_te$carbon, biomass_te$carbon,
      ylab = "pre-imputation", xlab = "imputed")
```

---

**Description**

`step_impute_mean` creates a specification of a recipe step that will substitute missing values of numeric variables by the training set mean of those variables.
**Usage**

```r
step_impute_mean(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  means = NULL,
  trim = 0,
  skip = FALSE,
  id = rand_id("impute_mean")
)
```

```r
step_meanimpute(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  means = NULL,
  trim = 0,
  skip = FALSE,
  id = rand_id("impute_mean")
)
```

## S3 method for class 'step_impute_mean'

tidy(x, ...)

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `means` A named numeric vector of means. This is NULL until computed by `prep.recipe()`. Note that, if the original data are integers, the mean will be converted to an integer to maintain the same data type.
- `trim` The fraction (0 to 0.5) of observations to be trimmed from each end of the variables before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
**step_impute_mean**

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A step_impute_mean object.

**Details**

step_impute_mean estimates the variable means from the data used in the training argument of prep.recipe. bake.recipe then applies the new values to new data sets using these averages.

As of recipes 0.1.16, this function name changed from step_meanimpute() to step_impute_mean().

**Value**

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and model (the mean value).

**Examples**

```r
library(modeldata)
data("credit_data")

## missing data per column
vapply(credit_data, function(x) mean(is.na(x)), c(num = 0))

set.seed(342)
in_training <- sample(1:nrow(credit_data), 2000)
credit_tr <- credit_data[in_training,]
credit_te <- credit_data[-in_training,]
missing_examples <- c(14, 394, 565)
rec <- recipe(Price ~ ., data = credit_tr)
impute_rec <- rec %>%
  step_impute_mean(Income, Assets, Debt)
imp_models <- prep(impute_rec, training = credit_tr)
imputed_te <- bake(imp_models, new_data = credit_te, everything())

credit_te[missing_examples,]
imputed_te[missing_examples, names(credit_te)]

tidy(impute_rec, number = 1)
tidy(imp_models, number = 1)
```
Impute Numeric Data Using the Median

Description

`step_impute_median` creates a specification of a recipe step that will substitute missing values of numeric variables by the training set median of those variables.

Usage

```r
step_impute_median(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  medians = NULL,
  skip = FALSE,
  id = rand_id("impute_median")
)

step_medianimpute(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  medians = NULL,
  skip = FALSE,
  id = rand_id("impute_median")
)
```

## S3 method for class 'step_impute_median'

tidy(x, ...)

Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **medians**
  A named numeric vector of medians. This is `NULL` until computed by `prep.recipe()`. Note that, if the original data are integers, the median will be converted to an integer to maintain the same data type.
**step_impute_median**

**skip**
A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

**id**
A character string that is unique to this step to identify it.

**x**
A `step_impute_median` object.

**Details**

`step_impute_median` estimates the variable medians from the data used in the `training` argument of `prep.recipe`. `bake.recipe` then applies the new values to new data sets using these medians.

As of recipes 0.1.16, this function name changed from `step_medianimpute()` to `step_impute_median()`.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables selected) and `model` (the median value).

**Examples**

```r
library(modeldata)
data("credit_data")

## missing data per column
vapply(credit_data, function(x) mean(is.na(x)), c(num = 0))

set.seed(342)
in_training <- sample(1:nrow(credit_data), 2000)
credit_tr <- credit_data[in_training, ]
credit_te <- credit_data[-in_training, ]
missing_examples <- c(14, 394, 565)
rec <- recipe(Price ~ ., data = credit_tr)
impute_rec <- rec %>%
  step_impute_median(Income, Assets, Debt)
imp_models <- prep(impute_rec, training = credit_tr)
imputed_te <- bake(imp_models, new_data = credit_te, everything())
credit_te[missing_examples,]
imputed_te[missing_examples,]
tidy(impute_rec, number = 1)
tidy(imp_models, number = 1)
```
**step_impute_mode**  
*Impute Nominal Data Using the Most Common Value*

**Description**

`step_impute_mode` creates a specification of a recipe step that will substitute missing values of nominal variables by the training set mode of those variables.

**Usage**

```r
step_impute_mode(
  recipe, 
  ..., 
  role = NA, 
  trained = FALSE, 
  modes = NULL, 
  skip = FALSE, 
  id = rand_id("impute_mode")
)
```

```r
step_modeimpute(
  recipe, 
  ..., 
  role = NA, 
  trained = FALSE, 
  modes = NULL, 
  skip = FALSE, 
  id = rand_id("impute_mode")
)
```

```r
# S3 method for class 'step_impute_mode'
tidy(x, ...)
```

**Arguments**

- `recipe`: A recipe object. The step will be added to the sequence of operations for this recipe.
- `...`: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- `role`: Not used by this step since no new variables are created.
- `trained`: A logical to indicate if the quantities for preprocessing have been estimated.
- `modes`: A named character vector of modes. This is `NULL` until computed by `prep.recipe()`.
- `skip`: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations
may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

**id**
A character string that is unique to this step to identify it.

**x**
A `step_impute_mode` object.

**Details**

`step_impute_mode` estimates the variable modes from the data used in the training argument of `prep.recipe`. `bake.recipe` then applies the new values to new data sets using these values. If the training set data has more than one mode, one is selected at random.

As of recipes 0.1.16, this function name changed from `step_modeimpute()` to `step_impute_mode()`.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables selected) and `model` (the mode value).

**Examples**

```r
library(modeldata)
data("credit_data")

## missing data per column
vapply(credit_data, function(x) mean(is.na(x)), c(num = 0))

set.seed(342)
in_training <- sample(1:nrow(credit_data), 2000)

credit_tr <- credit_data[ in_training, ]
credit_te <- credit_data[-in_training, ]
missing_examples <- c(14, 394, 565)

rec <- recipe(Price ~ ., data = credit_tr)
impute_rec <- rec %>%
  step_impute_mode(Status, Home, Marital)

imp_models <- prep(impute_rec, training = credit_tr)
imputed_te <- bake(imp_models, new_data = credit_te, everything())
table(credit_te$Home, imputed_te$Home, useNA = "always")
tidy(impute_rec, number = 1)
tidy(imp_models, number = 1)
```
Description

step_impute_roll creates a specification of a recipe step that will substitute missing values of numeric variables by the measure of location (e.g. median) within a moving window.

Usage

```r
step_impute_roll(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  statistic = median,
  window = 5,
  skip = FALSE,
  id = rand_id("impute_roll")
)
```

```r
step_rollimpute(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  statistic = median,
  window = 5,
  skip = FALSE,
  id = rand_id("impute_roll")
)
```

## S3 method for class 'step_impute_roll'
```r
tidy(x, ...)
```

Arguments

- `recipe` : A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` : One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. These columns should be non-integer numerics (i.e., double precision). For the tidy method, these are not currently used.
- `role` : Not used by this step since no new variables are created.
step_impute_roll

trained A logical to indicate if the quantities for preprocessing have been estimated.
columns A named numeric vector of columns. This is NULL until computed by prep.recipe().
ostatistic A function with a single argument for the data to compute the imputed value. Only complete values will be passed to the function and it should return a double precision value.
window The size of the window around a point to be imputed. Should be an odd integer greater than one. See Details below for a discussion of points at the ends of the series.
skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id A character string that is unique to this step to identify it.
x A step_impute_roll object.

Details

On the tails, the window is shifted towards the ends. For example, for a 5-point window, the windows for the first four points are 1:5, 1:5, 1:5, and then 2:6.

When missing data are in the window, they are not passed to the function. If all of the data in the window are missing, a missing value is returned.

The statistics are calculated on the training set values before imputation. This means that if previous data within the window are missing, their imputed values are not included in the window data used for imputation. In other words, each imputation does not know anything about previous imputations in the series prior to the current point.

As of recipes 0.1.16, this function name changed from step_rollimpute() to step_impute_roll().

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and window (the window size).

Examples

library(lubridate)

set.seed(145)
example_data <-
data.frame(
  day = ymd("2012-06-07") + days(1:12),
  x1 = round(runif(12), 2),
  x2 = round(runif(12), 2),
  x3 = round(runif(12), 2)
)
example_data$x1[c(1, 5, 6)] <- NA
example_data$x2[c(1:4, 10)] <- NA

library(recipes)
seven_pt <- recipe(~ . , data = example_data) %>%
  update_role(day, new_role = "time_index") %>%
  step_impute_roll(all_numeric_predictors(), window = 7) %>%
  prep(training = example_data)

# The training set:
bake(seven_pt, new_data = NULL)

---

**step_indicate_na**  
Create Missing Data Column Indicators

**Description**

`step_indicate_na` creates a specification of a recipe step that will create and append additional binary columns to the dataset to indicate which observations are missing.

**Usage**

```r
step_indicate_na(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  prefix = "na_ind",
  skip = FALSE,
  id = rand_id("indicate_na")
)
```

```r
## S3 method for class 'step_indicate_na'
tidy(x, ...)
```

**Arguments**

- `recipe`  
  A recipe object. The check will be added to the sequence of operations for this recipe.

- `...`  
  One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.

- `role`  
  For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new na indicator columns created from the original variables will be used as predictors in a model.

- `trained`  
  A logical for whether the selectors in `...` have been resolved by `prep()`.
**Step Integer**

**columns** A character string of variable names that will be populated (eventually) by the `terms` argument.

**prefix** A character string that will be the prefix to the resulting new variables. Defaults to "na_ind".

**skip** A logical. Should the check be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

**id** A character string that is unique to this step to identify it.

**x** A `step_indicate_na` object.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `model` (the median value).

**Examples**

```r
library(modeldata)
data("credit_data")

## missing data per column
purrr::map_dbl(credit_data, function(x) mean(is.na(x)))

set.seed(342)
in_training <- sample(1:nrow(credit_data), 2000)
credit_tr <- credit_data[ in_training, ]
credit_te <- credit_data[-in_training, ]
rec <- recipe(Price ~ ., data = credit_tr)

impute_rec <- rec %>%
  step_indicate_na(Income, Assets, Debt)

imp_models <- prep(impute_rec, training = credit_tr)
imputed_te <- bake(imp_models, new_data = credit_te, everything())
```

**Description**

`step_integer` creates a specification of a recipe step that will convert new data into a set of integers based on the original data values.
step_integer

Usage

```r
step_integer(
    recipe,
    ..., role = "predictor",
    trained = FALSE,
    strict = FALSE,
    zero_based = FALSE,
    key = NULL,
    skip = FALSE,
    id = rand_id("integer")
)
```

## S3 method for class 'step_integer'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**: One or more selector functions to choose which variables will be used to create the integer variables. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new columns created by the original variables will be used as predictors in a model.

- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.

- **strict**: A logical for whether the values should be returned as integers (as opposed to double).

- **zero_based**: A logical for whether the integers should start at zero and new values be appended as the largest integer.

- **key**: A list that contains the information needed to create integer variables for each variable contained in `terms`. This is NULL until the step is trained by `prep.recipe()`.

- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**: A character string that is unique to this step to identify it.

- **x**: A `step_integer` object.

Details

`step_integer` will determine the unique values of each variable from the training set (excluding missing values), order them, and then assign integers to each value. When baked, each data point
step_interact

Create Interaction Variables

Description

step_interact creates a specification of a recipe step that will create new columns that are interaction terms between two or more variables.
Usage

```r
step_interact(
  recipe,
  terms,
  role = "predictor",
  trained = FALSE,
  objects = NULL,
  sep = "_x_",
  skip = FALSE,
  id = rand_id("interact")
)
```

```r
## S3 method for class 'step_interact'
tidy(x, ...)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **terms**: A traditional R formula that contains interaction terms. This can include `.`, and selectors.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new columns created from the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **objects**: A list of `terms` objects for each individual interaction.
- **sep**: A character value used to delineate variables in an interaction (e.g. `var1_x_var2` instead of the more traditional `var1:var2`).
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_interact` object
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details, and consider using `tidyselect::starts_with()` when dummy variables have been created. For the tidy method, these are not currently used.

Details

`step_interact` can create interactions between variables. It is primarily intended for numeric data; categorical variables should probably be converted to dummy variables using `step_dummy()` prior to being used for interactions.
Unlike other step functions, the `terms` argument should be a traditional R model formula but should contain no inline functions (e.g. `log`). For example, for predictors A, B, and C, a formula such as `~A:B:C` can be used to make a three way interaction between the variables. If the formula contains terms other than interactions (e.g. `(A+B+C)^3`) only the interaction terms are retained for the design matrix.

The separator between the variables defaults to "\_\_\_" so that the three way interaction shown previously would generate a column named `A\_\_\_B\_\_\_C`. This can be changed using the `sep` argument.

When dummy variables are created and are used in interactions, selectors can help specify the interactions succinctly. For example, suppose a factor column `X` gets converted to dummy variables `x_2, x_3, ..., x_6` using `step_dummy()`. If you wanted an interaction with numeric column `z`, you could create a set of specific interaction effects (e.g. `x_2:z + x_3:z` and so on) or you could use `starts_with("x_")`:z. When `prep()` evaluates this step, `starts_with("x_")` resolves to `(x_2 + x_3 + x_4 + x_5 + x_6)` so that the formula is now `(x_2 + x_3 + x_4 + x_5 + x_6):z` and all two-way interactions are created.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the interaction effects.

### Examples

```r
library(modeldata)
data(penguins)
penguins <- penguins %>% na.omit()

rec <- recipe(flipper_length_mm ~ ., data = penguins)
int_mod_1 <- rec %>%
  step_interact(terms = ~ bill_depth_mm:bill_length_mm)
# specify all dummy variables succinctly with \`starts_with()\`
int_mod_2 <- rec %>%
  step_dummy(sex, species, island) %>%
  step_interact(terms = ~ body_mass_g:starts_with("species"))

int_mod_1 <- prep(int_mod_1, training = penguins)
int_mod_2 <- prep(int_mod_2, training = penguins)

dat_1 <- bake(int_mod_1, penguins)
dat_2 <- bake(int_mod_2, penguins)

names(dat_1)
names(dat_2)

tidy(int_mod_1, number = 1)
tidy(int_mod_2, number = 2)
```
Step Interception

**Description**

*step_intercept* creates a specification of a recipe step that will add an intercept or constant term in the first column of a data matrix. *step_intercept* has defaults to *predictor* role so that it is by default called in the bake step. Be careful to avoid unintentional transformations when calling steps with *all_predictors*.

**Usage**

```r
step_intercept(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE, 
  name = "intercept",
  value = 1,
  skip = FALSE,
  id = rand_id("intercept")
)
```

**Arguments**

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: Argument ignored; included for consistency with other step specification functions.
- **role**: For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new columns created from the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated. Again included for consistency.
- **name**: Character name for newly added column
- **value**: A numeric constant to fill the intercept column. Defaults to 1.
- **skip**: A logical. Should the step be skipped when the recipe is baked by *bake.recipe()*? While all operations are baked when *prep.recipe()* is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations
- **id**: A character string that is unique to this step to identify it.

**Value**

An updated version of *recipe* with the new step added to the sequence of existing steps (if any).
See Also

recipe() prep.recipe() bake.recipe()

Examples

library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
data = biomass_tr)
rec_trans <- recipe(HHV ~ ., data = biomass_tr[, -(1:2)]) %>%
  step_intercept(value = 2) %>%
  step_scale(carbon)

rec_obj <- prep(rec_trans, training = biomass_tr)

with_intercept <- bake(rec_obj, biomass_te)
with_intercept

step_inverse

Inverse Transformation

Description

step_inverse creates a specification of a recipe step that will inverse transform the data.

Usage

step_inverse(
  recipe,
  ...,
  role = NA,
  offset = 0,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("inverse")
)

## S3 method for class 'step_inverse'
tidy(x, ...)

---

step_inverse  Inverse Transformation

---

Description

step_inverse creates a specification of a recipe step that will inverse transform the data.

Usage

step_inverse(
  recipe,
  ...,
  role = NA,
  offset = 0,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("inverse")
)

## S3 method for class 'step_inverse'
tidy(x, ...)

---
Arguments

recipe  A recipe object. The step will be added to the sequence of operations for this
        recipe.

... One or more selector functions to choose which variables are affected by the
        step. See selections() for more details. For the tidy method, these are not
        currently used.

role  Not used by this step since no new variables are created.

offset  An optional value to add to the data prior to logging (to avoid 1/0).

trained  A logical to indicate if the quantities for preprocessing have been estimated.

columns  A character string of variable names that will be populated (eventually) by the
        terms argument.

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()?
        While all operations are baked when prep.recipe() is run, some operations
        may not be able to be conducted on new data (e.g. processing the outcome
        variable(s)). Care should be taken when using skip = TRUE as it may affect the
        computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_inverse object.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).
For the tidy method, a tibble with columns terms which is the columns that will be affected.

See Also

    step_log() step_sqrt() step_hyperbolic() recipe() prep.recipe() bake.recipe()

Examples

    set.seed(313)
    examples <- matrix(runif(40), ncol = 2)
    examples <- data.frame(examples)
    rec <- recipe(~ X1 + X2, data = examples)  
    inverse_trans <- rec %>%
        step_inverse(all_numeric_predictors())
    inverse_obj <- prep(inverse_trans, training = examples)
    transformed_te <- bake(inverse_obj, examples)
    plot(examples$X1, transformed_te$X1)
    tidy(inverse_trans, number = 1)
    tidy(inverse_obj, number = 1)
**Description**

`step_invlogit` creates a specification of a recipe step that will transform the data from real values to be between zero and one.

**Usage**

```r
define = step_invlogit(
    recipe,
    ..., 
    role = NA,
    trained = FALSE,
    columns = NULL,
    skip = FALSE,
    id = rand_id("invlogit")
)
```

```r
## S3 method for class 'step_invlogit'
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `columns` A character string of variable names that will be populated (eventually) by the terms argument.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
- `id` A character string that is unique to this step to identify it.
- `x` A `step_invlogit` object.

**Details**

The inverse logit transformation takes values on the real line and translates them to be between zero and one using the function \( f(x) = 1/(1+\exp(-x)) \).
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the columns that will be affected.

See Also

step_logit() step_log() step_sqrt() step_hyperbolic() recipe() prep.recipe() bake.recipe()

Examples

library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
             data = biomass_tr)

ilogit_trans <- rec %>%
    step_center(carbon, hydrogen) %>%
    step_scale(carbon, hydrogen) %>%
    step_invlogit(carbon, hydrogen)

ilogit_obj <- prep(ilogit_trans, training = biomass_tr)

transformed_te <- bake(ilogit_obj, biomass_te)
plot(biomass_te$carbon, transformed_te$carbon)

---

**step_isomap**

*Isomap Embedding*

Description

step_isomap creates a specification of a recipe step that will convert numeric data into one or more new dimensions.

Usage

```r
step_isomap(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_terms = 5,
  neighbors = 50,
  options = list(.mute = c("message", "output")),
  res = NULL,
```

---

98
step_isomap

prefix = "Isomap",
keep_original_cols = FALSE,
skip = FALSE,
id = rand_id("isomap")
)

## S3 method for class 'step_isomap'
tidy(x, ...)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this
... recipe.
role One or more selector functions to choose which variables will be used to com-
compute the dimensions. See selections() for more details. For the tidy method,
these are not currently used.
trained For model terms created by this step, what analysis role should they be as-
assigned?. By default, the function assumes that the new dimension columns
created by the original variables will be used as predictors in a model.
num_terms A logical to indicate if the quantities for preprocessing have been estimated.
neighbors The number of isomap dimensions to retain as new predictors. If num_terms
is greater than the number of columns or the number of possible dimensions, a
smaller value will be used.
options The number of neighbors.
res A list of options to dimRed::Isomap().
prefix A logical to indicate if the quantities for preprocessing have been estimated.
keep_original_cols The dimRed::Isomap() object is stored here once this preprocessing step has
be trained by prep.recipe().
options A logical to keep the original variables in the output. Defaults to FALSE.
res A character string that will be the prefix to the resulting new variables. See notes
below.
keep_original_cols A logical. Should the step be skipped when the recipe is baked by bake.recipe()?
skip While all operations are baked when prep.recipe() is run, some operations
may not be able to be conducted on new data (e.g. processing the outcome
variable(s)). Care should be taken when using skip = TRUE as it may affect the
computations for subsequent operations.

id A character string that is unique to this step to identify it.
x A character string that will be the prefix to the resulting new variables. See notes
below.

Details

Isomap is a form of multidimensional scaling (MDS). MDS methods try to find a reduced set of
dimensions such that the geometric distances between the original data points are preserved. This
version of MDS uses nearest neighbors in the data as a method for increasing the fidelity of the new
dimensions to the original data values.
This step requires the **dimRed**, **RSpectra**, **igraph**, and **RANN** packages. If not installed, the step will stop with a note about installing these packages.

It is advisable to center and scale the variables prior to running Isomap (**step_center** and **step_scale** can be used for this purpose).

The argument **num_terms** controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if **num_terms < 10**, their names will be **Isomap1 - Isomap9**. If **num_terms = 101**, the names would be **Isomap001 - Isomap101**.

**Value**

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns **terms** (the selectors or variables selected).

**References**


**See Also**

`step_pca()` `step_kpca()` `step_ica()` `recipe()` `prep.recipe()` `bake.recipe()`

**Examples**

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, 
data = biomass_tr)

im_trans <- rec %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_isomap(all_numeric_predictors(), neighbors = 100, num_terms = 2)

if (require(dimRed) & require(RSpectra)) {
  im_estimates <- prep(im_trans, training = biomass_tr)
  im_te <- bake(im_estimates, biomass_te)

  rng <- extendrange(c(im_te$Isomap1, im_te$Isomap2))
  plot(im_te$Isomap1, im_te$Isomap2, 
       xlim = rng, ylim = rng)
```
**step_kpca**

```r
 tidy(im_trans, number = 3)
tidy(im_estimates, number = 3)
```

---

**step_kpca**  
**Kernel PCA Signal Extraction**

**Description**

`step_kpca` a *specification* of a recipe step that will convert numeric data into one or more principal components using a kernel basis expansion.

**Usage**

```r
step_kpca(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  res = NULL,
  options = list(kernel = "rbfdot", kpar = list(sigma = 0.2)),
  prefix = "kPC",
  skip = FALSE,
  id = rand_id("kpca")
)
```

```r
## S3 method for class 'step_kpca'
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables will be used to compute the components. See `selections()` for more details. For the `tidy` method, these are not currently used.
- `role` For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new principal component columns created by the original variables will be used as predictors in a model.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `num_comp` The number of PCA components to retain as new predictors. If `num_comp` is greater than the number of columns or the number of possible components, a smaller value will be used.
An S4 \texttt{kernlab::kpca()} object is stored here once this preprocessing step has been trained by \texttt{prep.recipe()}.

A list of options to \texttt{kernlab::kpca()}. Defaults are set for the arguments kernel and \texttt{kpar} but others can be passed in. \textbf{Note} that the arguments \texttt{x} and \texttt{features} should not be passed here (or at all).

A character string that will be the prefix to the resulting new variables. See notes below.

A logical. Should the step be skipped when the recipe is baked by \texttt{bake.recipe()}? While all operations are baked when \texttt{prep.recipe()} is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using \texttt{skip = TRUE} as it may affect the computations for subsequent operations.

A character string that is unique to this step to identify it.

A \texttt{step_kpca} object

Kernel principal component analysis (kPCA) is an extension of a PCA analysis that conducts the calculations in a broader dimensionality defined by a kernel function. For example, if a quadratic kernel function were used, each variable would be represented by its original values as well as its square. This nonlinear mapping is used during the PCA analysis and can potentially help find better representations of the original data.

This step requires the \texttt{dimRed} and \texttt{kernlab} packages. If not installed, the step will stop with a note about installing these packages.

As with ordinary PCA, it is important to standardize the variables prior to running PCA (\texttt{step_center} and \texttt{step_scale} can be used for this purpose).

When performing kPCA, the kernel function (and any important kernel parameters) must be chosen. The \texttt{kernlab} package is used and the reference below discusses the types of kernels available and their parameter(s). These specifications can be made in the \texttt{kernel} and \texttt{kpar} slots of the \texttt{options} argument to \texttt{step_kpca}.

The argument \texttt{num_comp} controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with \texttt{prefix} and a sequence of numbers. The variable names are padded with zeros. For example, if \texttt{num_comp < 10}, their names will be kPC1 - kPC9. If \texttt{num_comp = 101}, the names would be kPC001 - kPC101.

An updated version of \texttt{recipe} with the new step added to the sequence of existing steps (if any). For the \texttt{tidy} method, a tibble with columns \texttt{terms} (the selectors or variables selected).


step_kpca_poly

**Polynomial Kernel PCA Signal Extraction**

**Description**

step_kpca_poly a specification of a recipe step that will convert numeric data into one or more principal components using a polynomial kernel basis expansion.

**Usage**

```r
step_kpca_poly(
  recipe,
  ..., 
  role = "predictor", 
  trained = FALSE,  
  num_comp = 5, 
  res = NULL, 
)```

**Examples**

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, 
  data = biomass_tr)

kpca_trans <- rec %>%
  step_YeoJohnson(all_numeric_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_kpca(all_numeric_predictors())

if (require(dimRed) & require(kernlab)) {
  kpca_estimates <- prep(kpca_trans, training = biomass_tr)

  kpca_te <- bake(kpca_estimates, biomass_te)

  rng <- extendrange(c(kpca_te$kPC1, kpca_te$kPC2))
  plot(kpca_te$kPC1, kpca_te$kPC2, 
       xlim = rng, ylim = rng)
  tidy(kpca_trans, number = 3)
  tidy(kpca_estimates, number = 3)
}
step_kpca_poly

degree = 2,
scale_factor = 1,
offset = 1,
prefix = "kPC",
keep_original_cols = FALSE,
skip = FALSE,
id = rand_id("kpca_poly")
)

## S3 method for class 'step_kpca_poly'
tidy(x, ...)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this
recipe.

... One or more selector functions to choose which variables will be used to com-
pute the components. See selections() for more details. For the tidy method,
these are not currently used.

role For model terms created by this step, what analysis role should they be assigned?
By default, the function assumes that the new principal component columns
created by the original variables will be used as predictors in a model.

trained A logical to indicate if the quantities for preprocessing have been estimated.

num_comp The number of PCA components to retain as new predictors. If num_comp is
greater than the number of columns or the number of possible components, a
smaller value will be used.

res An S4 kernlab::kpca() object is stored here once this preprocessing step has
be trained by prep.recipe().

degree, scale_factor, offset
Numeric values for the polynomial kernel function.

prefix A character string that will be the prefix to the resulting new variables. See notes
below.

keep_original_cols
A logical to keep the original variables in the output. Defaults to FALSE.

skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()?
While all operations are baked when prep.recipe() is run, some operations
may not be able to be conducted on new data (e.g. processing the outcome
variable(s)). Care should be taken when using skip = TRUE as it may affect the
computations for subsequent operations

id A character string that is unique to this step to identify it.

x A step_kpca_poly object

Details

Kernel principal component analysis (kPCA) is an extension of a PCA analysis that conducts the
calculations in a broader dimensionality defined by a kernel function. For example, if a quadratic
kernel function were used, each variable would be represented by its original values as well as its square. This nonlinear mapping is used during the PCA analysis and can potentially help find better representations of the original data.

This step requires the dimRed and kernlab packages. If not installed, the step will stop with a note about installing these packages.

As with ordinary PCA, it is important to standardize the variables prior to running PCA (step_center and step_scale can be used for this purpose).

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if num_comp < 10, their names will be kPC1 - kPC9. If num_comp = 101, the names would be kPC001 - kPC101.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected).

References


See Also

step_pca() step_ica() step_isomap() recipe() prep.recipe() bake.recipe()
step_k pca_rbf

Radial Basis Function Kernel PCA Signal Extraction

Description

step_k pca_rbf a specification of a recipe step that will convert numeric data into one or more principal components using a radial basis function kernel basis expansion.

Usage

step_k pca_rbf(
  recipe,
  ...,  
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  res = NULL,
  sigma = 0.2,
  prefix = "kPC",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("kpca_rbf")
)

## S3 method for class 'step_k pca_rbf'
tidy(x, ...)

Arguments

  recipe A recipe object. The step will be added to the sequence of operations for this recipe.

  ... One or more selector functions to choose which variables will be used to compute the components. See selections() for more details. For the tidy method, these are not currently used.

  role For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new principal component columns created by the original variables will be used as predictors in a model.

  trained A logical to indicate if the quantities for preprocessing have been estimated.
step_kpca_rbf

num_comp
The number of PCA components to retain as new predictors. If num_comp is greater than the number of columns or the number of possible components, a smaller value will be used.

res
An S4 kernlab::kpca() object is stored here once this preprocessing step has been trained by prep.recipe().

sigma
A numeric value for the radial basis function parameter.

prefix
A character string that will be the prefix to the resulting new variables. See notes below.

keep_original_cols
A logical to keep the original variables in the output. Defaults to FALSE.

skip
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id
A character string that is unique to this step to identify it.

x
A step_kpca_rbf object

Details
Kernel principal component analysis (kPCA) is an extension of a PCA analysis that conducts the calculations in a broader dimensionality defined by a kernel function. For example, if a quadratic kernel function were used, each variable would be represented by its original values as well as its square. This nonlinear mapping is used during the PCA analysis and can potentially help find better representations of the original data.

This step requires the dimRed and kernlab packages. If not installed, the step will stop with a note about installing these packages.

As with ordinary PCA, it is important to standardize the variables prior to running PCA (step_center and step_scale can be used for this purpose).

The argument num_comp controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with prefix and a sequence of numbers. The variable names are padded with zeros. For example, if num_comp < 10, their names will be kPC1 - kPC9. If num_comp = 101, the names would be kPC001 - kPC101.

Value
An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected).

References

See Also

step_pca() step_ica() step_isomap() recipe() prep.recipe() bake.recipe()

Examples

library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
               data = biomass_tr)

kpca_trans <- rec %>%
               step_YeoJohnson(all_numeric_predictors()) %>%
               step_normalize(all_numeric_predictors()) %>%
               step_kpca_rbf(all_numeric_predictors())

if (require(dimRed) & require(kernlab)) {
  kpca_estimates <- prep(kpca_trans, training = biomass_tr)
  kpca_te <- bake(kpca_estimates, biomass_te)
  rng <- extendrange(c(kpca_te$kPC1, kpca_te$kPC2))
  plot(kpca_te$kPC1, kpca_te$kPC2,
       xlim = rng, ylim = rng)
  tidy(kpca_trans, number = 3)
  tidy(kpca_estimates, number = 3)
}

---

## step_lag

**Create a lagged predictor**

### Description

step_lag creates a specification of a recipe step that will add new columns of lagged data. Lagged data will by default include NA values where the lag was induced. These can be removed with step_naomit(), or you may specify an alternative filler value with the default argument.

### Usage

```r
step_lag(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  lag = 1,
...)
```
step_lag

```r
prefix = "lag_",
default = NA,
columns = NULL,
skip = FALSE,
id = rand_id("lag")
)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details.
- `role` Defaults to "predictor"
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `lag` A vector of positive integers. Each specified column will be lagged for each value in the vector.
- `prefix` A prefix for generated column names, default to "lag_".
- `default` Passed to `dplyr::lag`, determines what fills empty rows left by lagging (defaults to NA).
- `columns` A character string of variable names that will be populated (eventually) by the terms argument.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
- `id` A character string that is unique to this step to identify it.

**Details**

The step assumes that the data are already *in the proper sequential order* for lagging.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

**See Also**

- `recipe()`  
- `prep.recipe()`  
- `bake.recipe()`  
- `step_naomit()`

**Examples**

```r
n <- 10
start <- as.Date('1999/01/01')
end <- as.Date('1999/01/10')
```
df <- data.frame(x = runif(n),
                 index = 1:n,
                 day = seq(start, end, by = "day"))

recipe(~ ., data = df) %>%
  step_lag(index, day, lag = 2:3) %>%
  prep(df) %>%
  bake(df)

---

**Description**

*step_lincomb* creates a specification of a recipe step that will potentially remove numeric variables that have linear combinations between them.

**Usage**

```r
step_lincomb(
  recipe, 
  ...,
  role = NA,
  trained = FALSE,
  max_steps = 5,
  removals = NULL,
  skip = FALSE,
  id = rand_id("lincomb")
)
```

```
## S3 method for class 'step_lincomb'
tidy(x, ...)
```

**Arguments**

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables are affected by the step. See *selections()* for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **max_steps**
  A value.

- **removals**
  A character string that contains the names of columns that should be removed. These values are not determined until *prep.recipe()* is called.
skip  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A `step_lincomb` object.

**Details**

This step finds exact linear combinations between two or more variables and recommends which column(s) should be removed to resolve the issue. This algorithm may need to be applied multiple times (as defined by `max_steps`).

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the columns that will be removed.

**Author(s)**

Max Kuhn, Kirk Mettler, and Jed Wing

**See Also**

`step_nzv()` `recipe()` `prep.recipe()` `bake.recipe()`

**Examples**

```r
library(modeldata)
data(biomass)

biomass$new_1 <- with(biomass, .1*carbon - .2*hydrogen + .6*sulfur)
biomass$new_2 <- with(biomass, .5*carbon - .2*oxygen + .6*nitrogen)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur + new_1 + new_2, data = biomass_tr)

lincomb_filter <- rec %>% step_lincomb(all_numeric_predictors())

lincomb_filter_trained <- prep(lincomb_filter, training = biomass_tr)
lincomb_filter_trained

tidy(lincomb_filter, number = 1)
```
**step_log**

`tidy(lincomb_filter_trained, number = 1)`

---

### Description

`step_log` creates a specification of a recipe step that will log transform data.

### Usage

```r
step_log(
  recipe,
  ..., 
  role = NA, 
  trained = FALSE, 
  base = exp(1), 
  offset = 0, 
  columns = NULL, 
  skip = FALSE, 
  signed = FALSE, 
  id = rand_id("log")
)
```

```r
# S3 method for class 'step_log'
tidy(x, ...)
```

### Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the `tidy` method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **base**: A numeric value for the base.
- **offset**: An optional value to add to the data prior to logging (to avoid `log(0)`).
- **columns**: A character string of variable names that will be populated (eventually) by the terms argument.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
step_log

signed A logical indicating whether to take the signed log. This is sign(x) * abs(x) when abs(x) => 1 or 0 if abs(x) < 1. If TRUE the offset argument will be ignored.

id A character string that is unique to this step to identify it.

x A step_log object.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the columns that will be affected) and base.

See Also

step_logit() step_invlogit() step_hyperbolic() step_sqrt() recipe() prep.recipe() bake.recipe()

Examples

```r
set.seed(313)
examples <- matrix(exp(rnorm(40)), ncol = 2)
examples <- as.data.frame(examples)

rec <- recipe(~ V1 + V2, data = examples)

log_trans <- rec %>%
  step_log(all_numeric_predictors())

log_obj <- prep(log_trans, training = examples)

transformed_te <- bake(log_obj, examples)
plot(examples$V1, transformed_te$V1)

tidy(log_trans, number = 1)
tidy(log_obj, number = 1)

# using the signed argument with negative values

examples2 <- matrix(rnorm(40, sd = 5), ncol = 2)
examples2 <- as.data.frame(examples2)

recipe(~ V1 + V2, data = examples2) %>%
  step_log(all_numeric_predictors()) %>%
  prep(training = examples2) %>%
bake(examples2)

recipe(~ V1 + V2, data = examples2) %>%
  step_log(all_numeric_predictors(), signed = TRUE) %>%
  prep(training = examples2) %>%
bake(examples2)
```
Description

`step_logit` creates a specification of a recipe step that will logit transform the data.

Usage

```r
step_logit(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("logit")
)
```

## S3 method for class 'step_logit'
tidy(x, ...)

Arguments

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the `tidy` method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `columns` A character string of variable names that will be populated (eventually) by the `terms` argument.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- `id` A character string that is unique to this step to identify it.
- `x` A `step_logit` object.

Details

The logit transformation takes values between zero and one and translates them to be on the real line using the function \( f(p) = \log(p/(1-p)) \).
Value
An updated version of recipe with the new step added to the sequence of existing steps (if any).
For the tidy method, a tibble with columns terms which is the columns that will be affected.

See Also
step_invlogit() step_log() step_sqrt() step_hyperbolic() recipe() prep.recipe() bake.recipe()

Examples

set.seed(313)
examples <- matrix(runif(40), ncol = 2)
examples <- data.frame(examples)

rec <- recipe(~ X1 + X2, data = examples)

logit_trans <- rec %>%
  step_logit(all_numeric_predictors())

logit_obj <- prep(logit_trans, training = examples)

transformed_te <- bake(logit_obj, examples)

plot(examples$X1, transformed_te$X1)
tidy(logit_trans, number = 1)
tidy(logit_obj, number = 1)

---

step_mutate

Add new variables using dplyr

Description
step_mutate creates a specification of a recipe step that will add variables using dplyr::mutate().

Usage

step_mutate(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  inputs = NULL,
  skip = FALSE,
  id = rand_id("mutate")
)

## S3 method for class 'step_mutate'
tidy(x, ...)
## S3 method for class 'step_mutate_at'
tidy(x, ...)

### Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: Name-value pairs of expressions. See `dplyr::mutate()`. If the argument is not named, the expression is converted to a column name.
- **role**: For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new dimension columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **inputs**: Quosure(s) of ....
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_mutate` object

### Details

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!`) to embed the value of the object in the expression (to be portable between sessions). See the examples.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `values` which contains the `mutate` expressions as character strings (and are not reparsable).

### Examples

```r
rec <-
  recipe(~ ., data = iris) %>%
  step_mutate(
    dbl_width = Sepal.Width * 2,
    half_length = Sepal.Length / 2
  )

prepped <- prep(rec, training = iris %>% slice(1:75))

library(dplyr)
```
**step_mutate_at**  

_Mutate multiple columns using dplyr_

---

**Description**

*step_mutate_at* creates a _specification_ of a recipe step that will modify the selected variables using a common function via `dplyr::mutate_at()`.

**Usage**

`step_mutate_at()`
step_mutate_at

```
recipe,
..., 
fn,
role = "predictor",
trained = FALSE,
inputs = NULL,
skip = FALSE,
id = rand_id("mutate_at")
)

Arguments

  recipe
  A recipe object. The step will be added to the sequence of operations for this
  recipe.
  ...
  One or more selector functions to choose which variables are affected by the
  step. See selections() for more details. For the tidy method, these are not
  currently used.
  fn
  A function fun, a quosure style lambda ‘~ fun(.)‘ or a list of either form. (see
dplyr::mutate_at()). Note that this argument must be named.
  role
  For model terms created by this step, what analysis role should they be assigned?
  By default, the function assumes that the new dimension columns created by the
  original variables will be used as predictors in a model.
  trained
  A logical to indicate if the quantities for preprocessing have been estimated.
  inputs
  A vector of column names populated by prep().
  skip
  A logical. Should the step be skipped when the recipe is baked by bake.recipe()?
  While all operations are baked when prep.recipe() is run, some operations
  may not be able to be conducted on new data (e.g. processing the outcome
  variable(s)). Care should be taken when using skip = TRUE as it may affect the
  computations for subsequent operations.
  id
  A character string that is unique to this step to identify it.

Value

  An updated version of recipe with the new step added to the sequence of existing steps (if any).
  For the tidy method, a tibble with columns terms which contains the columns being transformed.

Examples

library(dplyr)
recipe(~ ., data = iris) %>%
  step_mutate_at(contains("Length"), fn = ~ 1./.) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  slice(1:10)

recipe(~ ., data = iris) %>%
  # leads to more columns being created.
  step_mutate_at(contains("Length"), fn = list(log = log, sqrt = sqrt)) %>%
```
**Description**

`step_naomit` creates a *specification* of a recipe step that will remove observations (rows of data) if they contain `NA` or `NaN` values.

**Usage**

```r
step_naomit(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("naomit")
)
```

```r
## S3 method for class 'step_naomit'
tidy(x, ...)
```

**Arguments**

- `recipe`  
  - A recipe object. The step will be added to the sequence of operations for this recipe.
- `...`  
  - One or more selector functions to choose which variables will be used to remove observations containing `NA` or `NaN` values. See `selections()` for more details.
- `role`  
  - Unused, include for consistency with other steps.
- `trained`  
  - A logical to indicate if the quantities for preprocessing have been estimated. Again included for consistency.
- `columns`  
  - A character string of variable names that will be populated (eventually) by the `terms` argument.
- `skip`  
  - A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = FALSE`.
- `id`  
  - A character string that is unique to this step to identify it.
- `x`  
  - A `step_naomit` object.
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

Row Filtering

This step can entirely remove observations (rows of data), which can have unintended and/or problematic consequences when applying the step to new data later via bake.recipe(). Consider whether skip = TRUE or skip = FALSE is more appropriate in any given use case. In most instances that affect the rows of the data being predicted, this step probably should not be applied at all; instead, execute operations like this outside and before starting a preprocessing recipe().

See Also

step_filter() step_sample() step_slice()

Examples

recipe(Ozone ~ ., data = airquality) %>%
  step_naomit(Solar.R) %>%
  prep(airquality, verbose = FALSE) %>%
  bake(new_data = NULL)

---

**step_nnmf**  
**NNMF Signal Extraction**

Description

step_nnmf creates a specification of a recipe step that will convert numeric data into one or more non-negative components.

Usage

step_nnmf(
  recipe,  
  ...,  
  role = "predictor",  
  trained = FALSE,  
  num_comp = 2,  
  num_run = 30,  
  options = list(),  
  res = NULL,  
  prefix = "NNMF",  
  seed = sample.int(10^5, 1),  
  keep_original_cols = FALSE,  
  skip = FALSE,
### Arguments

**recipe**  
A recipe object. The step will be added to the sequence of operations for this recipe.

**...**  
One or more selector functions to choose which variables will be used to compute the components. See `selections()` for more details. For the `tidy` method, these are not currently used.

**role**  
For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new component columns created by the original variables will be used as predictors in a model.

**trained**  
A logical to indicate if the quantities for preprocessing have been estimated.

**num_comp**  
The number of components to retain as new predictors. If `num_comp` is greater than the number of columns or the number of possible components, a smaller value will be used.

**num_run**  
A positive integer for the number of computations runs used to obtain a consensus projection.

**options**  
A list of options to `nmf()` in the NMF package by way of the `NNMF()` function in the dimRed package. **Note** that the arguments `data` and `ndim` should not be passed here, and that NMF’s parallel processing is turned off in favor of resample-level parallelization.

**res**  
The `NNMF()` object is stored here once this preprocessing step has been trained by `prep.recipe()`.

**prefix**  
A character string that will be the prefix to the resulting new variables. See notes below.

**seed**  
An integer that will be used to set the seed in isolation when computing the factorization.

**keep_original_cols**  
A logical to keep the original variables in the output. Defaults to `FALSE`.

**skip**  
A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

**id**  
A character string that is unique to this step to identify it.

**x**  
A `step_nnmf` object.
Details

Non-negative matrix factorization computes latent components that have non-negative values and take into account that the original data have non-negative values.

The argument `num_comp` controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with `prefix` and a sequence of numbers. The variable names are padded with zeros. For example, if `num < 10`, their names will be `NNMF1 - NNMF9`. If `num = 101`, the names would be `NNMF001 - NNMF101`.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables selected) and the number of components.

See Also

`step_pca()`, `step_ica()`, `step_kpca()`, `step_isomap()`, `recipe()`, `prep.recipe()`, `bake.recipe()`

Examples

```r
library(modeldata)
data(biomass)

# rec <- recipe(HHV ~ ., data = biomass) %>%
# update_role(sample, new_role = "id var") %>%
# update_role(dataset, new_role = "split variable") %>%
# step_nnmf(all_numeric_predictors(), num_comp = 2, seed = 473, num_run = 2) %>%
# prep(training = biomass)
#
# bake(rec, new_data = NULL)
#
# library(ggplot2)
# bake(rec, new_data = NULL) %>%
# ggplot(aes(x = NNMF2, y = NNMF1, col = HHV)) + geom_point()
```

---

### step_normalize

Center and scale numeric data

**Description**

`step_normalize` creates a *specification* of a recipe step that will normalize numeric data to have a standard deviation of one and a mean of zero.
step_normalize

Usage

step_normalize(
  recipe,
  ..., 
  role = NA, 
  trained = FALSE, 
  means = NULL, 
  sds = NULL, 
  na_rm = TRUE, 
  skip = FALSE, 
  id = rand_id("normalize")
)

## S3 method for class 'step_normalize'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **means**: A named numeric vector of means. This is NULL until computed by `prep.recipe()`.
- **sds**: A named numeric vector of standard deviations. This is NULL until computed by `prep.recipe()`.
- **na_rm**: A logical value indicating whether NA values should be removed when computing the standard deviation and mean.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `bake.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_normalize` object.

Details

Centering data means that the average of a variable is subtracted from the data. Scaling data means that the standard deviation of a variable is divided out of the data. `step_normalize` estimates the variable standard deviations and means from the data used in the `training` argument of `prep.recipe`. `bake.recipe` then applies the scaling to new data sets using these estimates.
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected), value (the standard deviations and means), and statistic for the type of value.

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, 
              data = biomass_tr)

norm_trans <- rec %>%
              step_normalize(carbon, hydrogen)

norm_obj <- prep(norm_trans, training = biomass_tr)

transformed_te <- bake(norm_obj, biomass_te)

biomass_te[1:10, names(transformed_te)]
transformed_te
tidy(norm_trans, number = 1)
tidy(norm_obj, number = 1)

# To keep the original variables in the output, use `step_mutate_at`:

norm_keep_orig <- rec %>%
                  step_mutate_at(all_numeric_predictors(), fn = list(orig = ~.)) %>%
                  step_normalize(-contains("orig"), -all_outcomes())

keep_orig_obj <- prep(norm_keep_orig, training = biomass_tr)
keep_orig_te <- bake(keep_orig_obj, biomass_te)
keep_orig_te
```

---

**step_novel**

*Simple Value Assignments for Novel Factor Levels*

**Description**

*step_novel* creates a specification of a recipe step that will assign a previously unseen factor level to a new value.
**Usage**

```r
step_novel(
  recipe,
  ...,  # One or more selector functions to choose which variables that will be affected by
  role = NA,  # The step. These variables should be character or factor types. See selections()
  trained = FALSE,  # for more details. For the tidy method, these are not currently used.
  new_level = "new",  # A single character value that will be assigned to new factor levels.
  objects = NULL,  # A list of objects that contain the information on factor levels that will be deter-
  skip = FALSE,  # mined by prep.recipe().
  id = rand_id("novel")
)
```

```r
## S3 method for class 'step_novel'
 tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables that will be affected by the step. These variables should be character or factor types. See selections() for more details. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `new_level` A single character value that will be assigned to new factor levels.
- `objects` A list of objects that contain the information on factor levels that will be determined by prep.recipe().
- `skip` A logical. Should the step be skipped when the recipe is baked by bake.recipe()?
  While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- `id` A character string that is unique to this step to identify it.
- `x` A step_novel object.

**Details**

The selected variables are adjusted to have a new level (given by `new_level`) that is placed in the last position. During preparation there will be no data points associated with this new level since all of the data have been seen.

Note that if the original columns are character, they will be converted to factors by this step.

Missing values will remain missing.

If `new_level` is already in the data given to prep, an error is thrown.

When fitting a model that can deal with new factor levels, consider using workflows::add_recipe() with allow_novel_levels = TRUE set in hardhat::default_recipe_blueprint(). This will allow your model to handle new levels at prediction time, instead of throwing warnings or errors.
Description

`step_ns` creates a specification of a recipe step that will create new columns that are basis expansions of variables using natural splines.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the columns that will be affected) and `value` (the factor levels that is used for the new value).

See Also

`step_factor2string()`, `step_string2factor()`, `dummy_names()`, `step_regex()`, `step_count()`, `step_ordinalscore()`, `step_unorder()`, `step_other()`

Examples

```r
library(modeldata)
data(okc)

okc_tr <- okc[1:30000,]
okc_te <- okc[30001:30006,]
okc_te$diet[3] <- "cannibalism"
okc_te$diet[4] <- "vampirism"

rec <- recipe(~ diet + location, data = okc_tr)

rec <- rec %>%
  step_novel(diet, location)
rec <- prep(rec, training = okc_tr)

processed <- bake(rec, okc_te)
tibble(old = okc_te$diet, new = processed$diet)

tidy(rec, number = 1)
```

---

step_ns | Natural Spline Basis Functions

Description

`step_ns` creates a specification of a recipe step that will create new columns that are basis expansions of variables using natural splines.

Usage

```r
step_ns(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  objects = NULL,
  deg_free = 2,
) ```
## S3 method for class 'step_ns'
tidy(x, ...)

### Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new columns created from the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **objects**: A list of `splines::ns()` objects created once the step has been trained.
- **deg_free**: The degrees of freedom for the natural spline. As the degrees of freedom for a natural spline increase, more flexible and complex curves can be generated. When a single degree of freedom is used, the result is a rescaled version of the original data.
- **options**: A list of options for `splines::ns()` which should not include `x` or `df`.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_ns` object.

### Details

`step_ns` can create new features from a single variable that enable fitting routines to model this variable in a nonlinear manner. The extent of the possible nonlinearity is determined by the `df` or `knot` arguments of `splines::ns()`. The original variables are removed from the data and new columns are added. The naming convention for the new variables is `varname_ns_1` and so on.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` which is the columns that will be affected and `holiday`.
See Also

`step_poly()` `recipe()` `prep.recipe()` `bake.recipe()`

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
              data = biomass_tr)

with_splines <- rec %>%
               step_ns(carbon, hydrogen)

with_splines <- prep(with_splines, training = biomass_tr)

expanded <- bake(with_splines, biomass_te)
expanded
```

### step_num2factor

**Convert Numbers to Factors**

**Description**

`step_num2factor` will convert one or more numeric vectors to factors (ordered or unordered). This can be useful when categories are encoded as integers.

**Usage**

```r
step_num2factor(
  recipe,
  ..., 
  role = NA, 
  transform = function(x) x, 
  trained = FALSE, 
  levels, 
  ordered = FALSE, 
  skip = FALSE, 
  id = rand_id("num2factor")
)
```

```r
## S3 method for class 'step_num2factor'
tidy(x, ...)
```
Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.

- **role**: Not used by this step since no new variables are created.

- **transform**: A function taking a single argument `x` that can be used to modify the numeric values prior to determining the levels (perhaps using `base::as.integer()`). The output of a function should be an integer that corresponds to the value of `levels` that should be assigned. If not an integer, the value will be converted to an integer during `bake()`.

- **levels**: A character vector of values that will be used as the levels. These are the numeric data converted to character and ordered. This is modified once `prep.recipe()` is executed.

- **ordered**: A single logical value; should the factor(s) be ordered?

- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**: A character string that is unique to this step to identify it.

- **x**: A `step_num2factor` object.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `ordered`.

See Also

- `step_factor2string()`, `step_string2factor()`, `step_dummy()`

Examples

```r
library(dplyr)
library(modeldata)
data(attrition)

attrition %>%
  group_by(StockOptionLevel) %>%
count()

amnt <- c("nothin", "meh", "some", "copious")
rec <-
```
step_nzv

Near-Zero Variance Filter

Description

step_nzv creates a specification of a recipe step that will potentially remove variables that are highly sparse and unbalanced.
**step_nzv**

### Usage

```r
step_nzv(
  recipe,
  ...,  # one or more selector functions to choose which variables will be evaluated
  role = NA,
  trained = FALSE,
  freq_cut = 95/5,
  unique_cut = 10,
  options = list(freq_cut = 95/5, unique_cut = 10),
  removals = NULL,
  skip = FALSE,
  id = rand_id("nzv")
)
```

```r
## S3 method for class 'step_nzv'
tidy(x, ...)
```

### Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be evaluated by the filtering. See `selections()` for more details. For the `tidy` method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **freq_cut, unique_cut**: Numeric parameters for the filtering process. See the Details section below.
- **options**: A list of options for the filter (see Details below).
- **removals**: A character string that contains the names of columns that should be removed. These values are not determined until `prep.recipe()` is called.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
  While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_nzv` object.

### Details

This step diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that have both of the following characteristics:

1. they have very few unique values relative to the number of samples and
2. the ratio of the frequency of the most common value to the frequency of the second most common value is large.

For example, an example of near-zero variance predictor is one that, for 1000 samples, has two distinct values and 999 of them are a single value.

To be flagged, first, the frequency of the most prevalent value over the second most frequent value (called the "frequency ratio") must be above \( f_{\text{freq\_cut}} \). Secondly, the "percent of unique values," the number of unique values divided by the total number of samples (times 100), must also be below \( u_{\text{unique\_cut}} \).

In the above example, the frequency ratio is 999 and the unique value percent is 0.2%.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the columns that will be removed.

**See Also**

`step_corr()` `recipe()` `prep.recipe()` `bake.recipe()`

**Examples**

```r
library(modeldata)
data(biomass)

biomass$sparse <- c(1, rep(0, nrow(biomass) - 1))

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur + sparse,
              data = biomass_tr)

nzv_filter <- rec %>%
  step_nzv(all_predictors())

filter_obj <- prep(nzv_filter, training = biomass_tr)

filtered_te <- bake(filter_obj, biomass_te)

any(names(filtered_te) == "sparse")

tidy(nzv_filter, number = 1)
tidy(filter_obj, number = 1)
```
step_ordinalscores  

Convert Ordinal Factors to Numeric Scores

Description

step_ordinalscores creates a specification of a recipe step that will convert ordinal factor variables into numeric scores.

Usage

```r
step_ordinalscores(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  convert = as.numeric,
  skip = FALSE,
  id = rand_id("ordinalscores")
)
```

## S3 method for class 'step_ordinalscores'

tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See selections() for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **columns**: A character string of variables that will be converted. This is NULL until computed by prep.recipe().
- **convert**: A function that takes an ordinal factor vector as an input and outputs a single numeric variable.
- **skip**: A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A step_ordinalscores object.
Details

Dummy variables from ordered factors with \( C \) levels will create polynomial basis functions with \( C-1 \) terms. As an alternative, this step can be used to translate the ordered levels into a single numeric vector of values that represent (subjective) scores. By default, the translation uses a linear scale (1, 2, 3, ... \( C \)) but custom score functions can also be used (see the example below).

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the columns that will be affected).

Examples

```r
fail_lvls <- c("meh", "annoying", "really_bad")

ord_data <-
data.frame(item = c("paperclip", "twitter", "airbag"),
  fail_severity = factor(fail_lvls,
    levels = fail_lvls,
    ordered = TRUE))

model.matrix(~fail_severity, data = ord_data)

linear_values <- recipe(~ item + fail_severity, data = ord_data) %>%
  step_dummy(item) %>%
  step_ordinalscore(fail_severity)

linear_values <- prep(linear_values, training = ord_data)
bake(linear_values, new_data = NULL, everything())

custom <- function(x) {
  new_values <- c(1, 3, 7)
  new_values[as.numeric(x)]
}

nonlin_scores <- recipe(~ item + fail_severity, data = ord_data) %>%
  step_dummy(item) %>%
  step_ordinalscore(fail_severity, convert = custom)

tidy(nonlin_scores, number = 2)

nonlin_scores <- prep(nonlin_scores, training = ord_data)
bake(nonlin_scores, new_data = NULL, everything())
tidy(nonlin_scores, number = 2)
```
Description

`step_other` creates a specification of a recipe step that will potentially pool infrequently occurring values into an "other" category.

Usage

```r
step_other(
  recipe,
  ..., role = NA,
  trained = FALSE,
  threshold = 0.05,
  other = "other",
  objects = NULL,
  skip = FALSE,
  id = rand_id("other")
)
```

```r
## S3 method for class 'step_other'
tidy(x, ...)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will potentially be reduced. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **threshold**: A numeric value between 0 and 1 or an integer greater or equal to one. If it’s less than one then factor levels whose rate of occurrence in the training set are below threshold will be "othered". If it’s greater or equal to one then it’s treated as a frequency and factor levels that occur less then threshold times will be "othered".
- **other**: A single character value for the "other" category.
- **objects**: A list of objects that contain the information to pool infrequent levels that is determined by `prep.recipe()`.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome...
variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

<table>
<thead>
<tr>
<th>id</th>
<th>A character string that is unique to this step to identify it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A step_other object.</td>
</tr>
</tbody>
</table>

**Details**

The overall proportion (or total counts) of the categories are computed. The "other" category is used in place of any categorical levels whose individual proportion (or frequency) in the training set is less than `threshold`.

If no pooling is done the data are unmodified (although character data may be changed to factors based on the value of `strings_as_factors` in `prep.recipe()`). Otherwise, a factor is always returned with different factor levels.

If `threshold` is less than the largest category proportion, all levels except for the most frequent are collapsed to the other level.

If the retained categories include the value of `other`, an error is thrown. If `other` is in the list of discarded levels, no error occurs.

If no pooling is done, novel factor levels are converted to missing. If pooling is needed, they will be placed into the other category.

When data to be processed contains novel levels (i.e., not contained in the training set), the other category is assigned.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the columns that will be affected) and `retained` (the factor levels that were not pulled into "other")

**See Also**

- `step_factor2string()`, `step_string2factor()`, `dummy_names()`, `step_regex()`, `step_count()`, `step_ordinalscore()`, `step_unorder()`, `step_novel()`

**Examples**

```r
library(modeldata)
data(okc)
set.seed(19)
in_train <- sample(1:nrow(okc), size = 30000)
okc_tr <- okc[in_train,]
okc_te <- okc[-in_train,]
rec <- recipe(~ diet + location, data = okc_tr)
rec <- rec %>%
```

step_pca(diet, location, threshold = .1, other = "other values")
rec <- prep(rec, training = okc_tr)

collapsed <- bake(rec, okc_te)
table(okc_te$diet, collapsed$diet, useNA = "always")

tidy(rec, number = 1)

# novel levels are also "othered"
tahiti <- okc[1,]
tahiti$location <- "a magical place"
bake(rec, tahiti)

# threshold as a frequency
rec <- recipe(~ diet + location, data = okc_tr)
rec <- rec %>%
  step_other(diet, location, threshold = 2000, other = "other values")
rec <- prep(rec, training = okc_tr)
tidy(rec, number = 1)

# compare it to
# okc_tr %>% count(diet, sort = TRUE) %>% top_n(4)
# okc_tr %>% count(location, sort = TRUE) %>% top_n(3)

---

**step_pca**

*PCA Signal Extraction*

**Description**

`step_pca` creates a *specification* of a recipe step that will convert numeric data into one or more principal components.

**Usage**

```r
step_pca(
  recipe,
  
  ..., 
  role = "predictor",
  trained = FALSE,
  num_comp = 5,
  threshold = NA,
  options = list(),
  res = NULL,
  prefix = "PC",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("pca")
)
```
## S3 method for class 'step_pca'
tidy(x, type = "coef", ...)

### Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables will be used to compute the components. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**
  For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new principal component columns created by the original variables will be used as predictors in a model.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **num_comp**
  The number of PCA components to retain as new predictors. If `num_comp` is greater than the number of columns or the number of possible components, a smaller value will be used.

- **threshold**
  A fraction of the total variance that should be covered by the components. For example, `threshold = .75` means that `step_pca` should generate enough components to capture 75 percent of the variability in the variables. Note: using this argument will override and reset any value given to `num_comp`.

- **options**
  A list of options to the default method for `stats::prcomp()`. Argument defaults are set to `retx = FALSE`, `center = FALSE`, `scale. = FALSE`, and `tol = NULL`. Note that the argument `x` should not be passed here (or at all).

- **res**
  The `stats::prcomp.default()` object is stored here once this preprocessing step has been trained by `prep.recipe()`.

- **prefix**
  A character string that will be the prefix to the resulting new variables. See notes below.

- **keep_original_cols**
  A logical to keep the original variables in the output. Defaults to `FALSE`.

- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_pca` object.

- **type**
  For the `tidy()` method, either "coef" (for the variable loadings per component) or "variance" (how much variance does each component account for).

### Details

Principal component analysis (PCA) is a transformation of a group of variables that produces a new set of artificial features or components. These components are designed to capture the maximum
amount of information (i.e., variance) in the original variables. Also, the components are statistically
independent from one another. This means that they can be used to combat large inter-variables
correlations in a data set.

It is advisable to standardize the variables prior to running PCA. Here, each variable will be centered
and scaled prior to the PCA calculation. This can be changed using the options argument or by
using \texttt{step_center()} and \texttt{step_scale()}.

The argument \texttt{num\_comp} controls the number of components that will be retained (the original
variables that are used to derive the components are removed from the data). The new components
will have names that begin with prefix and a sequence of numbers. The variable names are padded
with zeros. For example, if \texttt{num\_comp} < 10, their names will be \texttt{PC1} - \texttt{PC9}. If \texttt{num\_comp} = 101, the
names would be \texttt{PC001} - \texttt{PC101}.

Alternatively, \texttt{threshold} can be used to determine the number of components that are required to
capture a specified fraction of the total variance in the variables.

\textbf{Value}

An updated version of \texttt{recipe} with the new step added to the sequence of existing steps (if any).
For the \texttt{tidy} method, a tibble with columns \texttt{terms} (the selectors or variables selected), \texttt{value} (the
loading), and component.

\textbf{References}


\textbf{See Also}

\texttt{step_ica()} \texttt{step_kpca()} \texttt{step_isomap()} \texttt{recipe()} \texttt{prep.recipe()} \texttt{bake.recipe()}

\textbf{Examples}

rec <- recipe(~., data = USArrests)
pca_trans <- rec %>%
  step_normalize(all_numeric()) %>%
  step_pca(all_numeric(), num_comp = 3)
pca_estimates <- prep(pca_trans, training = USArrests)
pca_data <- bake(pca_estimates, USArrests)

rng <- extendrange(c(pca_data$PC1, pca_data$PC2))
plot(pca_data$PC1, pca_data$PC2,
  xlim = rng, ylim = rng)

with_thresh <- rec %>%
  step_normalize(all_numeric()) %>%
  step_pca(all_numeric(), threshold = .99)
with_thresh <- prep(with_thresh, training = USArrests)
bake(with_thresh, USArrests)

tidy(pca_trans, number = 2)
tidy(pca_estimates, number = 2)
Partial Least Squares Feature Extraction

Description

step_pls creates a specification of a recipe step that will convert numeric data into one or more new dimensions.

Usage

step_pls(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  num_comp = 2,
  predictor_prop = 1,
  outcome = NULL,
  options = list(scale = TRUE),
  preserve = deprecated(),
  res = NULL,
  prefix = "PLS",
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("pls")
)

## S3 method for class 'step_pls'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables will be used to compute the dimensions. See selections() for more details. For the tidy method, these are not currently used.
- **role**: For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new dimension columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **num_comp**: The number of pls dimensions to retain as new predictors. If num_comp is greater than the number of columns or the number of possible dimensions, a smaller value will be used.
- **predictor_prop**: The maximum number of original predictors that can have non-zero coefficients for each PLS component (via regularization).
outcome When a single outcome is available, character string or call to `dplyr::vars()` can be used to specify a single outcome variable.

options A list of options to `mixOmics::pls()`, `mixOmics::spls()`, `mixOmics::plsda()`, or `mixOmics::splsda()` (depending on the data and arguments).

preserve Use `keep_original_cols` instead to specify whether the original predictor data should be retained along with the new features.

res A list of results are stored here once this preprocessing step has been trained by `prep.recipe()`.

prefix A character string that will be the prefix to the resulting new variables. See notes below.

keep_original_cols A logical to keep the original variables in the output. Defaults to `FALSE`.

skip A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A `step_pls` object

Details

PLS is a supervised version of principal component analysis that requires the outcome data to compute the new features.

This step requires the Bioconductor `mixOmics` package. If not installed, the step will stop with a note about installing the package.

The argument `num_comp` controls the number of components that will be retained (the original variables that are used to derive the components are removed from the data). The new components will have names that begin with `prefix` and a sequence of numbers. The variable names are padded with zeros. For example, if `num_comp < 10`, their names will be PLS1 - PLS9. If `num_comp = 101`, the names would be PLS001 - PLS101.

Sparsity can be encouraged using the `predictor_prop` parameter. This affects each PLS component, and indicates the maximum proportion of predictors with non-zero coefficients in each component. `step_pls()` converts this proportion to determine the `keepX` parameter in `mixOmics::spls()` and `mixOmics::splsda()`. See the references in `mixOmics::spls()` for details.

The `tidy()` method returns the coefficients that are usually defined as

\[ W(P'W)^{-1} \]

(See the Wikipedia article below)

When applied to data, these values are usually scaled by a column-specific norm. The `tidy()` method applies this same norm to the coefficients shown above.
Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected), components, and values.

References

https://en.wikipedia.org/wiki/Partial_least_squares_regression


See Also

step_pca(), step_kpca(), step_ica(), recipe(), prep.recipe(), bake.recipe()

Examples

# requires the Bioconductor mixOmics package
data(biomass, package = "modeldata")

biom_tr <-
  biomass %>%
  dplyr::filter(dataset == "Training") %>%
  dplyr::select(-dataset,-sample)
biom_te <-
  biomass %>%
  dplyr::filter(dataset == "Testing") %>%
  dplyr::select(-dataset,-sample,-HHV)

dense_pls <-
  recipe(HHV ~ ., data = biom_tr) %>%
  step_pls(all_numeric_predictors(), outcome = "HHV", num_comp = 3)
sparse_pls <-
  recipe(HHV ~ ., data = biom_tr) %>%
  step_pls(all_numeric_predictors(), outcome = "HHV", num_comp = 3, predictor_prop = 4/5)

## PLS discriminant analysis

data(cells, package = "modeldata")

cell_tr <-
  cells %>%
  dplyr::filter(case == "Train") %>%
  dplyr::select(-case)
cell_te <-
  cells %>%
  dplyr::filter(case == "Test") %>%
  dplyr::select(-case,-class)
```r
dense_plsda <- recipe(class ~ ., data = cell_tr) %>%
    step_pls(all_numeric_predictors(), outcome = "class", num_comp = 5)

sparse_plsda <- recipe(class ~ ., data = cell_tr) %>%
    step_pls(all_numeric_predictors(), outcome = "class", num_comp = 5, predictor_prop = 1/4)
```

---

**step_poly**

**Orthogonal Polynomial Basis Functions**

**Description**

`step_poly` creates a specification of a recipe step that will create new columns that are basis expansions of variables using orthogonal polynomials.

**Usage**

```r
step_poly(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  objects = NULL,
  degree = 2,
  options = list(),
  skip = FALSE,
  id = rand_id("poly")
)
```

---

## S3 method for class 'step_poly'

```r
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- `role` For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new columns created from the original variables will be used as predictors in a model.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `objects` A list of `stats::poly()` objects created once the step has been trained.
degree  The polynomial degree (an integer).

options  A list of options for `stats::poly()` which should not include `x`, `degree`, or `simple`. Note that the option `raw = TRUE` will produce the regular polynomial values (not orthogonalized).

skip  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations

id  A character string that is unique to this step to identify it.

x  A `step_poly` object.

Details

`step_poly` can new features from a single variable that enable fitting routines to model this variable in a nonlinear manner. The extent of the possible nonlinearity is determined by the `degree` argument of `stats::poly()`. The original variables are removed from the data and new columns are added. The naming convention for the new variables is `varname_poly_1` and so on.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the columns that will be affected) and `degree`.

See Also

`step_ns()` `recipe()` `prep.recipe()` `bake.recipe()`

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
data = biomass_tr)

quadratic <- rec %>%
  step_poly(carbon, hydrogen)

quadratic <- prep(quadratic, training = biomass_tr)

expanded <- bake(quadratic, biomass_te)
expanded

tidy(quadratic, number = 1)
```
**Description**

`step_profile` creates a *specification* of a recipe step that will fix the levels of all variables but one and will create a sequence of values for the remaining variable. This step can be helpful when creating partial regression plots for additive models.

**Usage**

```r
step_profile(
  recipe, 
  ..., 
  profile = NULL, 
  pct = 0.5, 
  index = 1, 
  grid = list(pctl = TRUE, len = 100), 
  columns = NULL, 
  role = NA, 
  trained = FALSE, 
  skip = FALSE, 
  id = rand_id("profile")
)
```

## S3 method for class 'step_profile'

```r
tidy(x, ...)
```

**Arguments**

- **recipe**
  - A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  - One or more selector functions to choose which variables will be fixed to a single value. See `selections()` for more details. For the tidy method, these are not currently used.

- **profile**
  - A call to `dplyr::vars()` to specify which variable will be profiled (see `selections()`). If a column is included in both lists to be fixed and to be profiled, an error is thrown.

- **pct**
  - A value between 0 and 1 that is the percentile to fix continuous variables. This is applied to all continuous variables captured by the selectors. For date variables, either the minimum, median, or maximum used based on their distance to `pct`.

- **index**
  - The level that qualitative variables will be fixed. If the variables are character (not factors), this will be the index of the sorted unique values. This is applied to all qualitative variables captured by the selectors.
grid A named list with elements pctl (a logical) and len (an integer). If pctl = TRUE, then len denotes how many percentiles to use to create the profiling grid. This creates a grid between 0 and 1 and the profile is determined by the percentiles of the data. For example, if pctl = TRUE and len = 3, the profile would contain the minimum, median, and maximum values. If pctl = FALSE, it defines how many grid points between the minimum and maximum values should be created. This parameter is ignored for qualitative variables (since all of their possible levels are profiled). In the case of date variables, pctl = FALSE will always be used since there is no quantile method for dates.

columns A character string that contains the names of columns that should be fixed and their values. These values are not determined until prep.recipe() is called.

role Not used by this step since no new variables are created.

trained A logical to indicate if the quantities for preprocessing have been estimated.

skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id A character string that is unique to this step to identify it.

x A step_profile object.

Details

This step is atypical in that, when baked, the new_data argument is ignored; the resulting data set is based on the fixed and profiled variable’s information.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (which is the columns that will be affected), and type (fixed or profiled).

Examples

library(modeldata)
data(okc)

# Setup a grid across date but keep the other values fixed
recipe(~ diet + height + date, data = okc) %>%
  step_profile(~date, profile = vars(date)) %>%
  prep(training = okc) %>%
  juice

##########

# An *additive* model; not for use when there are interactions or
# other functional relationships between predictors
lin_mod <- lm(mpg ~ poly(disp, 2) + cyl + hp, data = mtcars)

# Show the difference in the two grid creation methods

disp_pctl <- recipe(~ disp + cyl + hp, data = mtcars) %>%
  step_profile(~disp, profile = vars(disp)) %>%
  prep(training = mtcars)

disp_grid <- recipe(~ disp + cyl + hp, data = mtcars) %>%
  step_profile(~disp,
    profile = vars(disp),
    grid = list(pctl = FALSE, len = 100)
  ) %>%
  prep(training = mtcars)

grid_data <- bake(disp_grid, new_data = NULL)
grid_data <- grid_data %>%
  mutate(pred = predict(lin_mod, grid_data),
         method = "grid")

pctl_data <- bake(disp_pctl, new_data = NULL)
pctl_data <- pctl_data %>%
  mutate(pred = predict(lin_mod, pctl_data),
         method = "percentile")

plot_data <- bind_rows(grid_data, pctl_data)

library(ggplot2)

ggplot(plot_data, aes(x = disp, y = pred)) +
  geom_point(alpha = .5, cex = 1) +
  facet_wrap(~ method)

---

### step_range

**Scaling Numeric Data to a Specific Range**

**Description**

*step_range* creates a *specification* of a recipe step that will normalize numeric data to be within a pre-defined range of values.

**Usage**

```r
step_range(
  recipe,
  ..., 
  role = NA, 
  trained = FALSE,
  min = 0,
  max = 1
)
```
max = 1, 
ranges = NULL, 
skip = FALSE, 
id = rand_id("range")
)

## S3 method for class 'step_range'
tidy(x, ...)

### Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**
  One or more selector functions to choose which variables will be scaled. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**
  Not used by this step since no new variables are created.
- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.
- **min**
  A single numeric value for the smallest value in the range.
- **max**
  A single numeric value for the largest value in the range.
- **ranges**
  A character vector of variables that will be normalized. Note that this is ignored until the values are determined by `prep.recipe()`. Setting this value will be ineffective.
- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**
  A character string that is unique to this step to identify it.
- **x**
  A `step_range` object.

### Details

When a new data point is outside of the ranges seen in the training set, the new values are truncated at `min` or `max`.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected), `min`, and `max`.

### Examples

```r
library(modeldata)
data(biomass)
```
```r
biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
          data = biomass_tr)

ranged_trans <- rec %>%
  step_range(carbon, hydrogen)

ranged_obj <- prep(ranged_trans, training = biomass_tr)

transformed_te <- bake(ranged_obj, biomass_te)

biomass_te[1:10, names(transformed_te)]
transformed_te

tidy(ranged_trans, number = 1)
tidy(ranged_obj, number = 1)
```

---

### `step_ratio`

**Ratio Variable Creation**

**Description**

`step_ratio` creates a *specification* of a recipe step that will create one or more ratios out of numeric variables.

**Usage**

```
step_ratio(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  denom = denom_vars(),
  naming = function(numer, denom) make.names(paste(numer, denom, sep = "_o_")),
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("ratio")
)

denom_vars(...)
```

```
## S3 method for class 'step_ratio'
tidy(x, ...)
```
Arguments

recipe  A recipe object. The step will be added to the sequence of operations for this recipe.

...  One or more selector functions to choose which variables will be used in the numerator of the ratio. When used with denom_vars, the dots indicate which variables are used in the denominator. See selections() for more details. For the tidy method, these are not currently used.

role  For terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the newly created ratios created by the original variables will be used as predictors in a model.

trained  A logical to indicate if the quantities for preprocessing have been estimated.

denom  A call to denom_vars to specify which variables are used in the denominator that can include specific variable names separated by commas or different selectors (see selections()). If a column is included in both lists to be numerator and denominator, it will be removed from the listing.

naming  A function that defines the naming convention for new ratio columns.

columns  The column names used in the ratios. This argument is not populated until prep.recipe() is executed.

keep_original_cols  A logical to keep the original variables in the output. Defaults to TRUE.

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id  A character string that is unique to this step to identify it.

x  A step_ratio object

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and denom.

Examples

library(recipes)
library(modeldata)
data(biomass)

biomass$total <- apply(biomass[, 3:7], 1, sum)
biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur + total, 
data = biomass_tr)
ratio_recipe <- rec %>%
  # all predictors over total
  step_ratio(all_numeric_predictors(), denom = denom_vars(total)) %>%
  # get rid of the original predictors
  step_rm(all_predictors(), -endsWith("total"))

ratio_recipe <- prep(ratio_recipe, training = biomass_tr)

ratio_data <- bake(ratio_recipe, biomass_te)

ratio_data

---

### Create Dummy Variables using Regular Expressions

**Description**

`step_regex` creates a *specification* of a recipe step that will create a new dummy variable based on a regular expression.

**Usage**

```r
step_regex(
  recipe,
  ..., 
  role = "predictor", 
  trained = FALSE,
  pattern = ".",
  options = list(),
  result = make.names(pattern),
  input = NULL,
  skip = FALSE,
  id = rand_id("regex")
)
```

**Arguments**

- `recipe`: A recipe object. The step will be added to the sequence of operations for this recipe.

- `pattern`: A single selector functions to choose which variable will be searched for the pattern. The selector should resolve into a single variable. See `selections()` for more details. For the tidy method, these are not currently used.

- `role`: For a variable created by this step, what analysis role should they be assigned? By default, the function assumes that the new dummy variable column created by the original variable will be used as a predictor in a model.
trained  A logical to indicate if the quantities for preprocessing have been estimated.

pattern  A character string containing a regular expression (or character string for fixed = TRUE) to be matched in the given character vector. Coerced by as.character to a character string if possible.

options  A list of options to grepl() that should not include x or pattern.

result  A single character value for the name of the new variable. It should be a valid column name.

input  A single character value for the name of the variable being searched. This is NULL until computed by prep.recipe().

skip  A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id  A character string that is unique to this step to identify it.

x  A step_regex object.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and result (the new column name).

Examples

library(modeldata)
data(covers)

rec <- recipe(~ description, covers) %>%
  step_regex(description, pattern = "(rock|stony)", result = "rocks") %>%
  step_regex(description, pattern = "ratake families")

rec2 <- prep(rec, training = covers)
rec2

with_dummies <- bake(rec2, new_data = covers)
with_dummies
tidy(rec, number = 1)
tidy(rec2, number = 1)

---

**step_relevel**  Relevel factors to a desired level

**Description**

step_relevel creates a specification of a recipe step that will reorder the provided factor columns so that the level specified by ref_level is first. This is useful for contr.treatment contrasts which take the first level as the reference.
Usage

step_relevel(
  recipe,
  ..., role = NA,
  trained = FALSE,
  ref_level,
  objects = NULL,
  skip = FALSE,
  id = rand_id("relevel")
)

## S3 method for class 'step_relevel'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be affected by the step. These variables should be character or factor types. See selections() for more details.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **ref_level**: A single character value that will be used to relevel the factor column(s) (if the level is present).
- **objects**: A list of objects that contain the information on factor levels that will be determined by prep.recipe().
- **skip**: A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A step_relevel object.

Details

The selected variables are releveled to a level (given by ref_level). Placing the ref_level in the first position.

Note that if the original columns are character, they will be converted to factors by this step.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).
Examples

```r
library(modeldata)
data(okc)
rec <- recipe(~ diet + location, data = okc) %>%
    step_unknown(diet, new_level = "UNKNOWN") %>%
    step_relevel(diet, ref_level = "UNKNOWN") %>%
    prep()

data <- bake(rec, okc)
levels(data$diet)
```

---

**step_relu**

Apply (Smoothed) Rectified Linear Transformation

**Description**

`step_relu` creates a specification of a recipe step that will apply the rectified linear or softplus transformations to numeric data. The transformed data is added as new columns to the data matrix.

**Usage**

```r
step_relu(
    recipe,
    ..., role = "predictor",
    trained = FALSE,
    shift = 0,
    reverse = FALSE,
    smooth = FALSE,
    prefix = "right_relu_",
    columns = NULL,
    skip = FALSE,
    id = rand_id("relu")
)
```

## S3 method for class 'step_relu'

```r
tidy(x, ...)
```

**Arguments**

- **recipe**
  - A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**
  - One or more selector functions to choose which variables are affected by the step. See `selections()` for more details.
- **role**
  - Defaults to "predictor".
- **trained**
  - A logical to indicate if the quantities for preprocessing have been estimated.
**step_relu**

- **shift**: A numeric value dictating a translation to apply to the data.
- **reverse**: A logical to indicate if the left hinge should be used as opposed to the right hinge.
- **smooth**: A logical indicating if the softplus function, a smooth approximation to the rectified linear transformation, should be used.
- **prefix**: A prefix for generated column names, default to "right_relu_" when right hinge transformation and "left_relu_" for reversed/left hinge transformations.
- **columns**: A character string of variable names that will be populated (eventually) by the `terms` argument.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_relu` object.

**Details**

The rectified linear transformation is calculated as

$$\max(0, x - c)$$

and is also known as the ReLu or right hinge function. If `reverse` is true, then the transformation is reflected about the y-axis, like so:

$$\max(0, c - x)$$

Setting the `smooth` option to true will instead calculate a smooth approximation to ReLu according to

$$\ln(1 + e^{x - c})$$

The `reverse` argument may also be applied to this transformation.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any).

**Connection to MARS**

The rectified linear transformation is used in Multivariate Adaptive Regression Splines as a basis function to fit piecewise linear functions to data in a strategy similar to that employed in tree based models. The transformation is a popular choice as an activation function in many neural networks, which could then be seen as a stacked generalization of MARS when making use of ReLu activations. The hinge function also appears in the loss function of Support Vector Machines, where it penalizes residuals only if they are within a certain margin of the decision boundary.

**See Also**

`recipe()` `prep.recipe()` `bake.recipe()`
step_rename

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
data = biomass_tr)

transformed_te <- rec %>%
  step_relu(carbon, shift = 40) %>%
  prep(biomass_tr) %>%
  bake(biomass_te)

transformed_te
```

transformed_te

---

**step_rename**  
*Rename variables by name using dplyr*

**Description**

`step_rename` creates a specification of a recipe step that will add variables using `dplyr::rename()`.

**Usage**

```r
step_rename(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  inputs = NULL,
  skip = FALSE,
  id = rand_id("rename")
)
```

```r
## S3 method for class 'step_rename'
tidy(x, ...)

## S3 method for class 'step_rename_at'
tidy(x, ...)
```

**Arguments**

- `recipe`  
  A recipe object. The step will be added to the sequence of operations for this recipe.
... One or more unquoted expressions separated by commas. See `dplyr::rename()` where the convention is `new_name = old_name`.

role For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new dimension columns created by the original variables will be used as predictors in a model.

trained A logical to indicate if the quantities for preprocessing have been estimated.

inputs Quosure(s) of ....

skip A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

x A step_rename object

Details

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!`) to embed the value of the object in the expression (to be portable between sessions).

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `values` which contains the `rename` expressions as character strings (and are not reparsable).

Examples

```r
recipe(~ ., data = iris) %>%
  step_rename(Sepal_Width = Sepal.Width) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  slice(1:5)

tidy(car_rec, number = 1)
```
step_rename_at

Rename multiple columns using dplyr

Description

`step_rename_at` creates a *specification* of a recipe step that will rename the selected variables using a common function via `dplyr::rename_at()`.

Usage

```r
step_rename_at(
  recipe, ...
  fn,
  role = "predictor",
  trained = FALSE,
  inputs = NULL,
  skip = FALSE,
  id = rand_id("rename_at")
)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
- **fn**: A function `fun`, a quosure style lambda `~ fun(.)`` or a list of either form (but containing only a single function, see `dplyr::rename_at()`). **Note that this argument must be named.**
- **role**: For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new dimension columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **inputs**: A vector of column names populated by `prep()`.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
step_rm

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which contains the columns being transformed.

Examples

```r
library(dplyr)
recipe(~ ., data = iris) %>%
  step_rename_at(everything(), fn = ~ gsub(".", "_", ., fixed = TRUE)) %>%
  prep() %>%
  bake(new_data = NULL) %>%
  slice(1:10)
```

---

step_rm  General Variable Filter

Description

step_rm creates a specification of a recipe step that will remove variables based on their name, type, or role.

Usage

```r
step_rm(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  removals = NULL,
  skip = FALSE,
  id = rand_id("rm")
)
```

```r
## S3 method for class 'step_rm'
tidy(x, ...)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>recipe</td>
<td>A recipe object. The step will be added to the sequence of operations for this recipe.</td>
</tr>
<tr>
<td>...</td>
<td>One or more selector functions to choose which variables that will be evaluated by the filtering bake. See selections() for more details. For the tidy method, these are not currently used.</td>
</tr>
<tr>
<td>role</td>
<td>Not used by this step since no new variables are created.</td>
</tr>
<tr>
<td>trained</td>
<td>A logical to indicate if the quantities for preprocessing have been estimated.</td>
</tr>
</tbody>
</table>
### Description

`step_sample` creates a *specification* of a recipe step that will sample rows using `dplyr::sample_n()` or `dplyr::sample_frac()`.

---

#### step_sample

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>removals</code></td>
<td>A character string that contains the names of columns that should be removed. These values are not determined until <code>prep.recipe()</code> is called.</td>
</tr>
<tr>
<td><code>skip</code></td>
<td>A logical. Should the step be skipped when the recipe is baked by <code>bake.recipe()</code>? While all operations are baked when <code>prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = TRUE</code> as it may affect the computations for subsequent operations.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>A character string that is unique to this step to identify it.</td>
</tr>
<tr>
<td><code>x</code></td>
<td>A <code>step_rm</code> object.</td>
</tr>
</tbody>
</table>

#### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the columns that will be removed.

#### Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training", ]
biomass_te <- biomass[biomass$dataset == "Testing", ]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, data = biomass_tr)

library(dplyr)
smaller_set <- rec %>%
  step_rm(contains("gen"))

smaller_set <- prep(smaller_set, training = biomass_tr)

filtered_te <- bake(smaller_set, biomass_te)
filtered_te

tidy(smaller_set, number = 1)
```
**Usage**

```r
step_sample(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  size = NULL,
  replace = FALSE,
  skip = TRUE,
  id = rand_id("sample")
)
```

```r
## S3 method for class 'step_sample'
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` Argument ignored; included for consistency with other step specification functions. For the tidy method, these are not currently used.
- `role` Not used by this step since no new variables are created.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `size` An integer or fraction. If the value is within (0, 1), `dplyr::sample_frac()` is applied to the data. If an integer value of 1 or greater is used, `dplyr::sample_n()` is applied. The default of `NULL` uses `dplyr::sample_n()` with the size of the training set (or smaller for smaller `new_data`).
- `replace` Sample with or without replacement?
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?
  While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = FALSE`.
- `id` A character string that is unique to this step to identify it.
- `x` A step_sample object

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `size`, `replace`, and `id`.

**Row Filtering**

This step can entirely remove observations (rows of data), which can have unintended and/or problematic consequences when applying the step to new data later via `bake.recipe()`. Consider whether `skip = TRUE` or `skip = FALSE` is more appropriate in any given use case. In most instances that affect the rows of the data being predicted, this step probably should not be applied at all; instead, execute operations like this outside and before starting a preprocessing `recipe()`.
See Also

`step_filter()` `step_naomit()` `step_slice()`

Examples

```r
# Uses `sample_n`
recipe(~ ., data = mtcars) %>%
  step_sample(size = 1) %>%
  prep(training = mtcars) %>%
  bake(new_data = NULL) %>%
  nrow()

# Uses `sample_frac`
recipe(~ ., data = mtcars) %>%
  step_sample(size = 0.9999) %>%
  prep(training = mtcars) %>%
  bake(new_data = NULL) %>%
  nrow()

# Uses `sample_n` and returns _at maximum_ 20 samples.
smaller_cars <-
  recipe(~ ., data = mtcars) %>%
  step_sample() %>%
  prep(training = mtcars %>% slice(1:20))

bake(smaller_cars, new_data = NULL) %>%
  nrow()

bake(smaller_cars, new_data = mtcars %>% slice(21:32)) %>%
  nrow()
```

---

**step_scale**

*Scaling Numeric Data*

**Description**

`step_scale` creates a *specification* of a recipe step that will normalize numeric data to have a standard deviation of one.

**Usage**

```r
step_scale(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  sds = NULL,
  factor = 1,
  na.rm = TRUE,
  skip = FALSE,
  id = rand_id("scale")
)```
## S3 method for class 'step_scale'
tidy(x, ...)

### Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **sds**
  A named numeric vector of standard deviations. This is **NULL** until computed by `prep.recipe()`.

- **factor**
  A numeric value of either 1 or 2 that scales the numeric inputs by one or two standard deviations. By dividing by two standard deviations, the coefficients attached to continuous predictors can be interpreted the same way as with binary inputs. Defaults to 1. More in reference below.

- **na_rm**
  A logical value indicating whether NA values should be removed when computing the standard deviation.

- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_scale` object.

### Details

Scaling data means that the standard deviation of a variable is divided out of the data. `step_scale` estimates the variable standard deviations from the data used in the `training` argument of `prep.recipe`. `bake.recipe` then applies the scaling to new data sets using these standard deviations.

### Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` (the selectors or variables selected) and `value` (the standard deviations).

### References

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, 
              data = biomass_tr)

scaled_trans <- rec %>%
  step_scale(carbon, hydrogen)

scaled_obj <- prep(scaled_trans, training = biomass_tr)

transformed_te <- bake(scaled_obj, biomass_te)

biomass_te[1:10, names(transformed_te)]
transformed_te
tidy(scaled_trans, number = 1)
tidy(scaled_obj, number = 1)
```

---

**step_select**

*Select variables using dplyr*

**Description**

`step_select()` creates a *specification* of a recipe step that will select variables using `dplyr::select()`.

**Usage**

```r
step_select(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  skip = FALSE,
  id = rand_id("select")
)
```

```r
## S3 method for class 'step_select'
tidy(x, ...)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
... One or more selector functions to choose which variables will be selected when baking. See `selections()` for more details. For the tidy method, these are not currently used.

**role** For model terms selected by this step, what analysis role should they be assigned?

**trained** A logical to indicate if the quantities for preprocessing have been estimated.

**skip** A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

**id** A character string that is unique to this step to identify it.

**x** A `step_select` object

**Details**

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!`) to embed the value of the object in the expression (to be portable between sessions). See the examples.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with column `terms` which contains the select expressions as character strings (and are not reparsable).

**Examples**

```r
library(dplyr)

iris_tbl <- as_tibble(iris)
iris_train <- slice(iris_tbl, 1:75)
iris_test <- slice(iris_tbl, 76:150)

dplyr_train <- select(iris_train, Species, starts_with("Sepal"))
dplyr_test <- select(iris_test, Species, starts_with("Sepal"))

rec <- recipe(~., data = iris_train) %>%
  step_select(Species, starts_with("Sepal")) %>%
  prep(training = iris_train)

rec_train <- bake(rec, new_data = NULL)
all.equal(dplyr_train, rec_train)

rec_test <- bake(rec, iris_test)
all.equal(dplyr_test, rec_test)

# Local variables
sepal_vars <- c("Sepal.Width", "Sepal.Length")
```
qq_rec <-
  recipe(~., data = iris_train) %>%
  # fine for interactive usage
  step_select(Species, all_of(sepal_vars)) %>%
  # best approach for saving a recipe to disk
  step_select(Species, all_of(!sepal_vars))

  # Note that `sepal_vars` is inlined in the second approach
qq_rec

---

**step_shuffle**

**Shuffle Variables**

**Description**

`step_shuffle` creates a specification of a recipe step that will randomly change the order of rows for selected variables.

**Usage**

```r
step_shuffle(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("shuffle")
)
```

```r
## S3 method for class 'step_shuffle'
tidy(x, ...)
```

**Arguments**

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables will be permuted. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **columns**
  A character string that contains the names of columns that should be shuffled. These values are not determined until `prep.recipe()` is called.
skip

A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`?

While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id

A character string that is unique to this step to identify it.

x

A `step_shuffle` object.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` which is the columns that will be affected.

Examples

```r
integers <- data.frame(A = 1:12, B = 13:24, C = 25:36)

library(dplyr)
rec <- recipe(~ A + B + C, data = integers) %>%
  step_shuffle(A, B)
rand_set <- prep(rec, training = integers)
set.seed(5377)
bake(rand_set, integers)
tidy(rec, number = 1)
tidy(rand_set, number = 1)
```

---

**step_slice**  
Filter rows by position using `dplyr`

Description

`step_slice` creates a specification of a recipe step that will filter rows using `dplyr::slice()`.

Usage

```r
step_slice(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  inputs = NULL,
  skip = TRUE,
  id = rand_id("slice")
)
```

```r
## S3 method for class 'step_slice'
tidy(x, ...)
```
Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: Integer row values. See `dplyr::slice()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **inputs**: Quosure of values given by ....
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = FALSE`.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_slice` object

Details

When an object in the user’s global environment is referenced in the expression defining the new variable(s), it is a good idea to use quasiquotation (e.g. `!!`) to embed the value of the object in the expression (to be portable between sessions). See the examples.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` which contains the filtering indices.

Row Filtering

This step can entirely remove observations (rows of data), which can have unintended and/or problematic consequences when applying the step to new data later via `bake.recipe()`. Consider whether `skip = TRUE` or `skip = FALSE` is more appropriate in any given use case. In most instances that affect the rows of the data being predicted, this step probably should not be applied at all; instead, execute operations like this outside and before starting a preprocessing `recipe()`.

See Also

- `step_filter()`
- `step_naomit()`
- `step_sample()`

Examples

```r
rec <- recipe(~ ., data = iris) %>%
  step_slice(1:3)
prepped <- prep(rec, training = iris %>% slice(1:75))
tidy(prepped, number = 1)
library(dplyr)
```
```r
dplyr_train <-
  iris %>%
  as_tibble() %>%
  slice(1:75) %>%
  slice(1:3)

rec_train <- bake(prepped, new_data = NULL)
all.equal(dplyr_train, rec_train)

dplyr_test <-
  iris %>%
  as_tibble() %>%
  slice(76:150) %>%
  slice(1:3)
rec_test <- bake(prepped, iris %>% slice(76:150))
all.equal(dplyr_test, rec_test)

# Embedding the integer expression (or vector) into the
# recipe:
keep_rows <- 1:6

qq_rec <-
  recipe(~ ., data = iris) %>%
  # Embed `keep_rows` in the call using `!!`
  step_slice(!!keep_rows) %>%
  prep(training = iris)

tidy(qq_rec, number = 1)
```

---

**step_spatialsign**  
*Spatial Sign Preprocessing*

**Description**

`step_spatialsign` is a specification of a recipe step that will convert numeric data into a projection on to a unit sphere.

**Usage**

```r
step_spatialsign(
  recipe,
  ..., 
  role = "predictor",
  na_rm = TRUE,
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
```
id = rand_id("spatialsign")
)

## S3 method for class 'step_spatialsign'
tidy(x, ...)

Arguments

- recipe: A recipe object. The step will be added to the sequence of operations for this recipe.
- ...: One or more selector functions to choose which variables will be used for the normalization. See selections() for more details. For the tidy method, these are not currently used.
- role: For model terms created by this step, what analysis role should they be assigned?
- na.rm: A logical: should missing data be removed from the norm computation?
- trained: A logical to indicate if the quantities for preprocessing have been estimated.
- columns: A character string of variable names that will be populated (eventually) by the terms argument.
- skip: A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
- id: A character string that is unique to this step to identify it.
- x: A step_spatialsign object.

Details

The spatial sign transformation projects the variables onto a unit sphere and is related to global contrast normalization. The spatial sign of a vector \( \mathbf{w} \) is \( \mathbf{w}/\|\mathbf{w}\| \).

The variables should be centered and scaled prior to the computations.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the columns that will be affected.

References

Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, 
              data = biomass_tr)

ss_trans <- rec %>% 
  step_center(carbon, hydrogen) %>% 
  step_scale(carbon, hydrogen) %>% 
  step_spatialsign(carbon, hydrogen)

ss_obj <- prep(ss_trans, training = biomass_tr)

transformed_te <- bake(ss_obj, biomass_te)

plot(biomass_te$carbon, biomass_te$hydrogen)
plot(transformed_te$carbon, transformed_te$hydrogen)
tidy(ss_trans, number = 3)
tidy(ss_obj, number = 3)
```

---

**step_sqrt**

Square Root Transformation

**Description**

`step_sqrt` creates a specification of a recipe step that will square root transform the data.

**Usage**

```r
step_sqrt(
  recipe,
  ..., 
  role = NA, 
  trained = FALSE,
  columns = NULL, 
  skip = FALSE,
  id = rand_id("sqrt")
)
```

## S3 method for class 'step_sqrt'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables will be transformed. See `selections()` for more details. For the tidy method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **columns**: A character string of variable names that will be populated (eventually) by the terms argument.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_sqrt` object.

Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns `terms` which is the columns that will be affected.

See Also

- `step_logit()`
- `step_invlogit()`
- `step_log()`
- `step_hyperbolic()`
- `recipe()`
- `prep.recipe()`
- `bake.recipe()`

Examples

```r
set.seed(313)
examples <- matrix(rnorm(40)^2, ncol = 2)
examples <- as.data.frame(examples)

rec <- recipe(~ V1 + V2, data = examples)

sqrt_trans <- rec %>%
  step_sqrt(all_numeric_predictors())

sqrt_obj <- prep(sqrt_trans, training = examples)

transformed_te <- bake(sqrt_obj, examples)
plot(examples$V1, transformed_te$V1)

tidy(sqrt_trans, number = 1)
tidy(sqrt_obj, number = 1)
```
step_string2factor  Convert Strings to Factors

Description

step_string2factor will convert one or more character vectors to factors (ordered or unordered).

Usage

```r
step_string2factor(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  levels = NULL,
  ordered = FALSE,
  skip = FALSE,
  id = rand_id("string2factor")
)
```

## S3 method for class 'step_string2factor'
tidy(x, ...)

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables will be converted to factors. See `selections()` for more details. For the `tidy` method, these are not currently used.
- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **levels**: An options specification of the levels to be used for the new factor. If left `NULL`, the sorted unique values present when `bake` is called will be used.
- **ordered**: A single logical value; should the factor(s) be ordered?
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_string2factor` object.
**step_unknown**

Assign missing categories to "unknown"

**Description**

step_unknown creates a specification of a recipe step that will assign a missing value in a factor level to "unknown".

**Usage**

```r
step_unknown(
  recipe,
  ..., 
  role = NA,
)```

**Details**

If levels is given, step_string2factor will convert all variables affected by this step to have the same levels.

Also, note that prep has an option strings_as_factors that defaults to TRUE. This should be changed so that raw character data will be applied to step_string2factor. However, this step can also take existing factors (but will leave them as-is).

**Value**

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and ordered.

**See Also**

step_factor2string() step_dummy() step_other() step_novel()

**Examples**

```r
library(modeldata)
data(okc)

rec <- recipe(~ diet + location, data = okc)

make_factor <- rec %>%
  step_string2factor(diet)
make_factor <- prep(make_factor, 
  training = okc, 
  strings_as_factors = FALSE)

# note that `diet` is a factor
bake(make_factor, new_data = NULL) %>% head
okc %>% head
tidy(make_factor, number = 1)
```
## Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables that will be affected by the step. These variables should be character or factor types. See `selections()` for more details. For the `tidy` method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **new_level**
  A single character value that will be assigned to new factor levels.

- **objects**
  A list of objects that contain the information on factor levels that will be determined by `prep.recipe()`.

- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_unknown` object.

## Details

The selected variables are adjusted to have a new level (given by `new_level`) that is placed in the last position.

Note that if the original columns are character, they will be converted to factors by this step.

If `new_level` is already in the data given to `prep`, an error is thrown.

## Value

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the columns that will be affected) and `value` (the factor levels that is used for the new value)

## See Also

- `step_factor2string()`, `step_string2factor()`, `dummy_names()`, `step_regex()`, `step_count()`, `step_ordinalscore()`, `step_unorder()`, `step_other()`, `step_novel()`
Examples

library(modeldata)
data(okc)

rec <-
  recipe(~ diet + location, data = okc) %>%
  step_unknown(diet, new_level = "unknown diet") %>%
  step_unknown(location, new_level = "unknown location") %>%
  prep()

table(bake(rec, new_data = NULL) %>% pull(diet),
      okc %>% pull(diet),
      useNA = "always") %>%
      as.data.frame() %>%
      dplyr::filter(Freq > 0)

tidy(rec, number = 1)

---

**step_unorder**

*Convert Ordered Factors to Unordered Factors*

Description

*step_unorder* creates a specification of a recipe step that will transform the data.

Usage

```r
step_unorder(
    recipe,
    ..., 
    role = NA,
    trained = FALSE,
    columns = NULL,
    skip = FALSE,
    id = rand_id("unorder")
)
```

## S3 method for class 'step_unorder'
tidy(x, ...)

Arguments

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.
**step_unorder**

- **role**: Not used by this step since no new variables are created.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **columns**: A character string of variable names that will be populated (eventually) by the `terms` argument.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.
- **x**: A `step_unorder` object.

**Details**

The factors level order is preserved during the transformation.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the columns that will be affected).

**See Also**

- `step_ordinalscore()`
- `recipe()`
- `prep.recipe()`
- `bake.recipe()`

**Examples**

```r
lmh <- c("Low", "Med", "High")

examples <- data.frame(X1 = factor(rep(letters[1:4], each = 3)),
                        X2 = ordered(rep(lmh, each = 4),
                                    levels = lmh))

rec <- recipe(~ X1 + X2, data = examples)

factor_trans <- rec %>%
  step_unorder(all_nominal_predictors())

factor_obj <- prep(factor_trans, training = examples)

transformed_te <- bake(factor_obj, examples)

table(transformed_te$X2, examples$X2)

tidy(factor_trans, number = 1)

tidy(factor_obj, number = 1)
```
Description

step_window creates a specification of a recipe step that will create new columns that are the results of functions that compute statistics across moving windows.

Usage

step_window(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  size = 3,
  na_rm = TRUE,
  statistic = "mean",
  columns = NULL,
  names = NULL,
  skip = FALSE,
  id = rand_id("window")
)

## S3 method for class 'step_window'
tidy(x, ...)

Arguments

**recipe**
A recipe object. The step will be added to the sequence of operations for this recipe.

**...**
One or more selector functions to choose which variables are affected by the step. See selections() for more details. For the tidy method, these are not currently used.

**role**
For model terms created by this step, what analysis role should they be assigned? If names is left to be NULL, the rolling statistics replace the original columns and the roles are left unchanged. If names is set, those new columns will have a role of NULL unless this argument has a value.

**trained**
A logical to indicate if the quantities for preprocessing have been estimated.

**size**
An odd integer >= 3 for the window size.

**na_rm**
A logical for whether missing values should be removed from the calculations within each window.

**statistic**
A character string for the type of statistic that should be calculated for each moving window. Possible values are: 'max', 'mean', 'median', 'min', 'prod', 'sd', 'sum', 'var'
**columns**  
A character string that contains the names of columns that should be processed. These values are not determined until `prep.recipe()` is called.

**names**  
An optional character string that is the same length of the number of terms selected by `terms`. If you are not sure what columns will be selected, use the `summary` function (see the example below). These will be the names of the new columns created by the step.

**skip**  
A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

**id**  
A character string that is unique to this step to identify it.

**x**  
A `step_window` object.

**Details**

The calculations use a somewhat atypical method for handling the beginning and end parts of the rolling statistics. The process starts with the center justified window calculations and the beginning and ending parts of the rolling values are determined using the first and last rolling values, respectively. For example, if a column `x` with 12 values is smoothed with a 5-point moving median, the first three smoothed values are estimated by `median(x[1:5])` and the fourth uses `median(x[2:6])`. The step will stop with a note about installing the package.

**Value**

An updated version of `recipe` with the new step added to the sequence of existing steps (if any). For the `tidy` method, a tibble with columns `terms` (the selectors or variables selected) and `statistic` (the summary function name), and `size`.

**Examples**

```r
code here
```

names = paste0("mean_3pt_", 1:2),
role = "outcome")
rec <- prep(rec, training = sim_dat)

# If you aren't sure how to set the names, see which variables are selected
# and the order that they are selected:
terms_select(info = summary(rec), terms = quos(starts_with("y")))

smoothed_dat <- bake(rec, sim_dat, everything())

ggplot(data = sim_dat, aes(x = x1, y = y1)) +
  geom_point() +
  geom_line(data = smoothed_dat, aes(y = med_7pt_1)) +
  geom_line(data = smoothed_dat, aes(y = mean_3pt_1), col = "red") +
  theme_bw()

tidy(rec, number = 1)
tidy(rec, number = 2)

# If you want to replace the selected variables with the rolling statistic
# don't set `names`
sim_dat$original <- sim_dat$y1
rec <- recipe(y1 + y2 + original ~ x1 + x2 + x3, data = sim_dat) %>%
  step_window(starts_with("y"))
rec <- prep(rec, training = sim_dat)
smoothed_dat <- bake(rec, sim_dat, everything())

ggplot(smoothed_dat, aes(x = original, y = y1)) +
  geom_point() +
  theme_bw()

---

**step_YeoJohnson**

### Yeo-Johnson Transformation

**Description**

step_YeoJohnson creates a specification of a recipe step that will transform data using a simple Yeo-Johnson transformation.

**Usage**

```r
step_YeoJohnson(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  lambdas = NULL,
  limits = c(-5, 5),
  num_unique = 5,
  na_rm = TRUE,
)```
skip = FALSE,
id = rand_id("YeoJohnson")
)

## S3 method for class 'step_YeoJohnson'
tidy(x, ...)

Arguments

- **recipe**
  A recipe object. The step will be added to the sequence of operations for this recipe.

- **...**
  One or more selector functions to choose which variables are affected by the step. See `selections()` for more details. For the tidy method, these are not currently used.

- **role**
  Not used by this step since no new variables are created.

- **trained**
  A logical to indicate if the quantities for preprocessing have been estimated.

- **lambdas**
  A numeric vector of transformation values. This is `NULL` until computed by `prep.recipe()`.

- **limits**
  A length 2 numeric vector defining the range to compute the transformation parameter lambda.

- **num_unique**
  An integer where data that have less possible values will not be evaluated for a transformation.

- **na_rm**
  A logical value indicating whether `NA` values should be removed during computations.

- **skip**
  A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

- **id**
  A character string that is unique to this step to identify it.

- **x**
  A `step_YeoJohnson` object.

Details

The Yeo-Johnson transformation is very similar to the Box-Cox but does not require the input variables to be strictly positive. In the package, the partial log-likelihood function is directly optimized within a reasonable set of transformation values (which can be changed by the user).

This transformation is typically done on the outcome variable using the residuals for a statistical model (such as ordinary least squares). Here, a simple null model (intercept only) is used to apply the transformation to the predictor variables individually. This can have the effect of making the variable distributions more symmetric.

If the transformation parameters are estimated to be very close to the bounds, or if the optimization fails, a value of `NA` is used and no transformation is applied.
step_zv

Zero Variance Filter

Description

step_zv creates a specification of a recipe step that will remove variables that contain only a single value.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms (the selectors or variables selected) and value (the lambda estimate).

References


See Also

step_BoxCox() recipe() prep.recipe() bake.recipe()

Examples

library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

rec <- recipe(HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur, data = biomass_tr)
yj_transform <- step_YeoJohnson(rec, all_numeric())
yj_estimates <- prep(yj_transform, training = biomass_tr)
yj_te <- bake(yj_estimates, biomass_te)

plot(density(biomass_te$sulfur), main = "before")
plot(density(yj_te$sulfur), main = "after")
tidy(yj_transform, number = 1)
tidy(yj_estimates, number = 1)
Usage

step_zv(
  recipe,
  ..., 
  role = NA,
  trained = FALSE,
  removals = NULL,
  skip = FALSE,
  id = rand_id("zv")
)

## S3 method for class 'step_zv'
tidy(x, ...)

Arguments

recipe 
A recipe object. The step will be added to the sequence of operations for this recipe.

... 
One or more selector functions to choose which variables that will be evaluated by the filtering. See selections() for more details. For the tidy method, these are not currently used.

role 
Not used by this step since no new variables are created.

trained 
A logical to indicate if the quantities for preprocessing have been estimated.

removals 
A character string that contains the names of columns that should be removed. These values are not determined until prep.recipe() is called.

skip 
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations

id 
A character string that is unique to this step to identify it.

x 
A step_zv object.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any). For the tidy method, a tibble with columns terms which is the columns that will be removed.

See Also

step_nzv() step_corr() recipe() prep.recipe() bake.recipe()

Examples

library(modeldata)
data(biomass)
biomass$one_value <- 1
biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]
rec <- recipe(HHV ~ carbon + hydrogen + oxygen +
              nitrogen + sulfur + one_value,
data = biomass_tr)
zv_filter <- rec %>%
    step_zv(all_predictors())
filter_obj <- prep(zv_filter, training = biomass_tr)
filtered_te <- bake(filter_obj, biomass_te)
any(names(filtered_te) == "one_value")
tidy(zv_filter, number = 1)
tidy(filter_obj, number = 1)

summary.recipe

Description

This function prints the current set of variables/features and some of their characteristics.

Usage

## S3 method for class 'recipe'
summary(object, original = FALSE, ...)

Arguments

object A recipe object
original A logical: show the current set of variables or the original set when the recipe
        was defined.
... further arguments passed to or from other methods (not currently used).

Details

Note that, until the recipe has been trained, the current and original variables are the same.
It is possible for variables to have multiple roles by adding them with add_role(). If a variable has
multiple roles, it will have more than one row in the summary tibble.

Value

A tibble with columns variable, type, role, and source.
terms_select  Select Terms in a Step Function.

Description

This function bakes the step function selectors and might be useful when creating custom steps.

Usage

terms_select(terms, info, empty_fun = abort_selection)

Arguments

- **terms**: A list of formulas whose right-hand side contains quoted expressions. See `rlang::quos()` for examples.
- **info**: A tibble with columns `variable`, `type`, `role`, and `source` that represent the current state of the data. The function `summary.recipe()` can be used to get this information from a recipe.
- **empty_fun**: A function to execute when no terms are selected by the step. The default function throws an error with a message.

Value

A character string of column names or an error if there are no selectors or if no variables are selected.

See Also

`recipe()` `summary.recipe()` `prep.recipe()`
Examples

```r
library(rlang)
library(modeldata)
data(okc)
rec <- recipe(~ ., data = okc)
info <- summary(rec)
terms_select(info = info, quos(all_predictors()))
```

tidy.recipe  

Tidy the Result of a Recipe

Description

tidy will return a data frame that contains information regarding a recipe or operation within the recipe (when a tidy method for the operation exists).

Usage

```r
## S3 method for class 'recipe'
tidy(x, number = NA, id = NA, ...)

## S3 method for class 'step'
tidy(x, ...)

## S3 method for class 'check'
tidy(x, ...)
```

Arguments

- `x` A recipe object (trained or otherwise).
- `number` An integer or NA. If missing and `id` is not provided, the return value is a list of the operations in the recipe. If a number is given, a tidy method is executed for that operation in the recipe (if it exists). `number` must not be provided if `id` is.
- `id` A character string or NA. If missing and `number` is not provided, the return value is a list of the operations in the recipe. If a character string is given, a tidy method is executed for that operation in the recipe (if it exists). `id` must not be provided if `number` is.
- `...` Not currently used.

Value

A tibble with columns that would vary depending on what tidy method is executed. When `number` and `id` are NA, a tibble with columns `number` (the operation iteration), `operation` (either "step" or "check"), `type` (the method, e.g. "nzv", "center"), a logical column called `trained` for whether the operation has been estimated using `prep`, a logical for `skip`, and a character column `id`. 
**Examples**

```r
library(modeldata)
data(okc)

okc_rec <- recipe(~ ., data = okc) %>%
  step_other(all_nominal(), threshold = 0.05, other = "another") %>%
  step_date(date, features = "dow") %>%
  step_center(all_numeric()) %>%
  step_dummy(all_nominal()) %>%
  check_cols(starts_with("date"), age, height)

tidy(okc_rec)
tidy(okc_rec, number = 2)
tidy(okc_rec, number = 3)

okc_rec_trained <- prep(okc_rec, training = okc)

tidy(okc_rec_trained)
tidy(okc_rec_trained, number = 3)
```

---

**update.step**

*Update a recipe step*

**Description**

This step method for `update()` takes named arguments as ... who's values will replace the elements of the same name in the actual step.

**Usage**

```r
## S3 method for class 'step'
update(object, ...)
```

**Arguments**

- `object` A recipe step.
- `...` Key-value pairs where the keys match up with names of elements in the step, and the values are the new values to update the step with.

**Details**

For a step to be updated, it must not already have been trained. Otherwise, conflicting information can arise between the data returned from `bake(object, new_data = NULL)` and the information in the step.
Examples

```r
library(modeldata)
data(biomass)

biomass_tr <- biomass[biomass$dataset == "Training",]
biomass_te <- biomass[biomass$dataset == "Testing",]

# Create a recipe using step_bs() with degree = 3
rec <- recipe(
  HHV ~ carbon + hydrogen + oxygen + nitrogen + sulfur,
  data = biomass_tr
) %>%
  step_bs(carbon, hydrogen, degree = 3)

# Update the step to use degree = 4
rec2 <- rec
rec2$steps[[1]] <- update(rec2$steps[[1]], degree = 4)

# Prep both recipes
rec_prepped <- prep(rec, training = biomass_tr)
rec2_prepped <- prep(rec2, training = biomass_tr)

# Juice both to see what changed
bake(rec_prepped, new_data = NULL)
bake(rec2_prepped, new_data = NULL)

# Cannot update a recipe step that has been trained!
## Not run:
update(rec_prepped$steps[[1]], degree = 4)

## End(Not run)
```
Index

* basis_expansion
  step_bs, 38
  step_kpca, 101
  step_kpca_poly, 103
  step_kpca_rbf, 106
  step_ns, 126
  step_poly, 143

* datagen
  add_step, 4
  bake, 4
  check_class, 6
  check_range, 13
  discretize, 16
  has_role, 19
  names0, 21
  prep, 22
  recipe, 25
  roles, 29
  step_arrange, 32
  step_bin2factor, 34
  step_BoxCox, 36
  step_bs, 38
  step_center, 40
  step_classdist, 42
  step_corr, 44
  step_count, 46
  step_cut, 48
  step_date, 50
  step_depth, 52
  step_dummy, 55
  step_factor2string, 58
  step_filter, 60
  step_geodist, 62
  step_holiday, 64
  step_hyperbolic, 65
  step_ica, 67
  step_impute_bag, 69
  step_impute_knn, 72
  step_impute_linear, 75
  step_impute_lower, 77
  step_impute_mean, 79
  step_impute_median, 82
  step_impute_mode, 84
  step_impute_roll, 86
  step_indicate_na, 88
  step_integer, 89
  step_interact, 91
  step_inverse, 95
  step_invlogit, 97
  step_isomap, 98
  step_kpca, 101
  step_kpca_poly, 103
  step_kpca_rbf, 106
  step_lincomb, 110
  step_log, 112
  step_logit, 114
  step_mutate, 115
  step_mutate_at, 117
  step_nnmf, 120
  step_normalize, 122
  step_novel, 124
  step_ns, 126
  step_num2factor, 128
  step_nzv, 130
  step_ordinalscore, 133
  step_other, 135
  step_pca, 137
  step_pls, 140
  step_poly, 143
  step_profile, 145
  step_range, 147
  step_ratio, 149
  step_regex, 151
  step_relevel, 152
  step_rename, 156
  step_rename_at, 158
  step_rm, 159
  step_sample, 160
step_scale, 162
step_select, 164
step_shuffle, 166
step_slice, 167
stepSpatialSign, 169
step_sqrt, 171
step_string2factor, 173
step_unknown, 174
step_unorder, 176
step_window, 178
step_YeoJohnson, 180
step_zv, 182
terms_select, 185

* dates
step_date, 50
step_holiday, 64

* dimension_reduction
step_classdist, 42
step_depth, 52

* discretization
discretize, 16

* dummy_variables
step_bin2factor, 34
step_count, 46
step_dummy, 55
step_regex, 151

* factors
discretize, 16
step_bin2factor, 34
step_factor2string, 58
step_novel, 124
step_num2factor, 128
step_other, 135
step_relevel, 152
step_string2factor, 173
step_unknown, 174

* ica
step_ica, 67

* imputation
step_impute_bag, 69
step_impute_knn, 72
step_impute_linear, 75
step_impute_lower, 77
step_impute_mean, 79
step_impute_median, 82
step_impute_mode, 84
step_impute_roll, 86
step_indicate_na, 88

* isomap
step_isomap, 98

* kernel_methods
step_kpca, 101
step_kpca_poly, 103
step_kpca_rbf, 106

* model_specification
bake, 4
prep, 22
recipe, 25
roles, 29
step_date, 50
step_dummy, 55
step_holiday, 64
step_interact, 91

* moving_windows
step_window, 178

* naming_functions
names0, 21

* nnmf
step_nnmf, 120

* normalization_methods
check_range, 13
step_center, 40
step_normalize, 122
step_range, 147
step_scale, 162

* ordinal_data
step_ordinalscore, 133
step_unorder, 176

* pca
step_kpca, 101
step_kpca_poly, 103
step_kpca_rbf, 106
step_pca, 137

* permutation
step_shuffle, 166

* pls
step_pls, 140

* preprocessing normalization_methods
check_class, 6

* preprocessing
add_step, 4
bake, 4
check_range, 13
discretize, 16
prep, 22
recipe, 25
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>step_invlogit</td>
<td>97</td>
</tr>
<tr>
<td>step_log</td>
<td>95</td>
</tr>
<tr>
<td>step_logit</td>
<td>112</td>
</tr>
<tr>
<td>step_logit</td>
<td>114</td>
</tr>
<tr>
<td>step_mutate</td>
<td>115</td>
</tr>
<tr>
<td>step_mutate_at</td>
<td>117</td>
</tr>
<tr>
<td>step_rename</td>
<td>156</td>
</tr>
<tr>
<td>step_rename_at</td>
<td>158</td>
</tr>
<tr>
<td>step_sqrt</td>
<td>171</td>
</tr>
<tr>
<td>step_YeoJohnson</td>
<td>180</td>
</tr>
<tr>
<td>step_date</td>
<td>50</td>
</tr>
<tr>
<td>step_dummy</td>
<td>55</td>
</tr>
<tr>
<td>step_factor2string</td>
<td>58</td>
</tr>
<tr>
<td>step_holiday</td>
<td>64</td>
</tr>
<tr>
<td>step_integer</td>
<td>89</td>
</tr>
<tr>
<td>step_num2factor</td>
<td>128</td>
</tr>
<tr>
<td>step_string2factor</td>
<td>173</td>
</tr>
<tr>
<td>step_corr</td>
<td>44</td>
</tr>
<tr>
<td>step_lincomb</td>
<td>110</td>
</tr>
<tr>
<td>step_nzv</td>
<td>130</td>
</tr>
<tr>
<td>step_rm</td>
<td>159</td>
</tr>
<tr>
<td>step_select</td>
<td>164</td>
</tr>
<tr>
<td>step_zv</td>
<td>182</td>
</tr>
<tr>
<td>add_check</td>
<td>4</td>
</tr>
<tr>
<td>add_role (roles)</td>
<td>29</td>
</tr>
<tr>
<td>add_role</td>
<td>184</td>
</tr>
<tr>
<td>add_step</td>
<td>4</td>
</tr>
<tr>
<td>all_nominal</td>
<td>19</td>
</tr>
<tr>
<td>all_nominal (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>all_nominal ()</td>
<td>32</td>
</tr>
<tr>
<td>all_nominal_predictors</td>
<td>19</td>
</tr>
<tr>
<td>all_nominal_predictors (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>all_numeric</td>
<td>19</td>
</tr>
<tr>
<td>all_numeric (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>all_numeric ()</td>
<td>32</td>
</tr>
<tr>
<td>all_numeric_predictors</td>
<td>19</td>
</tr>
<tr>
<td>all_numeric_predictors (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>all_outcomes</td>
<td>19</td>
</tr>
<tr>
<td>all_outcomes ()</td>
<td>32</td>
</tr>
<tr>
<td>all_predictors</td>
<td>19</td>
</tr>
<tr>
<td>all_predictors (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>all_predictors ()</td>
<td>32</td>
</tr>
<tr>
<td>bake</td>
<td>4</td>
</tr>
<tr>
<td>bake()</td>
<td>5, 26, 28</td>
</tr>
<tr>
<td>base::as.integer()</td>
<td>129</td>
</tr>
<tr>
<td>base::make.names()</td>
<td>56</td>
</tr>
<tr>
<td>check_class</td>
<td>6</td>
</tr>
<tr>
<td>check_cols</td>
<td>8</td>
</tr>
<tr>
<td>check_missing</td>
<td>9</td>
</tr>
<tr>
<td>check_new_values</td>
<td>11</td>
</tr>
<tr>
<td>check_range</td>
<td>13</td>
</tr>
<tr>
<td>current_info (has_role)</td>
<td>19</td>
</tr>
<tr>
<td>ddalpha::depth.halfspace()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.Mahalanobis()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.potential()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.projection()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.simplicial()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.simplicialVolume()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.spatial()</td>
<td>53</td>
</tr>
<tr>
<td>ddalpha::depth.zonoid()</td>
<td>53</td>
</tr>
<tr>
<td>denom_vars (step_ratio)</td>
<td>149</td>
</tr>
<tr>
<td>detect_step</td>
<td>15</td>
</tr>
<tr>
<td>dimRed::Isomap()</td>
<td>99</td>
</tr>
<tr>
<td>discretize</td>
<td>16</td>
</tr>
<tr>
<td>discretize()</td>
<td>55</td>
</tr>
<tr>
<td>dplyr::arrange()</td>
<td>32</td>
</tr>
<tr>
<td>dplyr::filter()</td>
<td>60</td>
</tr>
<tr>
<td>dplyr::mutate()</td>
<td>115, 116</td>
</tr>
<tr>
<td>dplyr::mutate_at()</td>
<td>117, 118</td>
</tr>
<tr>
<td>dplyr::rename()</td>
<td>156, 157</td>
</tr>
<tr>
<td>dplyr::rename_at()</td>
<td>158</td>
</tr>
<tr>
<td>dplyr::sample_frac()</td>
<td>160, 161</td>
</tr>
<tr>
<td>dplyr::sample_n()</td>
<td>160, 161</td>
</tr>
<tr>
<td>dplyr::select()</td>
<td>164</td>
</tr>
<tr>
<td>dplyr::slice()</td>
<td>167, 168</td>
</tr>
<tr>
<td>dplyr::vars()</td>
<td>141, 145</td>
</tr>
<tr>
<td>dummy_names ()</td>
<td>21</td>
</tr>
<tr>
<td>dummy_names ()</td>
<td>57, 126, 136, 175</td>
</tr>
<tr>
<td>everything ()</td>
<td>5, 21</td>
</tr>
<tr>
<td>fastICA::fastICA()</td>
<td>68</td>
</tr>
<tr>
<td>formula.recipe</td>
<td>18</td>
</tr>
<tr>
<td>fully_trained</td>
<td>18</td>
</tr>
<tr>
<td>gower::gower_topn()</td>
<td>74</td>
</tr>
</tbody>
</table>
prep(); .47
grepl(); .152

hardhat::default_recipe_blueprint(), .125
has_role, .19
has_role(), .30, .32
has_type(has_role), .19
has_type(); .32

imp_vars(step_impute_bag), .69
ipred::ipredbagg(), .71

juice, .20

kernlab::kpca(), .102, .104, .107

lm(); .76

make.names(); .57

names0, .21

predict.discretize(discretize), .16

prepper, .24
print.recipe, .25

recipe; .4, .5, .7, .14, .21, .28, .29, .38, .39, .41,
        .45, .51, .61, .65, .66, .69, .95, .96, .98,
        .100, .103, .105, .108, .109, .111, .113,
        .115, .120, .122, .128, .132, .139, .142,
        .144, .155, .161, .168, .172, .177, .182,
        .183, .185

recipes, .28

remove_role(roles), .29

rlang::quos(); .183

roles, .29

selection(selections), .31

selections(), .5, .6, .9, .10, .12, .14, .21, .29, .35,
        .37, .39, .40, .42, .44, .47, .48, .50, .52, .54,
        .56, .58, .63, .64, .66, .68, .70, .71, .73, .74,
        .76, .78, .80, .82, .84, .86, .88, .90, .92, .96,
        .97, .99, .101, .104, .106, .109, .110, .112,
        .114, .118, .119, .121, .123, .125, .127,
        .129, .131, .133, .135, .138, .140, .143,
        .145, .148, .150, .151, .153, .154, .158,
        .159, .163, .165, .166, .170, .172, .173,
        .175, .176, .178, .181, .183

splines::bs(), .39

tsplines::ns(), .127

stats::cor(); .45

stats::poly(), .143, .144

stats::prcomp(), .138

stats::prcomp.default(), .138

stats::quantile(); .16

step_arrange, .32

step_impute_bag(step_impute_bag), .69

step_bin2factor, .34

step_BoxCox, .36

step_BoxCox(), .182

step_bs, .38

step_center, .40

step_center(), .29, .139

step_classdist, .42

step_corr, .44

step_corr(); .132, .183

step_count, .46

step_count(); .57, .91, .126, .136, .175

step_cut, .48

step_date, .50

step_date(); .65

step_depth, .52

step_discretize, .54

step_dummy, .55

step_dummy(), .21, .22, .29, .32, .59, .91–.93, .129,
            .174

step_factor2string, .58

step_factor2string(); .57, .91, .126, .129,
            .136, .174, .175

step_filter, .60

step_filter(); .120, .162, .168

step_geodist, .62

step_holiday, .64
step_holiday() \textcolor{red}{51} 
step_hyperbolic, \textcolor{red}{65} 
step_hyperbolic(), \textcolor{red}{96, 98, 113, 115, 172} 
step_ica, \textcolor{red}{67} 
step_ica(), \textcolor{red}{100, 103, 105, 108, 122, 139, 142} 
step_impute_bag, \textcolor{red}{69} 
step_impute_knn, \textcolor{red}{72} 
step_impute_linear, \textcolor{red}{75} 
step_impute_lower, \textcolor{red}{77} 
step_impute_mean, \textcolor{red}{79} 
step_impute_median, \textcolor{red}{82} 
step_impute_mode, \textcolor{red}{84} 
step_impute_roll, \textcolor{red}{86} 
step_indicate_na, \textcolor{red}{88} 
step_integer, \textcolor{red}{89} 
step_interact, \textcolor{red}{91} 
step_interact(), \textcolor{red}{32} 
step_intercept, \textcolor{red}{94} 
step_inverse, \textcolor{red}{95} 
step_invlogit, \textcolor{red}{97} 
step_invlogit(), \textcolor{red}{66, 113, 115, 172} 
step_isomap, \textcolor{red}{98} 
step_isomap(), \textcolor{red}{69, 103, 105, 108, 122, 139} 
step_knnimpute (step_impute_knn), \textcolor{red}{72} 
step_kpca, \textcolor{red}{101} 
step_kpca(), \textcolor{red}{69, 100, 122, 139, 142} 
step_kpca_poly, \textcolor{red}{103} 
step_kpca_rbf, \textcolor{red}{106} 
step_lag, \textcolor{red}{108} 
step_lincomb, \textcolor{red}{110} 
step_log, \textcolor{red}{112} 
step_log(), \textcolor{red}{66, 96, 98, 115, 172} 
step_logit, \textcolor{red}{114} 
step_logit(), \textcolor{red}{66, 98, 113, 172} 
step_lowerimpute (step_impute_lower), \textcolor{red}{77} 
step_meanimpute (step_impute_mean), \textcolor{red}{79} 
step_medianimpute (step_impute_median), \textcolor{red}{82} 
step_modeimpute (step_impute_mode), \textcolor{red}{84} 
step_mutate, \textcolor{red}{115} 
step_mutate_at, \textcolor{red}{117} 
step_naomit, \textcolor{red}{119} 
step_naomit(), \textcolor{red}{61, 108, 109, 162, 168} 
step_nnmf, \textcolor{red}{120} 
step_normalize, \textcolor{red}{122} 
step_novel, \textcolor{red}{124} 
step_novel(), \textcolor{red}{57, 91, 136, 174, 175} 
step_ns, \textcolor{red}{126} 
step_ns(), \textcolor{red}{39, 144} 
step_num2factor, \textcolor{red}{128} 
step_nzv, \textcolor{red}{130} 
step_nzv(), \textcolor{red}{45, 111, 183} 
step_ordinalscore, \textcolor{red}{133} 
step_ordinalscore(), \textcolor{red}{57, 91, 126, 136, 175, 177} 
step_other, \textcolor{red}{135} 
step_other(), \textcolor{red}{57, 91, 126, 174, 175} 
step_pca, \textcolor{red}{137} 
step_pca(), \textcolor{red}{31, 69, 100, 103, 105, 108, 122, 142} 
step_pls, \textcolor{red}{140} 
step_poly, \textcolor{red}{143} 
step_poly(), \textcolor{red}{39, 128} 
step_profile, \textcolor{red}{145} 
step_range, \textcolor{red}{147} 
step_ratio, \textcolor{red}{149} 
step_regex, \textcolor{red}{151} 
step_regex(), \textcolor{red}{57, 91, 126, 136, 175} 
step_relevel, \textcolor{red}{152} 
step_relu, \textcolor{red}{154} 
step_rename, \textcolor{red}{156} 
step_rename_at, \textcolor{red}{158} 
step_rm, \textcolor{red}{159} 
step_rm(), \textcolor{red}{51, 65} 
step_rollimpute (step_impute_roll), \textcolor{red}{86} 
step_sample, \textcolor{red}{160} 
step_sample(), \textcolor{red}{61, 120, 168} 
step_scale, \textcolor{red}{162} 
step_scale(), \textcolor{red}{139} 
step_select, \textcolor{red}{164} 
step_shuffle, \textcolor{red}{166} 
step_slice, \textcolor{red}{167} 
step_slice(), \textcolor{red}{61, 120, 162} 
step_spatsign, \textcolor{red}{169} 
step_sqrt, \textcolor{red}{171} 
step_sqrt(), \textcolor{red}{66, 96, 98, 113, 115} 
step_string2factor, \textcolor{red}{173} 
step_string2factor(), \textcolor{red}{57, 59, 91, 126, 129, 136, 175} 
step_unknown, \textcolor{red}{174} 
step_unknown(), \textcolor{red}{57} 
step_unorder, \textcolor{red}{176} 
step_unorder(), \textcolor{red}{57, 91, 126, 136, 175} 
step_window, \textcolor{red}{178} 
step_YeoJohnson, \textcolor{red}{180} 
step_YeoJohnson(), \textcolor{red}{38}
tidy.check (tidy.recipe), 186

tidy.check_class (tidy.recipe), 186

tidy.check_correlation (tidy.recipe), 44

tidy.check_missing (tidy.recipe), 9

tidy.check_range (tidy.recipe), 13

tidy.recipe, 186

(tidy.recipe) (tidy.recipe), 185

summary.recipe (), 185

summary.recipe, 184

summary.recipe(), 185

tidy.step (tidy.recipe), 186

tidy.step_arrange (tidy.recipe), 32

tidy.step_bin2factor (tidy.recipe), 34

tidy.step_BoxCox (tidy.recipe), 36

tidy.step_bs (tidy.recipe), 38

tidy.step_center (tidy.recipe), 40

tidy.step_classdist (tidy.recipe), 42

tidy.step_corr (tidy.recipe), 44

tidy.step_count (tidy.recipe), 46

tidy.step_cut (tidy.recipe), 48

tidy.step_date (tidy.recipe), 50

tidy.step_depth (tidy.recipe), 52

tidy.step_discretize (tidy.recipe), 54

tidy.step_dummy (tidy.recipe), 55

tidy.step_factor2string (tidy.recipe), 58

tidy.step_filter (tidy.recipe), 60

tidy.step_geodist (tidy.recipe), 62

tidy.step_holiday (tidy.recipe), 64

tidy.step_hyperbolic (tidy.recipe), 65

tidy.step_ica (tidy.recipe), 67

tidy.step_impute_bag (tidy.recipe), 69

tidy.step_impute_knn (tidy.recipe), 72

tidy.step_impute_linear (tidy.recipe), 75

tidy.step_impute_lower (tidy.recipe), 77

tidy.step_impute_mean (tidy.recipe), 79

tidy.step_impute_median (tidy.recipe), 82

tidy.step_impute_mode (tidy.recipe), 84

tidy.step_impute_roll (tidy.recipe), 86

tidy.step_impute_roll (tidy.recipe), 86

tidy.step_indicate_na (tidy.recipe), 88

tidy.step_integer (tidy.recipe), 89

tidy.step_interact (tidy.recipe), 91

tidy.step_inverse (tidy.recipe), 95

tidy.step_invlogit (tidy.recipe), 97

tidy.step_isomap (tidy.recipe), 98

tidy.step_kpca (tidy.recipe), 101

tidy.step_kpca_poly (tidy.recipe), 103

tidy.step_kpca_rbf (tidy.recipe), 106

tidy.step_lincomb (tidy.recipe), 110

tidy.step_log (tidy.recipe), 112

tidy.step_logit (tidy.recipe), 114

tidy.step_mutate (tidy.recipe), 115

tidy.step_mutate_at (tidy.recipe), 115

tidy.step_naomit (tidy.recipe), 119

tidy.step_nnmf (tidy.recipe), 120

tidy.step_normalize (tidy.recipe), 122

tidy.step_novel (tidy.recipe), 124

tidy.step_ns (tidy.recipe), 126

tidy.step_num2factor (tidy.recipe), 128

tidy.step_nzv (tidy.recipe), 130

tidy.step_ordinalscore (tidy.recipe), 133

tidy.step_other (tidy.recipe), 135

tidy.step_pls (tidy.recipe), 140

tidy.step_poly (tidy.recipe), 143

tidy.step_profile (tidy.recipe), 145

tidy.step_range (tidy.recipe), 147

tidy.step_ratio (tidy.recipe), 149

tidy.step_regex (tidy.recipe), 151

tidy.step_relevel (tidy.recipe), 152

tidy.step_relu (tidy.recipe), 154

tidy.step_rename (tidy.recipe), 156

tidy.step_rename_at (tidy.recipe), 156

tidy.step_rm (tidy.recipe), 159

tidy.step_sample (tidy.recipe), 160

tidy.step_scale (tidy.recipe), 162

tidy.step_select (tidy.recipe), 164

tidy.step_shuffle (tidy.recipe), 166

tidy.step_slice (tidy.recipe), 167

tidy.step_spatialsign
(step_spatialsign), 169
 tidy.step_sqrt (step_sqrt), 171
 tidy.step_string2factor
   (step_string2factor), 173
 tidy.step_unknown (step_unknown), 174
 tidy.step_unorder (step_unorder), 176
 tidy.step_window (step_window), 178
 tidy.step_YeoJohnson (step_YeoJohnson), 180
 tidy.step_zv (step_zv), 182
 tidyselect::all_of(), 32
 tidyselect::any_of(), 32
 tidyselect::contains(), 32
 tidyselect::ends_with(), 32
 tidyselect::everything(), 32
 tidyselect::matches(), 32
 tidyselect::num_range(), 32
 tidyselect::one_of(), 32
 tidyselect::starts_with(), 32, 92
 timeDate::listHolidays(), 64, 65

 update.step, 187
 update_role (roles), 29

 workflows::add_recipe(), 125