# recipes

## Introduction

The recipes package is an alternative method for creating and preprocessing design matrices that can be used for modeling or visualization. From [wikipedia](https://en.wikipedia.org/wiki/Design_matrix):

In statistics, a **design matrix** (also known as regressor matrix or model matrix) is a matrix of values of explanatory variables of a set of objects, often denoted by X. Each row represents an individual object, with the successive columns corresponding to the variables and their specific values for that object.

While R already has long-standing methods for creating these matrices (e.g. [formulas](https://www.rstudio.com/rviews/2017/02/01/the-r-formula-method-the-good-parts) and model.matrix), there are some [limitations to what the existing infrastructure can do](https://rviews.rstudio.com/2017/03/01/the-r-formula-method-the-bad-parts/).

The idea of the recipes package is to define a recipe or blueprint that can be used to sequentially define the encodings and preprocessing of the data (i.e. “feature engineering”). For example, to create a simple recipe containing only an outcome and predictors and have the predictors centered and scaled:

[library](https://www.rdocumentation.org/packages/base/topics/library)(recipes)

[library](https://www.rdocumentation.org/packages/base/topics/library)(mlbench)

[data](https://www.rdocumentation.org/packages/utils/topics/data)(Sonar)

sonar\_rec <- [recipe](https://tidymodels.github.io/recipes/reference/recipe.html)(Class ~ ., data = Sonar) %>%

[step\_center](https://tidymodels.github.io/recipes/reference/step_center.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)()) %>%

[step\_scale](https://tidymodels.github.io/recipes/reference/step_scale.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)())

To install it, use:

[install.packages](https://www.rdocumentation.org/packages/utils/topics/install.packages)("recipes")

## for development version:

[require](https://www.rdocumentation.org/packages/base/topics/library)("devtools")

[install\_github](https://www.rdocumentation.org/packages/devtools/topics/reexports)("tidymodels/recipes")

## Basic Recipes

Source: [vignettes/Simple\_Example.Rmd](https://github.com/tidymodels/recipes/blob/master/vignettes/Simple_Example.Rmd)

This document demonstrates some basic uses of recipes. First, some definitions are required:

* **variables** are the original (raw) data columns in a data frame or tibble. For example, in a traditional formula Y ~ A + B + A:B, the variables are A, B, and Y.
* **roles** define how variables will be used in the model. Examples are: predictor (independent variables), response, and case weight. This is meant to be open-ended and extensible.
* **terms** are columns in a design matrix such as A, B, and A:B. These can be other derived entities that are grouped such a a set of principal components or a set of columns that define a basis function for a variable. These are synonymous with features in machine learning. Variables that have predictor roles would automatically be main effect terms

## An Example

The packages contains a data set that used to predict whether a person will pay back a bank loan. It has 13 predictor columns and a factor variable Status (the outcome). We will first separate the data into a training and test set:

[library](https://www.rdocumentation.org/packages/base/topics/library)(recipes)

[library](https://www.rdocumentation.org/packages/base/topics/library)(rsample)

[data](https://www.rdocumentation.org/packages/utils/topics/data)("credit\_data")

[set.seed](https://www.rdocumentation.org/packages/base/topics/Random)(55)

train\_test\_split <- [initial\_split](https://www.rdocumentation.org/packages/rsample/topics/initial_split)(credit\_data)

credit\_train <- [training](https://www.rdocumentation.org/packages/rsample/topics/initial_split)(train\_test\_split)

credit\_test <- [testing](https://www.rdocumentation.org/packages/rsample/topics/initial_split)(train\_test\_split)

Note that there are some missing values in these data:

[vapply](https://www.rdocumentation.org/packages/base/topics/lapply)(credit\_train, function(x) [mean](https://www.rdocumentation.org/packages/base/topics/mean)(![is.na](https://www.rdocumentation.org/packages/base/topics/NA)(x)), [numeric](https://www.rdocumentation.org/packages/base/topics/numeric)(1))

#> Status Seniority Home Time Age Marital Records

#> 1.000 1.000 0.999 1.000 1.000 1.000 1.000

#> Job Expenses Income Assets Debt Amount Price

#> 0.999 1.000 0.914 0.991 0.996 1.000 1.000

Rather than remove these, their values will be imputed.

The idea is that the preprocessing operations will all be created using the training set and then these steps will be applied to both the training and test set.

## An Initial Recipe

First, we will create a recipe object from the original data and then specify the processing steps.

Recipes can be created manually by sequentially adding roles to variables in a data set.

If the analysis only required **outcomes** and **predictors**, the easiest way to create the initial recipe is to use the standard formula method:

rec\_obj <- [recipe](https://tidymodels.github.io/recipes/reference/recipe.html)(Status ~ ., data = credit\_train)

rec\_obj

#> Data Recipe

#>

#> Inputs:

#>

#> role #variables

#> outcome 1

#> predictor 13

The data contained in the data argument need not be the training set; this data is only used to catalog the names of the variables and their types (e.g. numeric, etc.).

(Note that the formula method here is used to declare the variables and their roles and nothing else. If you use inline functions (e.g. log) it will complain. These types of operations can be added later.)

## Preprocessing Steps

From here, preprocessing steps for some step X can be added sequentially in one of two ways:

rec\_obj <- step\_{X}(rec\_obj, arguments) ## or

rec\_obj <- rec\_obj %>% step\_{X}(arguments)

step\_dummy and the other functions will always return updated recipes.

One other important facet of the code is the method for specifying which variables should be used in different steps. The manual page [?selections](https://tidymodels.github.io/recipes/reference/selections.html) has more details but [dplyr](https://cran.r-project.org/package=dplyr)-like selector functions can be used:

* use basic variable names (e.g. x1, x2),
* [dplyr](https://cran.r-project.org/package=dplyr) functions for selecting variables: contains, ends\_with, everything, matches, num\_range, and starts\_with,
* functions that subset on the role of the variables that have been specified so far: all\_outcomes, all\_predictors, has\_role, or
* similar functions for the type of data: all\_nominal, all\_numeric, and has\_type.

Note that the methods listed above are the only ones that can be used to select variables inside the steps. Also, minus signs can be used to deselect variables.

For our data, we can add an operation to impute the predictors. There are many ways to do this and recipes includes a few steps for this purpose:

[grep](https://www.rdocumentation.org/packages/base/topics/grep)("impute$", [ls](https://www.rdocumentation.org/packages/base/topics/ls)("package:recipes"), value = TRUE)

#> [1] "step\_bagimpute" "step\_knnimpute" "step\_lowerimpute"

#> [4] "step\_meanimpute" "step\_medianimpute" "step\_modeimpute"

#> [7] "step\_rollimpute"

Here, K-nearest neighbor imputation will be used. This works for both numeric and non-numeric predictors and defaults K to five To do this, it selects all predictors then removes those that are numeric:

imputed <- rec\_obj %>%

[step\_knnimpute](https://tidymodels.github.io/recipes/reference/step_knnimpute.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)())

imputed

#> Data Recipe

#>

#> Inputs:

#>

#> role #variables

#> outcome 1

#> predictor 13

#>

#> Operations:

#>

#> 5-nearest neighbor imputation for all\_predictors()

It is important to realize that the specific variables have not been declared yet (as shown when the recipe is printed above). In some preprocessing steps, variables will be added or removed from the current list of possible variables.

Since some predictors are categorical in nature (i.e. nominal), it would make sense to convert these factor predictors into numeric dummy variables (aka indicator variables) using step\_dummy. To do this, the step selects all predictors then removes those that are numeric:

ind\_vars <- imputed %>%

[step\_dummy](https://tidymodels.github.io/recipes/reference/step_dummy.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)(), -[all\_numeric](https://tidymodels.github.io/recipes/reference/has_role.html)())

ind\_vars

#> Data Recipe

#>

#> Inputs:

#>

#> role #variables

#> outcome 1

#> predictor 13

#>

#> Operations:

#>

#> 5-nearest neighbor imputation for all\_predictors()

#> Dummy variables from all\_predictors(), -all\_numeric()

At this point in the recipe, all of the predictor should be encoded as numeric, we can further add more steps to center and scale them:

standardized <- ind\_vars %>%

[step\_center](https://tidymodels.github.io/recipes/reference/step_center.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)()) %>%

[step\_scale](https://tidymodels.github.io/recipes/reference/step_scale.html)([all\_predictors](https://tidymodels.github.io/recipes/reference/has_role.html)())

standardized

#> Data Recipe

#>

#> Inputs:

#>

#> role #variables

#> outcome 1

#> predictor 13

#>

#> Operations:

#>

#> 5-nearest neighbor imputation for all\_predictors()

#> Dummy variables from all\_predictors(), -all\_numeric()

#> Centering for all\_predictors()

#> Scaling for all\_predictors()

If there are the only preprocessing steps for the predictors, we can now estimate the means and standard deviations from the training set. The prep function is used with a recipe and a data set:

trained\_rec <- [prep](https://tidymodels.github.io/recipes/reference/prep.html)(standardized, training = credit\_train)

trained\_rec

#> Data Recipe

#>

#> Inputs:

#>

#> role #variables

#> outcome 1

#> predictor 13

#>

#> Training data contained 3341 data points and 309 incomplete rows.

#>

#> Operations:

#>

#> 5-nearest neighbor imputation for Home, Time, Age, Marital, Records, ... [trained]

#> Dummy variables from Home, Marital, Records, Job [trained]

#> Centering for Seniority, Time, Age, Expenses, ... [trained]

#> Scaling for Seniority, Time, Age, Expenses, ... [trained]

Note that the real variables are listed (e.g. Home etc.) instead of the selectors ([all\_predictors()](https://tidymodels.github.io/recipes/reference/has_role.html)).

Now that the statistics have been estimated, the preprocessing can be applied to the training and test set:

train\_data <- [bake](https://tidymodels.github.io/recipes/reference/bake.html)(trained\_rec, new\_data = credit\_train)

test\_data <- [bake](https://tidymodels.github.io/recipes/reference/bake.html)(trained\_rec, new\_data = credit\_test)

bake returns a tibble that, by default, includes all of the variables:

[class](https://www.rdocumentation.org/packages/base/topics/class)(test\_data)

#> [1] "tbl\_df" "tbl" "data.frame"

test\_data

#> # A tibble: 1,113 x 23

#> Status Seniority Time Age Expenses Income Assets Debt

#> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

#> 1 good -0.987 0.938 -1.19 0.376 0.497 -0.243 -0.275

#> 2 good -0.987 -0.724 -1.01 -0.499 -0.447 -0.457 -0.275

#> 3 good -0.987 0.938 -0.464 1.76 -0.447 0.827 -0.275

#> 4 good 1.36 -0.724 -0.00678 0.993 0.346 -0.157 -0.0739

#> 5 bad -0.740 0.938 -1.10 -0.499 -0.447 -0.457 -0.275

#> 6 bad -0.864 0.938 0.724 2.54 -0.384 -0.285 0.112

#> 7 good 2.23 0.938 1.27 -0.550 0.00644 -0.157 -0.275

#> 8 good 1.36 0.938 0.541 0.993 0.472 -0.114 -0.275

#> 9 bad -0.616 -1.56 -1.29 0.993 -0.723 -0.0287 -0.275

#> 10 good 2.97 -1.56 2.28 -0.550 -0.296 0.271 -0.275

#> # … with 1,103 more rows, and 15 more variables: Amount <dbl>,

#> # Price <dbl>, Home\_other <dbl>, Home\_owner <dbl>, Home\_parents <dbl>,

#> # Home\_priv <dbl>, Home\_rent <dbl>, Marital\_married <dbl>,

#> # Marital\_separated <dbl>, Marital\_single <dbl>, Marital\_widow <dbl>,

#> # Records\_yes <dbl>, Job\_freelance <dbl>, Job\_others <dbl>,

#> # Job\_partime <dbl>

[vapply](https://www.rdocumentation.org/packages/base/topics/lapply)(test\_data, function(x) [mean](https://www.rdocumentation.org/packages/base/topics/mean)(![is.na](https://www.rdocumentation.org/packages/base/topics/NA)(x)), [numeric](https://www.rdocumentation.org/packages/base/topics/numeric)(1))

#> Status Seniority Time Age

#> 1 1 1 1

#> Expenses Income Assets Debt

#> 1 1 1 1

#> Amount Price Home\_other Home\_owner

#> 1 1 1 1

#> Home\_parents Home\_priv Home\_rent Marital\_married

#> 1 1 1 1

#> Marital\_separated Marital\_single Marital\_widow Records\_yes

#> 1 1 1 1

#> Job\_freelance Job\_others Job\_partime

#> 1 1 1

Selectors can also be used. For example, if only the predictors are needed, you can use [bake(object, new\_data, all\_predictors())](https://tidymodels.github.io/recipes/reference/bake.html).

There are a number of other steps included in the package:

#> [1] "step\_arrange" "step\_bagimpute" "step\_bin2factor"

#> [4] "step\_BoxCox" "step\_bs" "step\_center"

#> [7] "step\_classdist" "step\_corr" "step\_count"

#> [10] "step\_date" "step\_depth" "step\_discretize"

#> [13] "step\_downsample" "step\_dummy" "step\_factor2string"

#> [16] "step\_filter" "step\_geodist" "step\_holiday"

#> [19] "step\_hyperbolic" "step\_ica" "step\_integer"

#> [22] "step\_interact" "step\_intercept" "step\_inverse"

#> [25] "step\_invlogit" "step\_isomap" "step\_knnimpute"

#> [28] "step\_kpca" "step\_lag" "step\_lincomb"

#> [31] "step\_log" "step\_logit" "step\_lowerimpute"

#> [34] "step\_meanimpute" "step\_medianimpute" "step\_modeimpute"

#> [37] "step\_mutate" "step\_naomit" "step\_nnmf"

#> [40] "step\_novel" "step\_ns" "step\_num2factor"

#> [43] "step\_nzv" "step\_ordinalscore" "step\_other"

#> [46] "step\_pca" "step\_pls" "step\_poly"

#> [49] "step\_profile" "step\_range" "step\_ratio"

#> [52] "step\_regex" "step\_relu" "step\_rm"

#> [55] "step\_rollimpute" "step\_sample" "step\_scale"

#> [58] "step\_shuffle" "step\_slice" "step\_spatialsign"

#> [61] "step\_sqrt" "step\_string2factor" "step\_unorder"

#> [64] "step\_upsample" "step\_window" "step\_YeoJohnson"

#> [67] "step\_zv"

## Checks

Another type of operation that can be added to a recipes is a check. Checks conduct some sort of data validation and, if no issue is found, returns the data as-is; otherwise, an error is thrown.

For example, check\_missing will fail if any of the variables selected for validation have missing values. This check is done when the recipe is prepared as well as when any data are baked. Checks are added in the same way as steps:

trained\_rec <- trained\_rec %>%

[check\_missing](https://tidymodels.github.io/recipes/reference/check_missing.html)(contains("Marital"))

Currently, recipes includes:

#> [1] "check\_cols" "check\_missing" "check\_name" "check\_range"

#> [5] "check\_type"