Note, you should already be fairly familiar with the [recipes](https://cran.r-project.org/web/packages/recipes/recipes.pdf) package before you continue reading this post or give customsteps a spin!

Recommended music for this reading session:

**Introducing the customsteps package**

Along with the recipes package distribution comes a number of pre-specified steps, that enables the user to manipulate data sets in various ways. The resulting data sets (/design matrices) can then be used as inputs into statistical or machine learning models.

If you want to apply a specific transformation to your data set, that is not supported by the pre-specified steps, you have two options. You can write an entire [custom recipe step **from scratch**](https://github.com/tidymodels/recipes/blob/master/vignettes/Custom_Steps.Rmd). This however takes quite a bit of work and code. An alternative – and sometimes better – approach is to apply the customsteps package, that I have just released on CRAN.

# install.packages("customsteps")

library(customsteps)

**Customizable Higher-Order Steps**

customsteps contains a set of customizable higher-order recipe step functions, that create specifications of recipe steps, that will transform or filter the data in accordance with custom input functions.

Let me just remind you of the definition of [**higher-order functions**](https://en.wikipedia.org/wiki/Higher-order_function):

*In mathematics and computer science, a higher-order function is a function that does at least one of the following: 1. takes one or more functions as arguments, 2. returns a function as its result.*

Next, I will present an example of how to use the customsteps package in order to create a recipe step, that will apply a custom transformation to a data set.

**Use Case: Centering and Scaling Numeric Data**

Assume, that I want to transform a variable \({\mathbf{x}}\) like this:

1. Center \({\mathbf{x}}\) around an arbitrary number \(\alpha\).
2. Scale the transformed variable, such that its standard deviation equals an arbitrary number \(\beta\).

The transformed variable \(\hat{\mathbf{x}}\) can then be derived as (try to do it yourself):

\(\hat{\mathbf{x}} = \alpha + (\mathbf{x} – \bar{\mathbf{x}})\frac{\beta}{s\_\mathbf{x}}\)

where \(\bar{\mathbf{x}}\) is the mean of \(\mathbf{x}\), and \(s\_\mathbf{x}\) is the standard deviation of \({\mathbf{x}}\).

Note that centering \({\mathbf{x}}\) around 0 and scaling it in order to arrive at a standard deviation of 1 is just a special case of the above transformation with parameters \(\alpha = 0, \beta = 1\).

**Write the prep helper function**

First, I need to write a function, that estimates the relevant statistical parameters from an initial data set. I call this function the prep helper function.

Obviously, the above transformation requires the mean \(\bar{\mathbf{x}}\) and standard deviation \(s\_\mathbf{x}\) to be learned from the initial data set. Therefore I define a function compute\_means\_sd, that estimates the two parameters for (any arbitrary number of) numeric variables.

By convention the prep helper function must take the argument x: the subset of selected variables from the initial data set.

library(purrr)

compute\_means\_sd <- function(x) {

map(.x = x, ~ list(mean = mean(.x), sd = sd(.x)))

}

Let us see the function in action. I will apply it to a subset of the famous mtcars data set.

library(dplyr)

# divide 'mtcars' into two data sets.

cars\_initial <- mtcars[1:16, ]

cars\_new <- mtcars[17:nrow(mtcars), ]

# learn parameters from initial data set.

params <- cars\_initial %>%

select(mpg, disp) %>%

compute\_means\_sd(.)

# display parameters.

as.data.frame(params)

#> mpg.mean mpg.sd disp.mean disp.sd

#> 1 18.2 4.14761 250.8187 113.372

It works like a charm. Great, we are halfway there!

**Write the bake helper function**

Second, I have to specify a bake helper function, that defines how to apply the transformation to a new data set using the parameters estimated from the intial data set.

By convention the bake helper function must take the following arguments:

* x: the new data set, that the step will be applied to.
* prep\_output: the output from the prep helper function containing any parameters estimated from the initial data set.

I define the function center\_scale, that will serve as my bake helper function. It will center and scale variables of a new data set.

center\_scale <- function(x, prep\_output, alpha, beta) {

# extract only the relevant variables from the new data set.

new\_data <- select(x, names(prep\_output))

# apply transformation to each of these variables.

# variables are centered around 'alpha' and scaled to have a standard

# deviation of 'beta'.

map2(.x = new\_data,

.y = prep\_output,

~ alpha + (.x - .y$mean) \* beta / .y$sd)

}

My first (sanity) check of the function is to apply it to the initial data set, that was used for estimation of the means and standard deviations.

library(tibble)

# center and scale variables of new data set to have a mean of zero

# and a standard deviation of one.

cars\_initial\_transformed <- center\_scale(x = cars\_initial,

prep\_output = params,

alpha = 0,

beta = 1)

# display transformed variables.

cars\_initial\_transformed %>%

compute\_means\_sd(.) %>%

as.data.frame(.)

#> mpg.mean mpg.sd disp.mean disp.sd

#> 1 1.731877e-16 1 7.199102e-17 1

Results are correct within computational precision.

Also, I will just check the function out on the other subset of mtcars.

# center and scale variables of new data set to have a mean of zero

# and a standard deviation of one.

cars\_new\_transformed <- center\_scale(x = cars\_new,

prep\_output = params,

alpha = 0,

beta = 1)

# display transformed variables.

cars\_new\_transformed %>%

as.tibble(.) %>%

head(.)

#> # A tibble: 6 x 2

#> mpg disp

#>

#> 1 -0.844 1.67

#> 2 3.42 -1.52

#> 3 2.94 -1.54

#> 4 3.79 -1.59

#> 5 0.796 -1.15

#> 6 -0.651 0.593

Looks right! All that is left now is to put the pieces together into my new very own custom recipe step.

**Putting the pieces together**

The function step\_custom\_transformation takes prep and bake helper functions as inputs and turns them into a complete recipe step, that can be used out of the box.

I create the specification of the recipe step from the new functions compute\_means\_sd and center\_scale by invoking step\_custom\_transformation.

library(recipes)

rec <- recipe(cars\_initial) %>%

step\_custom\_transformation(mpg, disp,

prep\_function = compute\_means\_sd,

bake\_function = center\_scale,

bake\_options = list(alpha = 0, beta = 1),

bake\_how = "replace")

And that is all there is to it! Easy.

Note, by setting ‘bake\_options’ to “replace”, the selected terms will be replaced with the transformed variables, when the recipe is baked.

I will just check, that the recipe works as expected. First I will prep(/train) the recipe.

# prep recipe.

rec <- prep(rec)

# print recipe.

rec

#> Data Recipe

#>

#> Inputs:

#>

#> 11 variables (no declared roles)

#>

#> Training data contained 16 data points and no missing data.

#>

#> Operations:

#>

#> The following variables are used for computing transformations

#> and will be dropped afterwards:

#> mpg, disp

I will go right ahead and bake the new recipe.

# bake recipe.

cars\_baked <- rec %>%

bake(cars\_new) %>%

select(mpg, disp)

# display results.

cars\_baked %>%

head(.)

#> # A tibble: 6 x 2

#> mpg disp

#>

#> 1 -0.844 1.67

#> 2 3.42 -1.52

#> 3 2.94 -1.54

#> 4 3.79 -1.59

#> 5 0.796 -1.15

#> 6 -0.651 0.593

Results are as expected (same as before). Great succes!

You should now be able to create your very own recipe steps to do (almost) whatever transformation you want to your data.

**Conclusions**

* Customized recipe steps can be created with a minimum of effort and code using my new R package customsteps.
* The customsteps step functions are higher-order functions, that create specificiations of recipe steps from custom input functions.

Please let me hear from you, if you have any feedback on the package.