Since once of the best way to learn, is to explain, I want to share with you this quick introduction to recipes package, from the tidymodels family.  
It can help us to automatize some data preparation tasks.

The overview is:

* How to create a recipe
* How to add a step
* How to do the prep
* Getting the data with juice!
* Apply the prep to new data
* What is the difference between bake and juice?
* Dealing with new values in recipes (step\_novel)

Since I’m new to this package, if you have something to add just put in the comments 

**Introduction**

If you are new to R or you do a 1-time analysis, you could not see the main advantage of this, which is -in my opinion- to have most of the **data preparation** steps in one place. This way is easier to split between dev and prod.

* Dev: The stage in which we create the model
* Prod: The moment in which we run the model with new data

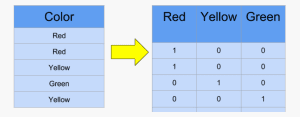
The other big advantage is it follows the *tidy* philosophy, so many things will be familiar.

**How to use recipes for one hot encoding**

It is focused on **one hot encoding**, but many other functions like scaling, applying PCA and others can be performed.

But first, **what is one hot encoding?**

It’s a data preparation technique to convert all the categorical variables into numerical, by assigning a value of 1 when the row belongs to the category. If the variable has 100 unique values, the final result will contain 100 columns.



That’s why it is a **good practice to reduce the cardinality** of the variable before continuing Learn more about it in the [High Cardinality Variable in Predictive Modeling](https://livebook.datascienceheroes.com/data-preparation.html#high_cardinality_predictive_modeling) from the *Data Science Live Book* .

Let’s start the example with recipes!

**1st – How to create a recipe**

library(recipes)

library(tidyverse)

set.seed(3.1415)

iris\_tr=sample\_frac(iris, size = 0.7)

rec = recipe( ~ ., data = iris\_tr)

rec

## Data Recipe

##

## Inputs:

##

## role #variables

## predictor 5

summary(rec)

## # A tibble: 5 x 4

## variable type role source

##

## 1 Sepal.Length numeric predictor original

## 2 Sepal.Width numeric predictor original

## 3 Petal.Length numeric predictor original

## 4 Petal.Width numeric predictor original

## 5 Species nominal predictor original

The formula ~ ., specifies that all the variables are predictors (with no outcomes).

Please note now we have two different data types, numeric and nominal (not factor nor character).

**2nd – How to add a step**

Now we add the step to create the dummy variables, or the **one hot encoding**, which can be seen as the same.

When we do the one hot encoding (one\_hot = T), all the levels will be present in the final result. Conversely, when we create the dummy variables, we could have all of the variables, or one less (to avoid the multi-correlation issue).

rec\_2 = rec %>% step\_dummy(Species, one\_hot = T)

rec\_2

## Data Recipe

##

## Inputs:

##

## role #variables

## predictor 5

##

## Operations:

##

## Dummy variables from Species

Now we see the dummy step.

**3rd – How to do the prep**

prep is like putting all the ingredients together, *but we didn’t cook yet!*

It generates the metadata to do the data preparation.

As we can see here:

# Aplico la receta, que tiene 1 step, a los datos

d\_prep=rec\_2 %>% prep(training = iris\_tr, retain = T)

d\_prep

## Data Recipe

##

## Inputs:

##

## role #variables

## predictor 5

##

## Training data contained 105 data points and no missing data.

##

## Operations:

##

## Dummy variables from Species [trained]

Note we are in the "training" or *dev* stage. That’s why we see the parameter training.

We will see retain = T in the next step.

Checking:

summary(d\_prep)

## # A tibble: 7 x 4

## variable type role source

##

## 1 Sepal.Length numeric predictor original

## 2 Sepal.Width numeric predictor original

## 3 Petal.Length numeric predictor original

## 4 Petal.Width numeric predictor original

## 5 Species\_setosa numeric predictor derived

## 6 Species\_versicolor numeric predictor derived

## 7 Species\_virginica numeric predictor derived

**Whoila!** We have the 3-new derived columns (one hot), and it removed the original Species.

**4th – Getting the data with juice!**

Using juice function:

d2=juice(d\_prep)

head(d2)

## # A tibble: 6 x 7

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species\_setosa

##

## 1 5 3 1.6 0.2 1

## 2 6.9 3.2 5.7 2.3 0

## 3 6.3 3.3 4.7 1.6 0

## 4 5.3 3.7 1.5 0.2 1

## 5 6.3 2.3 4.4 1.3 0

## 6 6.7 3 5.2 2.3 0

## # … with 2 more variables: Species\_versicolor ,

## # Species\_virginica

juice worked because we *retained* the training data in the 3rd step (retain = T). Otherwise it would have returned:

*Error: Use retain = TRUE in prep to be able to extract the training set*

**5th – Apply the prep to new data**

Now imagine we have **new data** as follows:

iris\_new=sample\_n(iris, size = 5) # taking 5 random rows

d\_baked=bake(d\_prep, newdata = iris\_new)

d\_baked

## # A tibble: 5 x 7

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species\_setosa

##

## 1 6.4 3.2 4.5 1.5 0

## 2 4.6 3.4 1.4 0.3 1

## 3 5.2 2.7 3.9 1.4 0

## 4 4.8 3.4 1.6 0.2 1

## 5 4.8 3 1.4 0.3 1

## # … with 2 more variables: Species\_versicolor ,

## # Species\_virginica

It worked!

bake receives the prep object (d\_prep) and it applies to the newdata (iris\_new)

**What is the difference between bake and juice?**

From this perspective given the training data, following data frames are the same:

d\_tr\_1=bake(d\_prep, newdata = iris\_tr)

d\_tr\_2=d2=juice(d\_prep) # with retain=T

identical(d\_tr\_1, d\_tr\_2)

## [1] TRUE

**Dealing with new values in recipes**

Simulate a new value:

new\_row=iris[1,] %>% mutate(Species=as.character(Species))

new\_row[1, "Species"]="i will break your code"

new\_row

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species

## 1 5.1 3.5 1.4 0.2 i will break your code

We use bake to convert the new data set:

d2\_b=bake(d\_prep, newdata = new\_row)

## Warning: There are new levels in a factor: i will break your code

**The solution! Use step\_novel**

(Thanks to Max Kuhn)

When we do the prep, we have to add step\_novel. So any new value will be assigned to the \_new category.

We will start right from the beginning:

rec\_2\_bis = recipe( ~ ., data = iris\_tr) %>%

step\_novel(Species) %>%

step\_dummy(Species, one\_hot = T)

prep\_bis = prep(rec\_2\_bis, training = iris\_tr)

Get to final data, and check it:

processed = bake(prep\_bis, iris\_tr)

funModeling::df\_status(processed)

## variable q\_zeros p\_zeros q\_na p\_na q\_inf p\_inf type unique

## 1 Sepal.Length 0 0.00 0 0 0 0 numeric 32

## 2 Sepal.Width 0 0.00 0 0 0 0 numeric 23

## 3 Petal.Length 0 0.00 0 0 0 0 numeric 42

## 4 Petal.Width 0 0.00 0 0 0 0 numeric 20

## 5 Species\_setosa 68 64.76 0 0 0 0 numeric 2

## 6 Species\_versicolor 69 65.71 0 0 0 0 numeric 2

## 7 Species\_virginica 73 69.52 0 0 0 0 numeric 2

## 8 Species\_new 105 100.00 0 0 0 0 numeric 1

Please note that Species\_new **has been automatically created** (with zeros).

This ensures it **runs well once** in production.

Now let’s see what happen when we have the new value:

new\_row\_2=bake(prep\_bis, newdata = new\_row)

new\_row\_2 %>% select(Species\_new)

## # A tibble: 1 x 1

## Species\_new

##

## 1 1

It works!

**Conclusions **

The recipes package seems to be a good way to standardize certain data preparation tasks.  
Probably one of the strongest points in R, alongside the dplyr package.

Take care of the **data pipeline**, it is what interviewers will ask you for.