# parsnip

One issue with different functions available in R *that do the same thing* is that they can have different interfaces and arguments. For example, to fit a random forest *classification* model, we might have:

# From randomForest

rf\_1 <- randomForest(x, y, mtry = 12, ntree = 2000, importance = TRUE)

# From ranger

rf\_2 <- ranger(

y ~ .,

data = dat,

mtry = 12,

num.trees = 2000,

importance = 'impurity'

)

# From sparklyr

rf\_3 <- ml\_random\_forest(

dat,

intercept = FALSE,

response = "y",

features = [names](https://www.rdocumentation.org/packages/base/topics/names)(dat)[[names](https://www.rdocumentation.org/packages/base/topics/names)(dat) != "y"],

col.sample.rate = 12,

num.trees = 2000

)

Note that the model syntax is very different and that the argument names (and formats) are also different. This is a pain if you go between implementations.

In this example,

* the **type** of model is “random forest”
* the **mode** of the model is “classification” (as opposed to regression, etc).
* the computational **engine** is the name of the R package.

The idea of parsnip is to:

* Separate the definition of a model from its evaluation.
* Decouple the model specification from the implementation (whether the implementation is in R, spark, or something else). For example, the user would call rand\_forest instead of [ranger::ranger](https://www.rdocumentation.org/packages/ranger/topics/ranger) or other specific packages.
* Harmonize the argument names (e.g. n.trees, ntrees, trees) so that users can remember a single name. This will help *across* model types too so that trees will be the same argument across random forest as well as boosting or bagging.

Using the example above, the parsnip approach would be

[rand\_forest](https://tidymodels.github.io/parsnip/reference/rand_forest.html)(mtry = 12, trees = 2000) %>%

[set\_engine](https://tidymodels.github.io/parsnip/reference/set_engine.html)("ranger", importance = 'impurity') %>%

[fit](https://www.rdocumentation.org/packages/generics/topics/fit)(y ~ ., data = dat)

The engine can be easily changed and the mode can be determined when fit is called. To use Spark, the change is simple:

[rand\_forest](https://tidymodels.github.io/parsnip/reference/rand_forest.html)(mtry = 12, trees = 2000) %>%

[set\_engine](https://tidymodels.github.io/parsnip/reference/set_engine.html)("spark") %>%

[fit](https://www.rdocumentation.org/packages/generics/topics/fit)(y ~ ., data = dat)

To install it, use:

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

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

# parsnip Basics

This package provides functions and methods to create and manipulate functions commonly used during modeling (e.g. fitting the model, making predictions, etc). It allows the user to manipulate how the same type of model can be created from different sources. It also contains a basic framework for model parameter tuning.

## Motivation

Modeling functions across different R packages can have very different interfaces. If you would like to try different approaches, there is a lot of syntactical minutiae to remember. The problem worsens when you move in-between platforms (e.g. doing a logistic regression in R’s glm versus Spark’s implementation).

parsnip tries to solve this by providing similar interfaces to models. For example, if you are fitting a random forest model and would like to adjust the number of trees in the forest there are different argument names to remember:

* [randomForest::randomForest](https://www.rdocumentation.org/packages/randomForest/topics/randomForest) uses ntree,
* [ranger::ranger](https://www.rdocumentation.org/packages/ranger/topics/ranger) uses num.trees,
* Spark’s [sparklyr::ml\_random\_forest](https://www.rdocumentation.org/packages/sparklyr/topics/ml_random_forest) uses num\_trees.

Rather than remembering these values, a common interface to these models can be used with

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

rf\_mod <- [rand\_forest](https://tidymodels.github.io/parsnip/reference/rand_forest.html)(trees = 2000)

The package makes the translation between trees and the real names in each of the implementations.

Some terminology:

* The **model type** differentiates models. Example types are: random forests, logistic regression, linear support vector machines, etc.
* The **mode** of the model denotes how it will be used. Two common modes are classification and regression. Others would include “censored regression” and “risk regression” (parametric and Cox PH models for censored data, respectively), as well as unsupervised models (e.g. “clustering”).
* The **computational engine** indicates how the actual model might be fit. These are often R packages (such as randomForest or ranger) but might also be methods outside of R (e.g. Stan, Spark, and others).

parsnip, similar to ggplot2, dplyr and recipes, separates the specification of what you want to do from the actual doing. This allows us to create broader functionality for modeling.

## Placeholders for Parameters

There are times where you would like to change a parameter from its default but you are not sure what the final value will be. This is the basis for model tuning. Since the model is not executing when created, these types of parameters can be changed using the [varying()](https://tidymodels.github.io/parsnip/reference/varying.html) function. This provides a simple placeholder for the value.

tune\_mtry <- [rand\_forest](https://tidymodels.github.io/parsnip/reference/rand_forest.html)(trees = 2000, mtry = [varying](https://tidymodels.github.io/parsnip/reference/varying.html)())

tune\_mtry

#> Random Forest Model Specification (unknown)

#>

#> Main Arguments:

#> mtry = varying()

#> trees = 2000

This will come in handy later when we fit the model over different values of mtry.

## Specifying Arguments

Commonly used arguments to the modeling functions have their parameters exposed in the function. For example, rand\_forest has arguments for:

* mtry: The number of predictors that will be randomly sampled at each split when creating the tree models.
* trees: The number of trees contained in the ensemble.
* min\_n: The minimum number of data points in a node that are required for the node to be split further.

The arguments to the default function are:

[args](https://www.rdocumentation.org/packages/base/topics/args)(rand\_forest)

#> function (mode = "unknown", mtry = NULL, trees = NULL, min\_n = NULL)

#> NULL

However, there might be other arguments that you would like to change or allow to vary. These are accessible using set\_engine. For example, ranger has an option to set the internal random number seed. To set this to a specific value:

rf\_with\_seed <-

[rand\_forest](https://tidymodels.github.io/parsnip/reference/rand_forest.html)(trees = 2000, mtry = [varying](https://tidymodels.github.io/parsnip/reference/varying.html)(), mode = "regression") %>%

[set\_engine](https://tidymodels.github.io/parsnip/reference/set_engine.html)("ranger", seed = 63233)

rf\_with\_seed

#> Random Forest Model Specification (regression)

#>

#> Main Arguments:

#> mtry = varying()

#> trees = 2000

#>

#> Engine-Specific Arguments:

#> seed = 63233

#>

#> Computational engine: ranger

## Process

To fit the model, you must:

* have a defined model, including the mode,
* have no [varying()](https://tidymodels.github.io/parsnip/reference/varying.html) parameters, and
* specify a computational engine.

For example, rf\_with\_seed above is not ready for fitting due the [varying()](https://tidymodels.github.io/parsnip/reference/varying.html) parameter. We can set that parameter’s value and then create the model fit:

rf\_with\_seed %>%

[set\_args](https://tidymodels.github.io/parsnip/reference/set_args.html)(mtry = 4) %>%

[set\_engine](https://tidymodels.github.io/parsnip/reference/set_engine.html)("ranger") %>%

[fit](https://www.rdocumentation.org/packages/generics/topics/fit)(mpg ~ ., data = mtcars)

#> parsnip model object

#>

#> Ranger result

#>

#> Call:

#> ranger::ranger(formula = formula, data = data, mtry = ~4, num.trees = ~2000, seed = ~63233, num.threads = 1, verbose = FALSE)

#>

#> Type: Regression

#> Number of trees: 2000

#> Sample size: 32

#> Number of independent variables: 10

#> Mtry: 4

#> Target node size: 5

#> Variable importance mode: none

#> Splitrule: variance

#> OOB prediction error (MSE): 5.57

#> R squared (OOB): 0.847

Or, using the randomForest package:

[set.seed](https://www.rdocumentation.org/packages/base/topics/Random)(56982)

rf\_with\_seed %>%

[set\_args](https://tidymodels.github.io/parsnip/reference/set_args.html)(mtry = 4) %>%

[set\_engine](https://tidymodels.github.io/parsnip/reference/set_engine.html)("randomForest") %>%

[fit](https://www.rdocumentation.org/packages/generics/topics/fit)(mpg ~ ., data = mtcars)

#> parsnip model object

#>

#>

#> Call:

#> randomForest(x = as.data.frame(x), y = y, ntree = ~2000, mtry = ~4, seed = ~63233)

#> Type of random forest: regression

#> Number of trees: 2000

#> No. of variables tried at each split: 4

#>

#> Mean of squared residuals: 5.52

#> % Var explained: 84.3

Note that the call objects show num.trees = ~2000. The tilde is the consequence of parsnip using quosures to process the model specification’s arguments.

Normally, when a function is executed, the function’s arguments are immediately evaluated. In the case of parsnip, the model specification’s arguments are not; the expression is captured along with the environment where it should be evaluated. That is what a quosure does.

parsnip uses these expressions to make a model fit call that is evaluated. The tilde in the call above reflects that the argument was captured using a quosure.