# Package: important (via r-universe)

May 30, 2026

**Title** Supervised Feature Selection

**Version** 0.2.1.9000

**Description** Interfaces for choosing important predictors in supervised regression, classification, and censored regression models. Permuted importance scores (Biecek and Burzykowski (2021) <doi:10.1201/9780429027192>) can be computed for 'tidymodels' model fits.

**License** MIT + file LICENSE

**URL** https://important.tidymodels.org/, https://github.com/tidymodels/important

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

**Depends** R (>= 4.1.0), recipes (>= 1.1.0)

**Imports** cli, desirability2 (>= 0.2.0), dplyr, filtro (>= 0.2.0), generics, ggplot2, hardhat (>= 1.4.1), purrr, rlang (>= 1.1.0), S7, tibble, tidyr, tune, vctrs, withr, workflows

**Suggests** censored, future, future.apply, mirai, modeldata, parsnip, ranger, spelling, survival, testthat (>= 3.0.0), yardstick

**Config/Needs/website** tidyverse/tidytemplate, tidymodels

**Config/testthat/edition** 3

**Config/usethis/last-upkeep** 2025-06-09

**Encoding** UTF-8

**Language** en-US

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Config/pak/sysreqs** cmake make libicu-dev libuv1-dev

**Repository** https://tidymodels.r-universe.dev

**Date/Publication** 2025-10-07 20:42:39 UTC

**RemoteUrl** https://github.com/tidymodels/important

**RemoteRef** HEAD

**RemoteSha** 4490a0c9be3b6214ec886b15ae95bb5c8b9750ef

# Contents



---

`autoplot.importance_perm` *Visualize importance scores*

---

## Description

Visualize importance scores

## Usage

```
## S3 method for class 'importance_perm'
autoplot(
  object,
  top = Inf,
  metric = NULL,
  eval_time = NULL,
  type = "importance",
  std_errs = stats::qnorm(0.95),
  ...
)
```

## Arguments

| | |
|---|---|
| `object` | A tibble of results from `importance_perm()`. |
| `top` | An integer for how many terms to show. To define importance when there are multiple metrics, the rankings of predictors are computed across metrics and the average rank is used. In the case of tied rankings, all the ties are included. |
| `metric` | A character vector or `NULL` for which metric to plot. By default, all metrics will be shown via facets. Possible options are the entries in `.metric` column of the object. |
| `eval_time` | For censored regression models, a vector of time points at which the survival probability is estimated. |
| `type` | A character value. The default is `"importance"` which shows the overall signal-to-noise ration (i.e., mean divided by standard error). Alternatively, `"difference"` shows the mean difference value with standard error bounds. |
| `std_errs` | The number of standard errors to plot (when `type = "difference"`). |
| `...` | Not used. |

## Value

A ggplot2 object.

## Examples

```
# Pre-computed results. See code at
system.file("make_imp_example.R", package = "important")

# Load the results
load(system.file("imp_examples.RData", package = "important"))

# A classification model with two classes and highly correlated predictors.
# To preprocess them, PCA feature extraction is used.
#
# Let's first view the importance in terms of the original predictor set
# using 50 permutations:

imp_orig

autoplot(imp_orig, top = 10)

# Now assess the importance in terms of the PCA components

imp_derv

autoplot(imp_derv)
autoplot(imp_derv, metric = "brier_class", type = "difference")
```

---

# importance_perm *Compute permutation-based predictor importance*

---

## Description

importance_perm() computes model-agnostic variable importance scores by permuting individual predictors (one at a time) and measuring how worse model performance becomes.

## Usage

```
importance_perm(
  wflow,
  data,
  metrics = NULL,
  type = "original",
  size = 500,
  times = 10,
  eval_time = NULL,
  event_level = "first"
)
```

## Arguments

| | |
|---|---|
| wflow | A fitted workflows::workflow(). |
| data | A data frame of the data passed to workflows::fit.workflow(), including the outcome and case weights (if any). |
| metrics | A yardstick::metric_set() or NULL. |
| type | A character string for which *level* of predictors to compute. A value of "original" (default) will return values in the same representation of data. Using "derived" will compute them for any derived features/predictors, such as dummy indicator columns, etc. |
| size | How many data points to predict for each permutation iteration. |
| times | How many iterations to repeat the calculations. |
| eval_time | For censored regression models, a vector of time points at which the survival probability is estimated. This is only needed if a dynamic metric is used, such as the Brier score or the area under the ROC curve. |
| event_level | A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". |

## Details

The function can compute importance at two different levels.

- The "original" predictors are the unaltered columns in the source data set. For example, for a categorical predictor used with linear regression, the original predictor is the factor column.
- "Derived" predictors are the final versions given to the model. For the categorical predictor example, the derived versions are the binary indicator variables produced from the factor version.

This can make a difference when pre-processing/feature engineering is used. This can help us understand *how* a predictor can be important

Importance scores are computed for each predictor (at the specified level) and each performance metric. If no metric is specified, defaults are used:

- Classification: yardstick::brier_class(), yardstick::roc_auc(), and yardstick::accuracy().
- Regression: yardstick::rmse() and yardstick::rsq().
- Censored regression: yardstick::brier_survival()

For censored data, importance is computed for each evaluation time (when a dynamic metric is specified).

By default, no parallelism is used to process models in **tune**; you have to opt-in.

**Using future to parallel process:**

You should install the package and choose your flavor of parallelism using the plan function. This allows you to specify the number of worker processes and the specific technology to use.

For example, you can use:

```
library(future)
plan(multisession, workers = 4)
```

and work will be conducted simultaneously (unless there is an exception; see the section below).

See `future::plan()` for possible options other than `multisession`.

**Using mirai to parallel process:**

To configure parallel processing with **mirai**, use the `mirai::daemons()` function. The first argument, `n`, determines the number of parallel workers. Using `daemons(0)` reverts to sequential processing.

The arguments `url` and `remote` are used to set up and launch parallel processes over the network for distributed computing. See `mirai::daemons()` documentation for more details.

## Value

A tibble with extra classes `"importance_perm"` and either `"original_importance_perm"` or `"derived_importance_perm"`. The columns are:

- `.metric` the name of the performance metric:
- `predictor`: the predictor
- `n`: the number of usable results (should be the same as `times`)
- `mean`: the average of the differences in performance. For each metric, larger values indicate worse performance (i.e., higher importance).
- `std_err`: the standard error of the differences.
- `importance`: the mean divided by the standard error.
- For censored regression models, an additional `.eval_time` column may also be included (depending on the metric requested).

## Examples

```
if (rlang::is_installed(c("modeldata", "recipes", "workflows", "parsnip"))) {
  library(modeldata)
  library(recipes)
  library(workflows)
  library(dplyr)
  library(parsnip)

  set.seed(12)
  dat_tr <-
    sim_logistic(250, ~ .1 + 2 * A - 3 * B + 1 * A *B, corr = .7) |>
    dplyr::bind_cols(sim_noise(250, num_vars = 10))

  rec <-
    recipe(class ~ ., data = dat_tr) |>
    step_interact(~ A:B) |>
    step_normalize(all_numeric_predictors()) |>
    step_pca(contains("noise"), num_comp = 5)

  lr_wflow <- workflow(rec, logistic_reg())
```

```
  lr_fit <- fit(lr_wflow, dat_tr)

  set.seed(39)
  orig_res <- importance_perm(lr_fit, data = dat_tr, type = "original",
                              size = 100, times = 3)
  orig_res

  set.seed(39)
  deriv_res <- importance_perm(lr_fit, data = dat_tr, type = "derived",
                               size = 100, times = 3)
  deriv_res
}
```

---

`step_predictor_best` *Supervised Feature Selection via Choosing the Top Predictors*

---

## Description

`step_predictor_best()` creates a *specification* of a recipe step that uses a single scoring function to measure how much each predictor is related to the outcome value. This step retains a proportion of the most important predictors, and this proportion can be tuned.

## Usage

```
step_predictor_best(
  recipe,
  ...,
  score,
  role = NA,
  trained = FALSE,
  prop_terms = 0.5,
  update_prop = TRUE,
  results = NULL,
  removals = NULL,
  skip = FALSE,
  id = rand_id("predictor_best")
)
```

## 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 this step. See `selections()` for more details. |
| `score` | The name of a single score function from the **filtro** package, such as `"imp_rf"` (for `filtro::score_imp_rf()`), etc. See the Details and Examples sections below. This argument *should be named* when 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. |
| prop_terms | The proportion of predictors that should be retained when ordered by overall desirability. A value of hardhat::tune() can also be used. |
| update_prop | A logical: should prop_terms be updated so that at least one predictor will be retained? |
| results | A data frame of score and desirability values for each predictor evaluated. These values are not determined until recipes::prep() is called. |
| removals | A character string that contains the names of predictors that should be removed. These values are not determined until recipes::prep() is called. |
| skip | A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() 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

### Scoring Functions:

As of version 0.2.0 of the **filtro** package, the following score functions are available:

- aov_fstat (documentation)
- aov_pval (documentation)
- cor_pearson (documentation)
- cor_spearman (documentation)
- gain_ratio (documentation)
- imp_rf (documentation)
- imp_rf_conditional (documentation)
- imp_rf_oblique (documentation)
- info_gain (documentation)
- roc_auc (documentation)
- sym_uncert (documentation)
- xtab_pval_chisq (documentation)
- xtab_pval_fisher (documentation)

Some important notes:

- Scores that are p-values are automatically transformed by **filtro** to be in the format -log10(pvalue) so that a p-value of 0.1 is converted to 1.0. For these, use the maximize() goal.
- Other scores are also transformed in the data. For example, the correlation scores given to the recipe step are in absolute value format. See the **filtro** documentation for each score.
- You can use some in-line functions using base R functions. For example, maximize(max(score_cor_spearman)).
- If a predictor cannot be computed for all scores, it is given a "fallback value" that will prevent it from being excluded for this reason.

This step can potentially remove columns from the data set. This may cause issues for subsequent steps in your recipe if the missing columns are specifically referenced by name. To avoid this, see the advice in the *Tips for saving recipes and filtering columns* section of recipes::selections.

**Ties:**

Note that `dplyr::slice_max()` with the argument `with_ties = TRUE` is used to select predictors. If there are many ties in overall desirability, the proportion selected can be larger than the value given to `prep_terms()`.

**Case Weights:**

Case weights can be used by some scoring functions. To learn more, load the **filtro** package and check the `case_weights` property of the score object (see Examples below). For a recipe, use one of the tidymodels case weight functions such as `hardhat::importance_weights()` or hardhat::frequency_weights, to assign the correct data type to the vector of case weights. A recipe will then interpret that class to be a case weight (and no other role). A full example is below.

**Tidy method:**

For a trained recipe, the `tidy()` method will return a tibble with columns `terms` (the predictor names), `id`, and columns for the estimated scores. The score columns are the raw values, before being filled with "safe values" or transformed.

There is an additional local column called `removed` that notes whether the predictor failed the filter and was removed after this step is executed.

## Value

An updated version of `recipe` with the new step added to the sequence of any existing operations. When you `tidy()` this step, a tibble::tibble is returned with columns `terms` and `id`:

**terms** character, the selectors or variables selected to be removed

**id** character, id of this step

Once trained, additional columns are included (see Details section).

## Examples

```
library(recipes)

rec <- recipe(mpg ~ ., data = mtcars) |>
  step_predictor_best(
    all_predictors(),
    score = "cor_spearman"
  )

prepped <- prep(rec)

bake(prepped, mtcars)

tidy(prepped, 1)
```

`step_predictor_desirability` *Supervised Multivariate Feature Selection via Desirability Functions*

## Description

`step_predictor_desirability()` creates a *specification* of a recipe step that uses one or more "score" functions to measure how much each predictor is related to the outcome value. These scores are combined into a composite value using user-specified *desirability* functions and a proportion of the most desirable predictors are retained.

## Usage

```
step_predictor_desirability(
  recipe,
  ...,
  score,
  role = NA,
  trained = FALSE,
  prop_terms = 0.5,
  update_prop = TRUE,
  results = NULL,
  removals = NULL,
  skip = FALSE,
  id = rand_id("predictor_desirability")
)
```

## 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 this step. See `selections()` for more details. |
| `score` | An object produced by `desirability2::desirability()` that uses one or more score functions from the **filtro** package. See the Details and Examples sections below. This argument *should be named* when 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. |
| `prop_terms` | The proportion of predictors that should be retained when ordered by overall desirability. A value of `hardhat::tune()` can also be used. |
| `update_prop` | A logical: should `prop_terms` be updated so that at least one predictor will be retained? |
| `results` | A data frame of score and desirability values for each predictor evaluated. These values are not determined until `recipes::prep()` is called. |

| | |
|---|---|
| removals | A character string that contains the names of predictors that should be removed. These values are not determined until recipes::prep() is called. |
| skip | A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() 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 recipe step can compute one or more scores and conduct a simultaneous selection of the top predictors using *desirability functions*. These are functions that, for some type of goal, translate the score's values to a scale of [0, 1], where 1.0 is the best result and 0.0 is unacceptable. Once we have these for each score, the overall desirability is computed using the geometric mean of the individual desirabilities. See the examples in desirability2::d_overall() and desirability2::d_max().

To define desirabilities, use desirability2::desirability() function to define *goals* for each score and pass that to the recipe in the score argument.

### Scoring Functions:

As of version 0.2.0 of the **filtro** package, the following score functions are available:

- aov_fstat (documentation)
- aov_pval (documentation)
- cor_pearson (documentation)
- cor_spearman (documentation)
- gain_ratio (documentation)
- imp_rf (documentation)
- imp_rf_conditional (documentation)
- imp_rf_oblique (documentation)
- info_gain (documentation)
- roc_auc (documentation)
- sym_uncert (documentation)
- xtab_pval_chisq (documentation)
- xtab_pval_fisher (documentation)

Some important notes:

- Scores that are p-values are automatically transformed by **filtro** to be in the format -log10(pvalue) so that a p-value of 0.1 is converted to 1.0. For these, use the maximize() goal.
- Other scores are also transformed in the data. For example, the correlation scores given to the recipe step are in absolute value format. See the **filtro** documentation for each score.
- You can use some in-line functions using base R functions. For example, maximize(max(cor_spearman)).
- If a predictor cannot be computed for all scores, it is given a "fallback value" that will prevent it from being excluded for this reason.

This step can potentially remove columns from the data set. This may cause issues for subsequent steps in your recipe if the missing columns are specifically referenced by name. To avoid this, see the advice in the *Tips for saving recipes and filtering columns* section of recipes::selections.

**Ties:**

Note that `dplyr::slice_max()` with the argument `with_ties = TRUE` is used to select predictors. If there are many ties in overall desirability, the proportion selected can be larger than the value given to `prep_terms()`.

**Case Weights:**

Case weights can be used by some scoring functions. To learn more, load the **filtro** package and check the `case_weights` property of the score object (see Examples below). For a recipe, use one of the tidymodels case weight functions such as `hardhat::importance_weights()` or hardhat::frequency_weights, to assign the correct data type to the vector of case weights. A recipe will then interpret that class to be a case weight (and no other role). A full example is below.

**Tidy method:**

For a trained recipe, the `tidy()` method will return a tibble with columns `terms` (the predictor names), `id`, columns for the estimated scores, and the desirability results. The score columns are the raw values, before being filled with "safe values" or transformed.

The desirability columns will have the same name as the scores with an additional prefix of `.d_`. The overall desirability column is called `.d_overall`.

There is an additional local column called `removed` that notes whether the predictor failed the filter and was removed after this step is executed.

## Value

An updated version of `recipe` with the new step added to the sequence of any existing operations. When you `tidy()` this step, a tibble::tibble is returned with columns `terms` and `id`:

**terms** character, the selectors or variables selected to be removed

**id** character, id of this step

Once trained, additional columns are included (see Details section).

## References

Derringer, G. and Suich, R. (1980), Simultaneous Optimization of Several Response Variables. *Journal of Quality Technology*, 12, 214-219.

https://desirability2.tidymodels.org/reference/inline_desirability.html

## See Also

`desirability2::desirability()`

## Examples

```
library(recipes)
library(desirability2)

if (rlang::is_installed("modeldata")) {
 # The `ad_data` has a binary outcome column ("Class") and mostly numeric
 # predictors. For these, we score the predictors using an analysis of
 # variance model where the predicor is the outcome and the outcome class
 # defines the groups.
 # There is also a single factor predictor ("Genotype") and we'll use
 # Fisher's Exact test to score it. NOTE that for scores using hypothesis
 # tests, the -log10(pvalue) is returned so that larger values are more
 # important.

 # The score_* objects here are from the filtro package. See Details above.
 goals <-
  desirability(
   maximize(xtab_pval_fisher),
   maximize(aov_pval)
  )

 example_data <- modeldata::ad_data
 rec <-
  recipe(Class ~ ., data = example_data) |>
  step_predictor_desirability(
   all_predictors(),
   score = goals,
   prop_terms = 1 / 2
  )
 rec

 # Now evaluate the predictors and rank them via desirability:
 prepped <- prep(rec)
 prepped

 # Use the tidy() method to get the results:
 predictor_scores <- tidy(prepped, number = 1)
 mean(predictor_scores$removed)
 predictor_scores

 # ------------------------------------------------------------------------

 # Case-weight example: use the hardhat package to create the appropriate type
 # of case weights. Here, we'll increase the weights for the minority class and
 # add them to the data frame.

 library(hardhat)

 example_weights <- example_data
 weights <- ifelse(example_data$Class == "Impaired", 5, 1)
 example_weights$weights <- importance_weights(weights)
```

```
# To see if the scores can use case weights, load the filtro package and
# check the `case_weights` property:

library(filtro)

score_xtab_pval_fisher@case_weights
score_aov_pval@case_weights

# The recipe will automatically find the case weights and will
# not treat them as predictors.
rec_wts <-
 recipe(Class ~ ., data = example_weights) |>
 step_predictor_desirability(
  all_predictors(),
  score = goals,
  prop_terms = 1 / 2
 ) |>
 prep()
rec_wts

predictor_scores_wts <-
 tidy(rec_wts, number = 1) |>
 select(terms, .d_overall_weighted = .d_overall)

library(dplyr)
library(ggplot2)

# The selection did not substantially change with these case weights
full_join(predictor_scores, predictor_scores_wts, by = "terms") |>
 ggplot(aes(.d_overall, .d_overall_weighted)) +
 geom_abline(col = "darkgreen", lty = 2) +
 geom_point(alpha = 1 / 2) +
 coord_fixed(ratio = 1) +
 labs(x = "Unweighted", y = "Class Weighted")
}
```

---

`step_predictor_retain` *Supervised Feature Selection via A Single Filter*

---

## Description

`step_predictor_retain()` creates a *specification* of a recipe step that uses a logical statement that includes one or more scoring functions to measure how much each predictor is related to the outcome value. This step retains the predictors that pass the logical statement.

## Usage

```
step_predictor_retain(
  recipe,
```

```
  ...,
  score,
  role = NA,
  trained = FALSE,
  results = NULL,
  removals = NULL,
  skip = FALSE,
  id = rand_id("predictor_retain")
)
```

## 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 this step. See selections() for more details. |
| score | A valid R expression that produces a logical result. The equation can contain the names of one or more score functions from the **filtro** package, such as filtro::score_imp_rf(), filtro:: score_roc_auc(). See the Details and Examples sections below. This argument *should be named* when 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. |
| results | A data frame of score and desirability values for each predictor evaluated. These values are not determined until recipes::prep() is called. |
| removals | A character string that contains the names of predictors that should be removed. These values are not determined until recipes::prep() is called. |
| skip | A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() 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

The score should be valid R syntax that produces a logical result and should not use external data. The list of variables that can be used is in the section below.

### Scoring Functions:

As of version 0.2.0 of the **filtro** package, the following score functions are available:

- aov_fstat (documentation)
- aov_pval (documentation)
- cor_pearson (documentation)
- cor_spearman (documentation)
- gain_ratio (documentation)
- imp_rf (documentation)

- `imp_rf_conditional` (documentation)
- `imp_rf_oblique` (documentation)
- `info_gain` (documentation)
- `roc_auc` (documentation)
- `sym_uncert` (documentation)
- `xtab_pval_chisq` (documentation)
- `xtab_pval_fisher` (documentation)

Some important notes:

- Scores that are p-values are automatically transformed by **filtro** to be in the format `-log10(pvalue)` so that a p-value of 0.1 is converted to 1.0. For these, use the `maximize()` goal.
- Other scores are also transformed in the data. For example, the correlation scores given to the recipe step are in absolute value format. See the **filtro** documentation for each score.
- You can use some in-line functions using base R functions. For example, `maximize(max(score_cor_spearman))`.
- If a predictor cannot be computed for all scores, it is given a "fallback value" that will prevent it from being excluded for this reason.

This step can potentially remove columns from the data set. This may cause issues for subsequent steps in your recipe if the missing columns are specifically referenced by name. To avoid this, see the advice in the *Tips for saving recipes and filtering columns* section of recipes::selections.

**Case Weights:**

Case weights can be used by some scoring functions. To learn more, load the **filtro** package and check the `case_weights` property of the score object (see Examples below). For a recipe, use one of the tidymodels case weight functions such as `hardhat::importance_weights()` or hardhat::frequency_weights, to assign the correct data type to the vector of case weights. A recipe will then interpret that class to be a case weight (and no other role). A full example is below.

**Tidy method:**

For a trained recipe, the `tidy()` method will return a tibble with columns `terms` (the predictor names), `id`, and columns for the estimated scores. The score columns are the raw values, before being filled with "safe values" or transformed.

There is an additional local column called `removed` that notes whether the predictor failed the filter and was removed after this step is executed.

## Value

An updated version of `recipe` with the new step added to the sequence of any existing operations. When you `tidy()` this step, a tibble::tibble is returned with columns `terms` and `id`:

**terms** character, the selectors or variables selected to be removed

**id** character, id of this step

Once trained, additional columns are included (see Details section).

## Examples

```
library(recipes)

rec <- recipe(mpg ~ ., data = mtcars) |>
  step_predictor_retain(
    all_predictors(),
    score = cor_pearson >= 0.75 | cor_spearman >= 0.75
  )

prepped <- prep(rec)

bake(prepped, mtcars)

tidy(prepped, 1)
```

# Index