Extension and Customization

Table of contents


```
library(modelsummary)
```

```
`modelsummary` 2.0.0 uses `tinytable` as its default table-drawing
  package. Learn more at:

    https://vincentarelbundock.github.io/tinytable/

  You can revert to `kableExtra` for one session:

    options(modelsummary_factory_default = 'kableExtra')

  Or change the backend persistently:

    config_modelsummary(factory_default = 'gt')
```

	(1)
coef1	1.000
	(3.142)
coef2	2.000
	(2.718)
coef3	3.000
	(1.414)
stat1	blah
stat2	blah blah

1 Unsupported models: `modelsummary_list`

The simplest way to summarize an unsupported model is to create a `modelsummary_list` object. This approach is super flexible, but it requires manual intervention, and it can become tedious if you need to summarize many models. The next section shows how to add formal support for an unsupported model type.

A `modelsummary_list` is a list with two element that conform to the `broom` package specification: `tidy` and `glance`. `tidy` is a data.frame with at least three columns: `term`, `estimate`, and `std.error`. `glance` is a data.frame with only a single row, and where each column will be displayed at the bottom of the table in the goodness-of-fit section. Finally, we wrap those two elements in a list and assign it a `modelsummary_list` class:

```
ti <- data.frame(
  term = c("coef1", "coef2", "coef3"),
  estimate = 1:3,
  std.error = c(pi, exp(1), sqrt(2)))

gl <- data.frame(
  stat1 = "blah",
  stat2 = "blah blah")

mod <- list(
  tidy = ti,
  glance = gl)
class(mod) <- "modelsummary_list"

modelsummary(mod)
```

2 Unsupported models: `glance` and `tidy`

`modelsummary` relies on two functions from the `broom` package to extract model information: `tidy` and `glance`. If `broom` doesn't support the type of model you are trying to summarize, `modelsummary` won't support it out of the box. Thankfully, it is extremely easy to add support for most models using custom methods.

For example, models produced by the `MCMCglmm` package are not currently supported by `broom`. To add support, you simply need to create a `tidy` and a `glance` method:

```
# load packages and data
library(modelsummary)
library(MCMCglmm)
data(PlodiaPO)

# add custom functions to extract estimates (tidy) and goodness-of-fit (glance) information
tidy.MCMCglmm <- function(x, ...) {
    s <- summary(x, ...)
    ret <- data.frame(
      term      = row.names(s$solutions),
      estimate  = s$solutions[, 1],
      conf.low  = s$solutions[, 2],
      conf.high = s$solutions[, 3])
    ret
}

glance.MCMCglmm <- function(x, ...) {
    ret <- data.frame(
      dic = x$DIC,
      n   = nrow(x$X))
    ret
}

# estimate a simple model
model <- MCMCglmm(PO ~ 1 + plate, random = ~ FSfamily, data = PlodiaPO, verbose=FALSE, pr=TRU

# summarize the model
modelsummary(model, statistic = 'conf.int')
```

Three important things to note.

First, the methods are named `tidy.MCMCglmm` and `glance.MCMCglmm` because the model object I am trying to summarize is of class `MCMCglmm`. You can find the class of a model by running:

`class(model)`.

Second, both of the methods include the ellipsis `...` argument.

Third, in the example above we used the `statistic = 'conf.int'` argument. This is because the `tidy` method produces `conf.low` and `conf.high` columns. In most cases, users will define `std.error` column in their custom `tidy` methods, so the `statistic` argument will need to be adjusted.

If you create new `tidy` and `glance` methods, please consider contributing them to `broom` so that the rest of the community can benefit from your work: https://github.com/tidymodels/broom

3 Modifying information: `tidy_custom` and `glance_custom`

Users may want to include more information than is made available by the default extractor function. For example, models produced by the `MASS::polr` do not produce p values by default, which means that we cannot use the `stars=TRUE` argument in `modelsummary`. However, it is possible to extract this information by using the `lmtest::coeftest` function. To include such custom information, we will define new `glance_custom` and `tidy_custom` methods.

We begin by estimating a model with the `MASS::polr`:

```
library(MASS)

mod_ordinal <- polr(as.ordered(gear) ~ mpg + drat, data = mtcars)

get_estimates(mod_ordinal)
```

```
  term     estimate  std.error conf.level   conf.low  conf.high   statistic
1  3|4 13.962948761 4.04107300       0.95  5.6851860 22.2407116  3.45525774
2  4|5 16.898937342 4.39497069       0.95  7.8962480 25.9016267  3.84506258
3  mpg -0.008646682 0.09034201       0.95 -0.1916706  0.1708667 -0.09571053
4 drat  3.949431923 1.30665144       0.95  1.6191505  6.8457246  3.02255965
  df.error      p.value component s.value group
1       28 0.0017706303     alpha     9.1
2       28 0.0006356348     alpha    10.6
3       28 0.9244322098      beta     0.1
4       28 0.0053120619      beta     7.6
```

The `get_estimates` function shows that our default extractor does *not* produce a `p.value` column. As a result, setting `stars=TRUE` in `modelsummary` will produce an error.

We know that the `MASS::polr` produces an object of class `polr`:

```
class(mod_ordinal)
```

```
[1] "polr"
```

To extract more (custom) information from a model of this class, we thus define a method called `tidy_custom.polr` which returns a data.frame with two columns: `term` and `p.value`:

```
tidy_custom.polr <- function(x, ...) {
  s <- lmtest::coeftest(x)
  out <- data.frame(
    term = row.names(s),
    p.value = s[, "Pr(>|t|)"])
  out
}
```

When this method is defined, `modelsummary` can automatically extract p values from all models of this class, and will now work properly with `stars=TRUE`:

```
modelsummary(mod_ordinal, stars = TRUE)
```

4 New information: `tidy_custom` and `glance_custom`

Sometimes users will want to include information that is not supplied by those functions. A pretty easy way to include extra information is to define new `glance_custom` and `tidy_custom` methods. To illustrate, we estimate two linear regression models using the `lm` function:

```
library(modelsummary)

mod <- list()
mod[[1]] <- lm(hp ~ mpg + drat, mtcars)
mod[[2]] <- lm(wt ~ mpg + drat + am, mtcars)
```

In `R`, the `lm` function produces models of class "lm":

```
class(mod[[1]])
```

```
[1] "lm"
```

	(1)
3|4	13.963**
	(4.041)
4|5	16.899***
	(4.395)
mpg	−0.009
	(0.090)
drat	3.949**
	(1.307)
Num.Obs.	32
AIC	51.1
BIC	57.0
RMSE	3.44

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Let's say you would like to print the dependent variable for each model of this particular class. All you need to do is define a new method called `glance_custom.lm`. This method should return a data.frame (or tibble) with 1 row, and 1 column per piece of information you want to display. For example:

```
glance_custom.lm <- function(x, ...) {
    dv <- as.character(formula(x)[2])
    out <- data.frame("DV" = dv)
    return(out)
}
```

Now, let's customize the body of the table. The `vcov` argument already allows users to customize uncertainty estimates. But imagine you want to override the *coefficient estimates* of your "lm" models. Easy! All you need to do is define a `tidy_custom.lm` method which returns a data.frame (or tibble) with one column called "term" and one column called "estimate".

Here, we'll substitute estimates by an up/down-pointing triangles which represents their signs:

```
tidy_custom.lm <- function(x, ...) {
    s <- summary(x)$coefficients
```

	(1)	(2)
(Intercept)		
	(55.415)	(0.728)
mpg		
	(1.792)	(0.019)
drat		
	(20.198)	(0.245)
am		
		(0.240)
Num.Obs.	32	32
R2	0.614	0.803
R2 Adj.	0.588	0.782
AIC	337.9	46.4
BIC	343.7	53.7
Log.Lik.	−164.940	−18.201
F	23.100	38.066
RMSE	41.91	0.43
DV	hp	wt

```
    out <- data.frame(
      term = row.names(s),
      estimate = ifelse(s[,1] > 0, ' ', ' '))
    return(out)
}
```

After you define the `glance_custom` and `tidy_custom` methods, `modelsummary` will automatically display your customized model information:

```
modelsummary(mod)
```

Note that you can define a `std.error` column in `tidy_custom.lm` to replace the uncertainty estimates instead of the coefficients.

5 Customization: New model class

An even more fundamental way to customize the output would be to completely bypass `modelsummary`'s extractor functions by assigning a new class name to your model. For example,

```
# estimate a linear model
mod_custom <- lm(hp ~ mpg + drat, mtcars)

# assign it a new class
class(mod_custom) <- "custom"

# define tidy and glance methods
tidy.custom <- function(x, ...) {
  data.frame(
    term = names(coef(x)),
    estimate = letters[1:length(coef(x))],
    std.error = seq_along(coef(x))
  )
}

glance.custom <- function(x, ...) {
  data.frame(
    "Model" = "Custom",
    "nobs" = stats:::nobs.lm(x)
  )
}

# summarize
modelsummary(mod_custom)
```

Warning: When defining new `tidy` and `glance` methods, it is important to include an ellipsis argument (`...`).

Note that in the `glance.custom()` method, we called `stats:::nobs.lm()` instead of the default `stats::nobs()` method, because the latter know does not know where to dispatch models of our new "custom" class. Being more explicit solves the problem.

An alternative would be to set a new class that inherits from the previous one, and to use a global option to set `broom` as the default extractor function (otherwise `modelsummary` will use its standard `lm` extractors by inheritance):

	(1)
(Intercept)	a
	(1.000)
mpg	b
	(2.000)
drat	c
	(3.000)
Num.Obs.	32
Model	Custom

```
options(modelsummary_get = "broom")
class(mod_custom) <- c("custom", "lm")
```

6 Customization: `modelsummary_list`

Another flexible way to customize model output is to use `output = "modelsummary_list"`. With this output option, `modelsummary()` returns a list with two elements: `tidy` contains parameter estimates, standard errors, etc., and `glance` contains model statistics such as the AIC. For example,

```
mod <- lm(hp ~ mpg + drat, mtcars)
mod_list <- modelsummary(mod, output = "modelsummary_list")
mod_list$tidy
```

```
         term   estimate std.error  statistic df.error      p.value s.value
1 (Intercept) 278.515455 55.414866  5.0260061       29 2.359726e-05    15.4
2         mpg  -9.985499  1.791837 -5.5727709       29 5.172030e-06    17.6
3        drat  19.125752 20.197756  0.9469246       29 3.515013e-01     1.5
  group conf.low conf.high
1            NA        NA
2            NA        NA
3            NA        NA
```

```
mod_list$glance
```

```
       aic      bic r.squared adj.r.squared     rmse nobs        F    logLik
1 337.8809 343.7438 0.6143611     0.5877653 41.90687   32 23.09994 -164.9404
```

	Wald	Kenward
(Intercept)	−5.159	−5.159
	6.409 (0.428)	6.409 (0.680)
drat	7.045	7.045
	1.736 (<0.001) ***	1.736 (0.086) +
Num.Obs.	32	32
R2 Marg.	0.402	0.402
R2 Cond.	0.440	0.440
AIC	188.7	188.7
BIC	194.6	194.6
ICC	0.1	0.1
RMSE	4.28	4.28

Both `tidy` and `glance` can now be customized, and the updated model can be passed back to modelsummary using `modelsummary(mod_list)`. All information that is displayed in the table is contained in `mod_list`, so this pattern allows for very flexible adjustments of output tables.

A useful example for this pattern concerns mixed models using `lme4`. Assume we want to compare the effect of using different degrees-of-freedom adjustments on the significance of the coefficients. The models have identical parameter estimates, standard errors, and model fit statistics - we only want to change the p-values. We use the `parameters` package to compute the adjusted p-values.

```
library("lme4")
mod <- lmer(mpg ~ drat + (1 | am), data = mtcars)
mod_list <- modelsummary(mod, output = "modelsummary_list", effects = "fixed")
# create a copy, where we'll change the p-values
mod_list_kenward <- as.list(mod_list)
mod_list_kenward$tidy$p.value <- parameters::p_value_kenward(mod)$p

modelsummary(list("Wald" = mod_list, "Kenward" = mod_list_kenward),
             statistic = "{std.error} ({p.value}) {stars}")
```