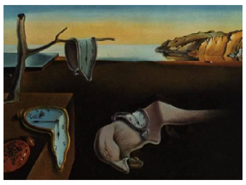


Journal of Statistical Software

July 2022, Volume 103, Issue 1. *doi: 10.18637/jss.v103.i01*

modelsummary: Data and Model Summaries in R

Vincent Arel-Bundock
Université de Montréal

Abstract

modelsummary is a package to summarize data and statistical models in R. It supports over one hundred types of models out-of-the-box, and allows users to report the results of those models side-by-side in a table, or in coefficient plots. It makes it easy to execute common tasks such as computing robust standard errors, adding significance stars, and manipulating coefficient and model labels. Beyond model summaries, the package also includes a suite of tools to produce highly flexible data summary tables, such as dataset overviews, correlation matrices, (multi-level) cross-tabulations, and balance tables (also known as "Table 1"). The appearance of the tables produced by **modelsummary** can be customized using external packages such as **kableExtra**, **gt**, **flextable**, or **huxtable**; the plots can be customized using **ggplot2**. Tables can be exported to many output formats, including HTML, LaTeX, Text/Markdown, Microsoft Word, Powerpoint, Excel, RTF, PDF, and image files. Tables and plots can be embedded seamlessly in **rmarkdown**, **knitr**, or `Sweave` dynamic documents. The **modelsummary** package is designed to be simple, robust, modular, and extensible.

Keywords: tables, data summary, model summary, crosstab, regression table, coefficient plot, R.

1. Introduction

Data analysts often communicate their results using regression tables, coefficient plots, descriptive summaries, balance tables, crosstabs, or correlation tables. Creating these tables and plots can be a time-consuming and aggravating process. The **modelsummary** package eases this burden by allowing users to create a wide range of publication-ready data and model summaries under one roof, using a simple, consistent, and powerful set of functions.

The **modelsummary** package follows four key design principles:

1. *Simplicity.* All of the package's functions use a simple and consistent user interface. The number of arguments per function is limited. Each argument is well documented and accompanied by copious examples.

Function	Description
Data summary functions	
`datasummary`	Extremely flexible tool to draw (multi-level) crosstabs and statistical summaries of all kinds
`datasummary_skim`	Quick overview of a dataset, with variables' mean, standard deviation, minimum, median, maximum, number of unique values, share of missing observations, and an inline histogram
`datasummary_balance`	Balance table (also known as "Table 1") with the mean, standard deviation, and differences in means across treatment groups
`datasummary_correlation`	Correlation tables with support for various correlation methods
`datasummary_crosstab`	(Multi-level) cross-tabulations
`datasummary_df`	Turns a `data.frame` or a `tibble` into a deeply customizable, print-ready table
Model summary functions	
`modelsummary`	Tables to summarize one statistical model or multiple models side-by-side. It makes it easy to rename, reorder, or omit coefficients and goodness-of-fit statistics; add significance stars; compute "robust" standard errors or confidence intervals; insert extra information, notes, or captions; and customize the display of numeric entries.
`modelplot`	Coefficient plots ("dot and whisker") which can be customized with the **ggplot2** package

Table 1: A list of the main functions in the **modelsummary** package.

2. *Flexibility.* **modelsummary** supports over one hundred types of statistical models out-of-the-box, and makes it very easy for users and developers to add support for new models. Tables can be exported to many output formats, including HTML, LaTeX, Text/Markdown, Microsoft Word, Powerpoint, Excel, RTF, PDF, and image files. They can be customized extensively with R (R Core Team 2022) packages such as **kableExtra** (Zhu 2021), **gt** (Iannone, Cheng, and Schloerke 2022), **flextable** (Gohel 2022), and **huxtable** (Hugh-Jones 2022). Plots can be customized using **ggplot2** (Wickham 2016).

3. *Robustness.* Each new release of **modelsummary** is checked against an extensive series of tests which cover nearly all of the code base. Each function inspects user input carefully to preempt problems and return informative error messages.

4. *Community.* **modelsummary** does not try to reinvent the wheel, but rather builds on the work of the R community to empower users. Instead of implementing its own engine to extract the results of different estimation routines, **modelsummary** relies on the **broom** (Robinson, Hayes, and Couch 2022) and **easystats** (Lüdecke, Ben-Shachar, Patil, and Makowski 2020) projects. Instead of implementing its own engine to generate HTML or LaTeX code, **modelsummary** supports several of the most popular table-making packages in the R ecosystem. This gives users access to cutting edge features, improves maintainability for the **modelsummary** developers, and ensures that functionality will automatically improve as upstream packages develop.

The **modelsummary** package includes two main families of functions. The first produces *data* summaries. The second produces *model* summaries. Table 1 gives an overview of the package's main functions. These functions are all designed with a simple and consistent user interface. This makes **modelsummary** easy to learn for users who seek an integrated tool for scientific communication. The package **modelsummary** is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=modelsummary`. The next section illustrates the benefits of the package with practical examples.

2. Illustrations

To illustrate how to use the **modelsummary** package, we will consider two datasets. The first holds information about penguins near Palmer Station in Antarctica (Horst, Hill, and Gorman 2020). The second tracks the results of *RuPaul's Drag Race*, a televised reality competition series (Miller 2022). Both datasets are available as standalone R packages and at the *RDatasets* archive, a website which hosts over 1700 free datasets in CSV format: `https://vincentarelbundock.github.io/Rdatasets`.

2.1. Data summaries

The first step in our data analysis is to load the **modelsummary** package and the `penguins` dataset. Then, we take a subset of columns and rename them:

```
R> library("modelsummary")
R> data("penguins", package = "palmerpenguins")
R> variables <- c(
+    "flipper_length_mm" = "Flipper",
+    "bill_length_mm"    = "Bill",
+    "body_mass_g"       = "Body Mass",
+    "sex"               = "Sex",
+    "island"            = "Island",
+    "species"           = "Species")
R> penguins <- setNames(penguins[, names(variables)], variables)
```

To begin the data exploration, we use `datasummary_skim`. This function was heavily inspired by the **skimr** package (Waring, Quinn, McNamara, Arino de la Rubia, Zhu, and Ellis 2022). It gives a high-level overview of the dataset, with key descriptive statistics and an inline histogram. This code produces Table 2:

```
R> datasummary_skim(penguins)
```

By default, `datasummary_skim` only summarizes continuous variables, but this behavior can be altered with the `type` argument.

After "skimming" the data, it is often useful to report descriptive statistics for subgroups of the sample. In an experimental setting, for example, the analyst may want to verify that covariates are balanced between treatment arms, and might like to highlight differences in means for important variables. This kind of table is colloquially referred to as "Table 1" or "Balance Table." To create such a table, we use the `datasummary_balance` command.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
Flipper	56	1	200.9	14.1	172.0	197.0	231.0	
Bill	165	1	43.9	5.5	32.1	44.5	59.6	
Body Mass	95	1	4201.8	802.0	2700.0	4050.0	6300.0	

Table 2: A summary table produced by the `datasummary_skim` function.

		Female (N=165)		Male (N=168)			
		Mean	Std. Dev.	Mean	Std. Dev.	Diff. in Means	Std. Error
Flipper		197.4	12.5	204.5	14.5	7.1	1.5
Bill		42.1	4.9	45.9	5.4	3.8	0.6
Body Mass		3862.3	666.2	4545.7	787.6	683.4	79.9
		N	Pct.	N	Pct.		
Island	Biscoe	80	48.5	83	49.4		
	Dream	61	37.0	62	36.9		
	Torgersen	24	14.5	23	13.7		
Species	Adelie	73	44.2	73	43.5		
	Chinstrap	34	20.6	34	20.2		
	Gentoo	58	35.2	61	36.3		

Table 3: A balance table produced by the `datasummary_balance` function.

	Flipper	Bill	Body Mass
Flipper	1	.	.
Bill	0.66	1	.
Body Mass	0.87	0.60	1

Table 4: A correlation table produced by the `datasummary_correlation` function.

The first argument of `datasummary_balance` is a one-sided formula which identifies the groups that we want to compare. The output of this command is shown in Table 3:

```
R> datasummary_balance(~ Sex, data = penguins)
```

Under the hood, `datasummary_balance` uses the **estimatr** package (Blair, Cooper, Coppock, Humphreys, and Sonnet 2022) to estimate standard errors around the differences in means. As a result, the function can produce estimates that automatically take into account clusters, weights, or blocked experimental designs.

To explore the relationships between numeric variables, it is often useful to create a correlation table. The `datasummary_correlation` makes this easy. This function can report different kinds of correlations (e.g., Pearson, Kendall, Spearman), and allows users to supply their own methods. This code produces Table 4:

```
R> datasummary_correlation(penguins)
```

The `datasummary_crosstab` can help us explore the relationships between categorical variables. The first argument of this function is a two-sided formula, where the left-hand side

		Female			Male			
Island		Adelie	Chinstrap	Gentoo	Adelie	Chinstrap	Gentoo	All
Biscoe	N	22	0	58	22	0	61	168
	% row	13.1	0.0	34.5	13.1	0.0	36.3	100.0
Dream	N	27	34	0	28	34	0	124
	% row	21.8	27.4	0.0	22.6	27.4	0.0	100.0
Torgersen	N	24	0	0	23	0	0	52
	% row	46.2	0.0	0.0	44.2	0.0	0.0	100.0
All	N	73	34	58	73	34	61	344
	% row	21.2	9.9	16.9	21.2	9.9	17.7	100.0

Table 5: A cross-tabulation table produced by the `datasummary_crosstab` function.

represents the row variable and the right-hand side identifies the column variables. For example, to draw a cross-tab of the `Island` variable against the `Species` variable, we could type:

```
R> datasummary_crosstab(Island ~ Species, data = penguins)
```

The `datasummary_crosstab` function can also produce multi-level crosstabs. To achieve this, we use the asterisk (`*`) as a nesting operator. The code below produces Table 5, which shows counts and shares of penguins by `Island`, `Sex`, and `Species`:

```
R> datasummary_crosstab(Island ~ Sex * Species, data = penguins)
```

If Tables 2–5 do not fill our particular needs, we can turn to the `datasummary` function. `datasummary` is a general purpose tool built on top of the **tables** package (Murdoch 2020). It can make crosstabs and data summaries, and it can display the output of virtually any function in the R language.

The `datasummary` function builds a table by reference to a two-sided formula: the left side defines rows and the right side defines columns. The terms of the formula represent variables and the functions that we want to apply to those variables. Terms linked by a `+` sign are displayed one after the other, in the order in which they enter the formula.

For example, if we want to display the `Flipper` and `Body Mass` variables on separate rows, and the variables' means and standard deviations in distinct columns, we type:

```
R> datasummary(Flipper + `Body Mass` ~ Mean + SD, data = penguins)
```

The command above produces Table 6a. Two aspects of this code are noteworthy. First, when a variable name includes spaces (e.g., `Body Mass`), we can enclose it in backticks in the `datasummary` formula.

Second, the `Mean` and `SD` terms of the formula above are convenience functions supplied by the **modelsummary** package. These functions call the corresponding `mean` and `sd` functions in base R, but set `na.rm = TRUE` by default. Since the `Flipper` and `Body Mass` variables of the `penguins` dataset include missing observations, using the base R `mean` and `sd` functions in the `datasummary` formula — with their `na.rm = FALSE` default — would produce a table with empty cells. The convenience functions offered by `modelsummary` include: `Mean`, `Median`, `Min`, `Max`, `N`, `Ncol`, `NPercent`, `NUnique`, `P0`, `P25`, `P50`, `P75`, `P100`, `SD`, `Var`.

Crucially, users are *not* limited to the summary functions supplied by **modelsummary** itself. Indeed, `datasummary` can display the output of *any* arbitrary R function that accepts a vector and returns a single numeric or character value. For instance, we could define a new `Range` function which displays the minimum and maximum values of variables in square brackets:

```
R> Range <- function(x) {
+    sprintf("[%s, %s]", min(x, na.rm = TRUE), max(x, na.rm = TRUE))
+  }
```

Then, we can add our new `Range` function to the formula, and pivot the previous table with functions on rows (left side of the formula) and variables in columns (right side of the formula). This code produces Table 6b:

```
R> datasummary(Mean + SD + Range ~ Flipper + Bill, data = penguins)
```

`datasummary` also allows users to compute statistics for different subgroups of the data. To achieve this, we use the asterisk (`*`) as a nesting operator, connecting a factor variable to a function: `Sex * Mean` will calculate the mean of each variable, for each value of the `Sex` variable. This code produces Table 6c:

```
R> datasummary(Flipper + Bill ~ Sex * Mean + SD, data = penguins)
```

The `*` nesting operator can be "distributed" across terms with parentheses. For instance, `Sex * (Mean + SD)` will calculate both the means and standard deviations of our variables, for each value of `Sex`. This code produces Table 6d:

```
R> datasummary(Flipper + Bill ~ Sex * (Mean + SD), data = penguins)
```

Thanks to the **tables** package, `datasummary` supports a series of "pseudo-functions" which can be used in formulae. When a formula includes factor or character variables, we can use `N` to count the number of observations in each category; `Percent()` to compute percentages; and the number `1` to represent a "total" category. These pseudo-functions can be useful to create highly customized cross-tabs. For example, this code produces Table 6e:

```
R> datasummary(Island + 1 ~ N + Percent(), data = penguins)
```

Here again, we can use the `*` nesting operators to count penguins in subgroups. Inserting `Sex * Species` in the formula will compute statistics for each sex/species subgroup. `Heading` is another useful pseudo-function which allows us to rename columns and rows. This code produces Table 6f:

```
R> datasummary(Sex * Species ~ Heading("#") * N + Heading("%") * Percent(),
+    data = penguins)
```

In sum, the **modelsummary** package includes a convenient set of "templates" to produce common tables: skim, balance, correlation, and cross-tabs. Thanks to the work of Murdoch (2020), users can also leverage a powerful formula syntax to build highly customized tables using the `datasummary` function. The **modelsummary** website includes more details and several additional examples: `https://vincentarelbundock.github.io/modelsummary/`.

	Mean	SD
Flipper	200.92	14.06
Body Mass	4201.75	801.95

(a) `Flipper + `Body Mass``$\sim$ `Mean + SD`

	Flipper	Bill
Mean	200.92	43.92
SD	14.06	5.46
Range	[172, 231]	[32.1, 59.6]

(b) `Mean + SD + Range` $\sim$ `Flipper + Bill`

	Female	Male	
	Mean	Mean	SD
Flipper	197.36	204.51	14.06
Bill	42.10	45.85	5.46

(c) `Flipper + Bill` $\sim$ `Sex * Mean + SD`

	Female		Male	
	Mean	SD	Mean	SD
Flipper	197.36	12.50	204.51	14.55
Bill	42.10	4.90	45.85	5.37

(d) `Flipper + Bill` $\sim$ `Sex * (Mean + SD)`

Island	N	Percent
Biscoe	168	48.84
Dream	124	36.05
Torgersen	52	15.12
All	344	100.00

(e) `Island + 1` $\sim$ `N + Percent()`

Sex	Species	#	%
Female	Adelie	73	21.22
	Chinstrap	34	9.88
	Gentoo	58	16.86
Male	Adelie	73	21.22
	Chinstrap	34	9.88
	Gentoo	61	17.73

(f) `Sex * Species` $\sim$ `Heading("#") * N + Heading("%") * Percent()`

Table 6: Six tables drawn by the `datasummary` function. The subtable captions show the two-sided formulae used to create the tables.

	Model 1
(Intercept)	7.045
	(0.169)
minichalw	−1.023
	(0.404)
missc	0.298
	(0.308)
episode	−0.309
	(0.024)
Num.Obs.	1286
R2	0.120
R2 Adj.	0.118
AIC	6634.3
BIC	6660.1
Log.Lik.	−3312.163
F	58.284

Table 7: A table produced by the `modelsummary` function.

2.2. Model summaries

The second family of functions in the **modelsummary** package are designed to help users communicate the results of statistical models. The `modelsummary` function produces tables to summarize one or many models side-by-side. The `modelplot` function draws coefficient plots (dot and whisker). Both functions are highly customizable, and they support over one hundred types of statistical models out-of-the-box.

To illustrate, we load the `dragracer` dataset, which includes information about contestants' performance in each episode of *RuPaul's Drag Race*:

```
R> data("rpdr_contep", package = "dragracer")
R> dragracer <- rpdr_contep
```

Columns of this dataset include a contestant's rank in each episode (`rank`), Mini Challenge winners (`minichalw`), Miss Congeniality titles (`missc`), a season identifier (`season`), and the position of each episode within a season (`episode`).

Our analysis begins by using the `lm` function to estimate a linear model with `rank` as dependent variable. We then use the `modelsummary` function to summarize the findings. The output of this code is displayed in Table 7:

```
R> mod <- lm(rank ~ minichalw + missc + episode, data = dragracer)
R> modelsummary(mod, gof_omit = "RMSE")
```

Now suppose we want to compare three models: the simple linear model, a linear mixed effects model with random slopes by season, and a generalized (Poisson) linear mixed effects. We also want those models to be clearly labelled in the table. To do this, we use the `lmer` and `glmer` functions from the **lme4** package to estimate mixed effects models, and we store everything in a named list:

	LM	LMER	GLMER
(Intercept)	7.073	7.079	1.994
	(0.166)	(0.269)	(0.046)
minichalw	−1.004	−0.702	−0.156
	(0.403)	(0.401)	(0.064)
episode	−0.309	−0.348	−0.071
	(0.024)	(0.024)	(0.004)
missc		0.356	0.073
		(0.302)	(0.041)
SD (Intercept)		0.761	0.146
SD (Observations)		3.112	1.000
Num.Obs.	1286	1286	1286
R2	0.119		
R2 Adj.	0.118		
R2 Marg.		0.142	0.310
R2 Cond.		0.190	0.403
AIC	6633.3	6609.8	6669.5
BIC	6653.9	6640.8	6695.3
ICC		0.1	0.1
Log.Lik.	−3312.633		
F	86.963		
RMSE		3.10	3.11

Table 8: A table produced by the `modelsummary` function.

```
R> library("lme4")
R> models <- list(
+    "LM" = lm(rank ~ minichalw + episode, data = dragracer),
+    "LMER" = lmer(rank ~ minichalw + missc + episode + (1 | season),
+      data = dragracer),
+    "GLMER" = glmer(rank ~ minichalw + missc + episode + (1 | season),
+      data = dragracer, family = poisson))
```

Named or unnamed lists of models can be fed directly to `modelsummary`. We can also use the `align = "lddd"` argument to left-align the first column and dot-align the other ones (each character represents a column).[1] This code produces Table 8:

```
R> modelsummary(models, output = "latex", align = "lddd")
```

We can improve and customize this table by altering the argument values of the `modelsummary` function. Assign nicer labels to the coefficients of our models by passing a named vector to `coef_rename`. Change the number of digits with `fmt`. Set the type of (classical, robust, or clustered) standard errors to display for each model using the `vcov` argument. Feed a regular expression to `gof_omit` to omit all statistics from the bottom panel, except the number of

[1]Dot alignment is only available for LaTeX and PDF output, since it relies on the **siunitx** LaTeX package.

observations and the standard errors identifier.[2] Change the width of the confidence intervals with `conf_level`. Drop the uncertainty statistics in parentheses by setting `statistic = NULL`. Define a caption with the `title` argument. Add a note to the bottom of the table with the `notes` argument.

A useful feature of the `modelsummary` function is that it can leverage the **glue** package to accept interpreted string literals (Hester 2022). Users can thus define exactly how they want coefficient or uncertainty estimates to be displayed in their tables, by using the **glue** curly braces syntax. For example, to display a confidence interval in brackets next to the estimate, we can set `estimate = "{estimate} [{conf.low}, {conf.high}]"`. In this expression, the `estimate`, `conf.low`, and `conf.high` values follow the naming convention established by the **broom** package. Users can see a list of available values by applying the `modelsummary::get_estimates` function to one of their models.

Putting everything together gives us this code, which produces Table 9:

```
R> coef_labels <- c(
+    "minichalw"   = "Mini Challenge Winner",
+    "missc"       = "Miss Congeniality",
+    "episode"     = "Episode",
+    "(Intercept)" = "Constant")
R> modelsummary(models,
+    coef_rename = coef_labels,
+    fmt         = 1,
+    vcov        = list("robust", "classical", "classical"),
+    gof_omit    = "^(?!Num|Std)",
+    conf_level  = 0.99,
+    statistic   = NULL,
+    title       = "A \\code{modelsummary} table about RuPaul's Drag Race.",
+    notes       = "Source: dragracer package (Miller 2020).",
+    estimate    = "{estimate} [{conf.low}, {conf.high}]")
```

Many analysts prefer to report the results of statistical models in graphical form, using coefficient plots. The `modelplot` function has the same user interface as `modelsummary`; both functions accept the same kinds of objects and the same arguments. This code produces the **ggplot2** graphic in Figure 1:

```
R> modelplot(models, coef_rename = coef_labels)
```

As stated above, the `modelplot` and `modelsummary` are designed with very similar user interfaces. One useful argument that those functions share is `vcov`, which allows users to display various types of uncertainty estimates: heteroskedasticity-robust, panel-corrected, bootstrap, HAC, etc. When the analyst sets `vcov`, corrected standard errors and confidence intervals are computed automatically using the **sandwich** (Zeileis, Köll, and Graham 2020) and **lmtest** (Zeileis and Hothorn 2002) packages.

For example, if we want to compare the confidence intervals associated with six different variance estimators, we can create a named list of this form:

[2]The `"^(?!Num|Std)"` regular expression uses a "negative lookahead" to match every term in the goodness-of-fit section of the table *except* those that start with "Num" or "Std". For information on Perl-compatible regular expressions, see the `grep` function documentation.

	LM	LMER	GLMER
Constant	7.1 [6.6, 7.6]	7.1 [6.4, 7.8]	2.0 [1.9, 2.1]
Mini Challenge Winner	−1.0 [−1.7, −0.3]	−0.7 [−1.7, 0.3]	−0.2 [−0.3, 0.0]
Episode	−0.3 [−0.4, −0.2]	−0.3 [−0.4, −0.3]	−0.1 [−0.1, −0.1]
Miss Congeniality		0.4 [−0.4, 1.1]	0.1 [0.0, 0.2]
SD (Intercept)		0.8	0.1
SD (Observations)		3.1	1.0
Num.Obs.	1286	1286	1286
Std.Errors	Robust	Classical	Classical

Source: dragracer package (Miller 2020).

Table 9: A `modelsummary` table about RuPaul's Drag Race.

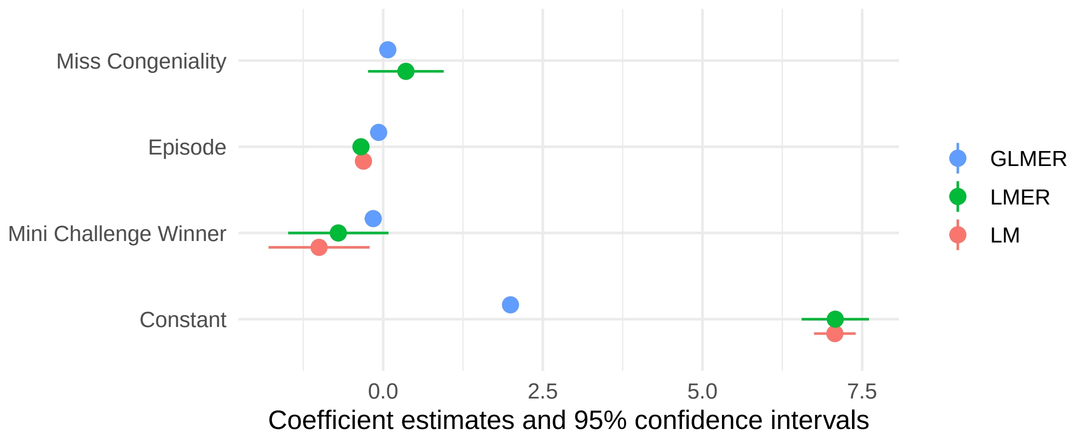


Figure 1: A coefficient plot produced by the `modelplot` function.

```
R> vcov_list <- list(
+     "Classical"            = "classical",
+     "Robust"               = "robust",
+     "Clustered"            = ~ episode,
+     "Andrews' kernel HAC" = sandwich::kernHAC,
+     "Newey-West"           = sandwich::NeweyWest,
+     "Bootstrap"            = sandwich::vcovBS)
```

This list can include strings like `"classical"`, `"robust"`, `"stata"`, or `"HC3"` which describe the type of standard errors to use. It can include one-sided formulae which are passed to the `sandwich::vcovCL` function to request clustered standard errors. Finally, the `vcov` list can include variance-covariance matrices, or functions that return such matrices.

In the current application, we want to compare six different confidence intervals for a single model. Therefore, we create a named list which includes the same model six times. Then, we call the `modelplot` function, and customize the plot's appearance with **ggplot2**'s `scale_color_brewer`, `guides`, and `theme` functions. The result of this code is shown in Figure 2:

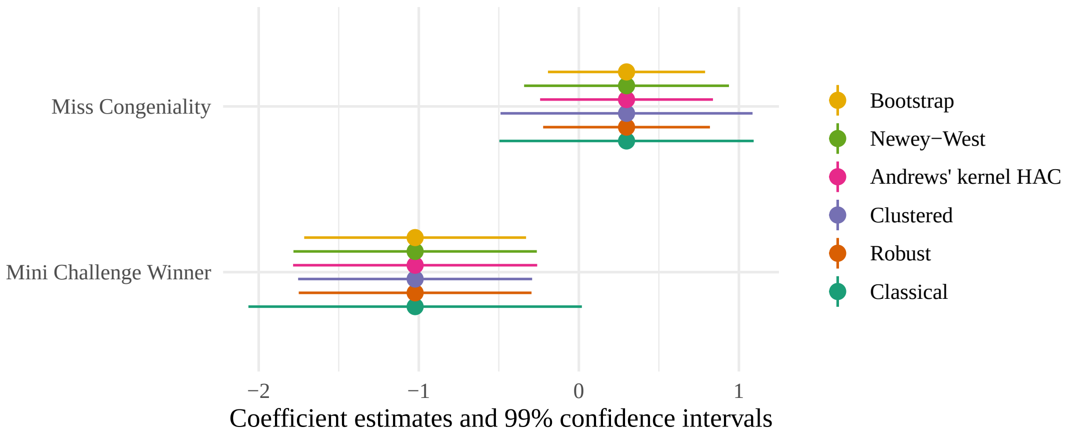


Figure 2: A coefficient plot produced by the `modelplot` function.

```
R> library("ggplot2")
R> mod_list <- lapply(vcov_list, function(x) mod)
R> modelplot(mod_list, vcov = vcov_list, conf_level = 0.99,
+    coef_omit = "Intercept|episode", coef_rename = coef_labels) +
+    scale_color_brewer(palette = "Dark2") +
+    guides(color = guide_legend(reverse = TRUE)) +
+    theme(text = element_text(family = "Times"))
```

Note that when the analyst supplies a single model but multiple `vcov` entries, the `modelsummary` and `modelplot` functions will automatically "recycle" the model by repeating it as many times as necessary. However, the resulting table or plot would not be as nicely labeled as Figure 2.

By default, when users add layers to a **ggplot2** plot using the `+` operator, new geoms will be added *on top* of the default point range. We can add **ggplot2** geoms in the background of a plot (e.g., a vertical line at 0) using the `background` argument.

2.3. Customizing tables

One of the major benefits of **modelsummary** is that the tables it produces are compatible with four of the most popular table-making packages in the R ecosystem: **kableExtra**, **gt**, **flextable**, **huxtable**. By default, the functions produce **kableExtra** tables, which means that we can use the `|>` operator to pass our tables to that packages' functions for customization.

For example, to identify two models as `"Gaussian"` and one as `"Poisson"`, we can use the `add_header_above` function. To color and bold one of the rows, we can use `row_spec`. This code produces Table 10:

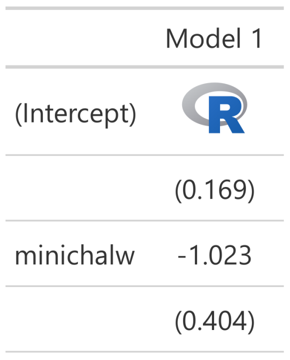
```
R> library("kableExtra")
R> modelsummary(models, stars = TRUE, gof_omit = ".*",
+    coef_rename = coef_labels) |>
+    add_header_above(c(" " = 1, "Gaussian" = 2, "Poisson" = 1)) |>
+    row_spec(3, background = "pink", bold = TRUE)
```

To use a different package to customize tables, we simply change `modelsummary`'s `output`

	Gaussian		Poisson
	LM	LMER	GLMER
Constant	7.073***	7.079***	1.994***
	(0.166)	(0.269)	(0.046)
Mini Challenge Winner	**−1.004***	**−0.702+**	**−0.156***
	(0.403)	(0.401)	(0.064)
Episode	−0.309***	−0.348***	−0.071***
	(0.024)	(0.024)	(0.004)
Miss Congeniality		0.356	0.073+
		(0.302)	(0.041)
SD (Intercept)		0.761	0.146
SD (Observations)		3.112	1.000

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

Table 10: A regression table produced by the `modelsummary` function and customized with the **kableExtra** package.



	Model 1
(Intercept)	
	(0.169)
minichalw	-1.023
	(0.404)

Figure 3: A portion of an HTML regression table with embedded image, produced by combining the **modelsummary** and **gt** packages.

argument. For example, this code uses **gt** to insert a web-hosted image in an HTML regression table, as shown in Figure 3:

```
R> library("gt")
R> img <- "https://www.R-project.org/logo/Rlogo.svg"
R> modelsummary(mod, output = "gt") |>
+    text_transform(locations = cells_body(columns = 2, rows = 1),
+      fn = function(x) web_image(url = img))
```

2.4. Saving and exporting

The tables produced by the `datasummary` and `modelsummary` families of functions can be saved and exported to a wide array of formats, including HTML, LaTeX, Text/Markdown, Microsoft Word, Powerpoint, Excel, RTF, PDF, and image files.

Users can save tables directly to file by setting the `output` argument to a valid file path. The desired output format is then inferred from the file extension. For example:

```
R> modelsummary(mod, output = "table.tex")
R> modelsummary(mod, output = "table.html")
R> modelsummary(mod, output = "table.docx")
```

Tables can also be echoed to the R console in raw form:

```
R> modelsummary(mod, output = "markdown")
R> modelsummary(mod, output = "html")
R> modelsummary(mod, output = "latex")
R> modelsummary(mod, output = "latex_tabular")
```

Tables can be returned as objects for further processing using external packages, by setting the `output` argument to `"kableExtra"`, `"gt"`, `"flextable"`, or `"huxtable"`.

When compiling R Markdown documents, **modelsummary** infers the target format and sets the `output` argument automatically, such that no further user intervention is required. An R Markdown document with simple calls like `modelsummary(mod)` or `datasummary_skim(mtcars)` will typically compile to PDF or HTML without modification.

3. Package internals

The **modelsummary** package is designed to be modular and extensible. Figure 4 gives a schematic representation of its internal structure.

The left branch of Figure 4 gives an overview of the code used to summarize statistical models. The box at the top represents the user interface. The `modelsummary` and `modeplot` functions are harmoniously designed, in the sense that they accept almost all the same arguments. When these functions are called, user inputs are validated with the **checkmate** package, a dependency-free argument checking tool which returns helpful error messages on failure (Lang 2017).

The next step is to extract results from model objects. The `get_estimates` function tries to extract estimates with `broom::tidy`, and then falls back to `parameters::parameters`. The `get_gof` function tries to extract goodness-of-fit statistics with `broom::glance`, and then falls back to `performance::performance`. The order of priority between **broom** (Robinson *et al.* 2022), **performance** (Lüdecke, Ben-Shachar, Patil, Waggoner, and Makowski 2021), and **parameters** (Lüdecke *et al.* 2020) can be modified by changing the `modelsummary_get` global option. The `get_estimates` and `get_gof` functions were designed for internal use by `modelsummary`, but they are exported to the namespace for users who need a versatile and standardized way to extract raw results from over a hundred distinct object types.

Once statistical results are extracted, **modelsummary** transforms the data to suit the desired output format: multiple models are merged; coefficients and statistics are renamed, omitted, and/or sorted; numeric values are rounded; robust standard errors are computed; significance stars are added; etc.

Finally, **modelsummary** infers the output format that users need by looking at the `output` argument, and it feeds the data to a "table factory" function. By default, the factory builds

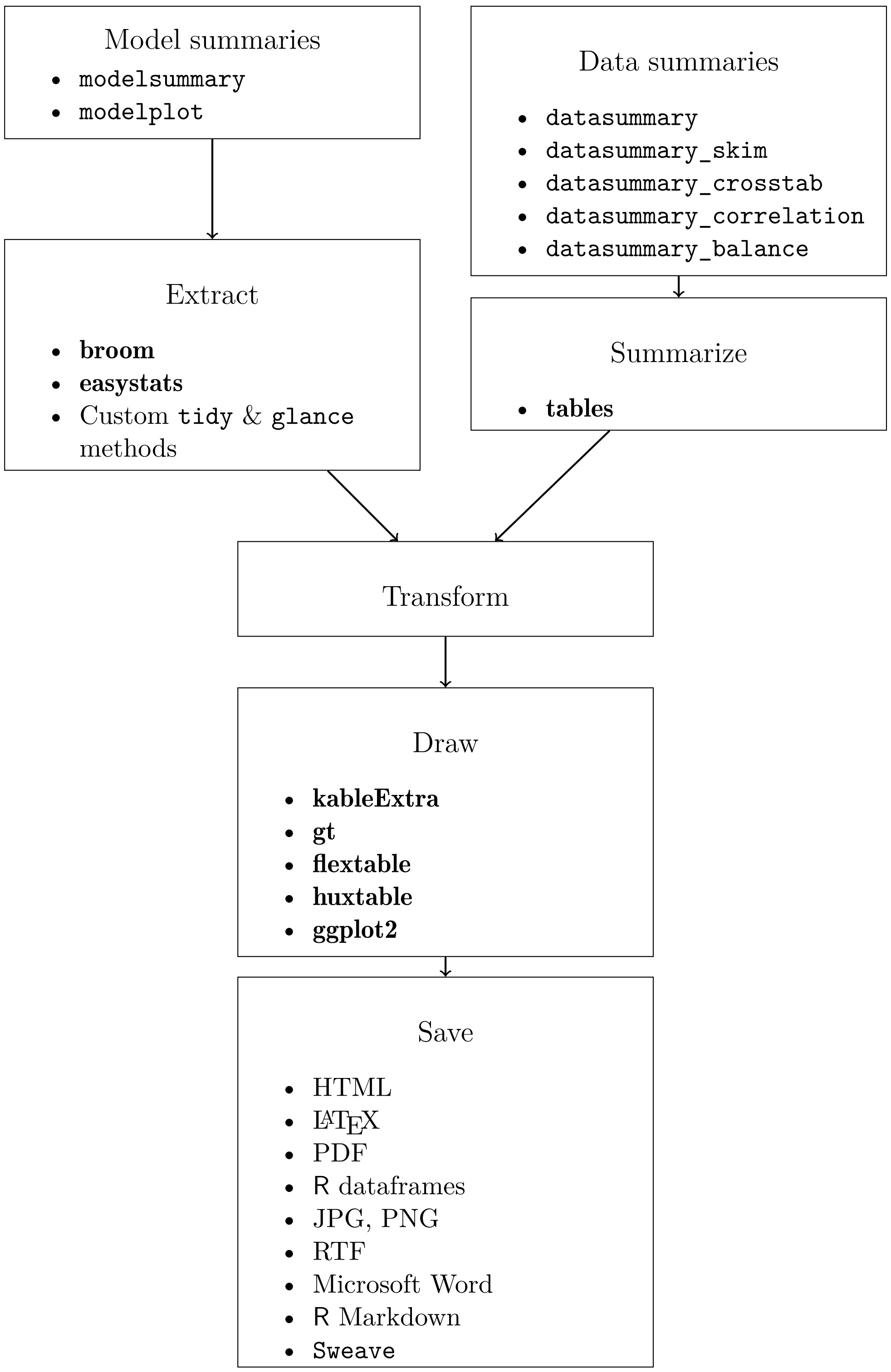


Figure 4: The internal structure of the **modelsummary** package is modular and extensible.

HTML, LaTeX, and Markdown tables using **kableExtra**, but users can request a different object type by changing the `output` argument. If the user specifies a valid file path as output (e.g., `output = "file.tex"`), an appropriate table factory is selected based on the file extension, and the table is saved to file automatically.

In the right branch of Figure 4, we see that functions in the *data* summary family go through a similar process. However, instead of extracting results from statistical models, the `datasummary` functions use the **tables** package to compute statistical summaries from a dataset. Then, internal functions from the **modelsummary** package transform the results slightly, before feeding them to a table factory.

The key benefit of the modular design described in Figure 4 is that both families of functions are funnelled to the same table factories. This means that the *model* and *data* summary functions can have very similar user interfaces, and that the resulting tables can be customized and saved in exactly the same ways.

Another benefit of the modular approach is that **modelsummary** is very easy to extend. As of version 0.9.4, tables can be exported using four different table-making packages. Adding support for new table factories is a trivial task, often requiring less than 50 lines of code. The next section shows that it is also very easy to add support for new models and statistics.

4. Extending and customizing modelsummary

There are many ways to extend and customize the **modelsummary** package or the outputs of its functions. Here, we consider a few: supporting new statistical models, transforming the numerical results of a model, and adding custom statistics.

4.1. Support for new statistical models

The **modelsummary** package supports over one hundred statistical models out-of-the-box. To add support for a new model type, users can define two S3 methods (`tidy` and `glance`) which conform to the specification described on the **broom** package website: https://broom.tidymodels.org/

The `tidy` method is a function called `tidy.CLASSNAME`, which accepts a statistical model of class '`CLASSNAME`', and returns a data frame with one row per term/coefficient, and distinct columns with standardized names: `term`, `estimate`, `std.error`, `statistic`, `p.value`, `conf.low`, `conf.high`. For example, a minimal `tidy` method to extract results from a model of class '`lm`' could be:

```
R> tidy.lm <- function(x, ...) {
+    out <- data.frame(
+      term = names(coef(x)),
+      estimate = coef(x),
+      std.error = sqrt(vcov(x)))
+    return(out)
+  }
```

The `glance` method is a function called `glance.CLASSNAME`, which accepts a statistical model of class '`CLASSNAME`', and returns a data frame with a single row, and one model characteristic per column. For example, a minimal `glance` method to extract information from a model of class ‘`lm`’ could be:

```
R> glance.lm <- function(x, ...) {
+    out <- data.frame(
+      nobs = nobs(x),
+      r.squared = summary(x)$r.squared)
+    return(out)
+  }
```

The minimalist methods given above are superfluous because `modelsummary` already supports `lm` models by default. But they illustrate the general point: As soon as valid `tidy.CLASSNAME` and `glance.CLASSNAME` methods are defined, `modelsummary` automatically supports all models of the relevant class. When those methods are defined, calling `modelsummary(mod)` should just work.

Users who define `tidy` and `glance` methods to support new statistical models are strongly encouraged to give back to the community by submitting their methods for inclusion in the **broom** package. Interested readers are also encouraged to visit the **parameters** and **performance** websites to learn how they can support the work of those who develop these essential infrastructure packages.

Two other extension strategies deserve a note. First, since **broom** supports the objects produced by the `coeftest` function of the **lmtest** package (Zeileis and Hothorn 2002), any model supported by that package will automatically be supported by **modelsummary**. All that users need to do is apply `coeftest` to the model before feeding the result to `modelsummary`. Second, **modelsummary** allows users to summarize arbitrary data by storing them in a list of class `modelsummary_list`. See details on the **modelsummary** website: `https://vincentarelbundock.github.io/modelsummary/`.

4.2. Transformations

Analysts often wish to transform their model estimates before reporting them. Some transformations are so common that packages like **broom** offer built-in machinery to execute them. For example, it is common to exponentiate the coefficient estimates produced by a logistic regression model, and the `broom::tidy` function includes an `exponentiate` argument to do just that. This argument can be supplied directly to `modelsummary`, which will use the ellipsis `"..."` argument to push through the request to `broom::tidy`. This code estimates a logistic regression model and draws a table with exponentiated coefficients and confidence intervals:

```
R> mod_logit <- glm(vs ~ hp + mpg, data = mtcars, family = binomial)
R> modelsummary(mod_logit, exponentiate = TRUE, statistic = "conf.int")
```

For deeper customization, package **modelsummary** offers an alternative mechanism: defining `tidy_custom` and `glance_custom` S3 methods. These methods follow the same specification as the `tidy` and `glance` methods described in Section 4.1. When they are defined, their output will override the default values extracted from the models being summarized.

For example, to draw Table 11, which uses arrows to display the signs of a linear model’s coefficient estimates, we can call this code:

	Model 1
(Intercept)	↑
minichalw	↓
missc	↑
episode	↓

Table 11: The content of a `modelsummary` table can be customized by defining a `tidy_custom` or a `glance_custom` method.

```
R> tidy_custom.lm <- function(x, ...) {
+    out <- data.frame(
+      "term" = names(coef(x)),
+      "estimate" = ifelse(coef(x) > 0, "↑", "↓"))
+    return(out)
+  }
R> modelsummary(mod, statistic = NULL, gof_omit = ".*")
```

4.3. Adding custom statistics

The mechanism described in the previous section can also be used to add custom statistics to a table. For example, many researchers want to adjust the *p* values that they report to account for multiple comparisons. The `p.adjust` function in R can calculate many types of corrected *p* values, adjusted following the methods of Bonferroni (1936); Holm (1979), and others.

To adjust the *p* values of a linear regression model, we define a new `tidy_custom.lm` method which returns a data frame with one column called `term` and another column with the new statistic we wish to report. We can also add statistics to the bottom of the table by defining a `glance_custom.lm` method which returns a data frame with one row and one piece of information per column. For instance, if the analyst plans to conduct 10 tests with the `minichalw` coefficient, they could write:

```
R> tidy_custom.lm <- function(x, ...) {
+    out <- broom::tidy(x)
+    out$bonferroni <- p.adjust(out$p.value, n = 10, method = "bonferroni")
+    out$holm <- p.adjust(out$p.value, n = 10, method = "holm")
+    return(out)
+  }
R> glance_custom.lm <- function(x, ...) {
+    out <- data.frame("Num.Comparisons" = "10", "Model" = class(x)[1])
+    return(out)
+  }
```

Then, we call `modelsummary` and use **glue** strings in the `statistic` argument to label the different *p* values. To focus on the `minichalw` variable, we use the `coef_map` argument which allows users to select, reorder, and rename a subset of variables. This code produces Table 12:

	Model 1
Mini Challenge	−1.0227
	p = 0.0114
	p (Bonferroni) = 0.1143
	p (Holm) = 0.0914
Num.Obs.	1286
Num.Comparisons	10
Model	lm

Table 12: A regression table with adjusted *p* values.

```
R> modelsummary(mod,
+    statistic = c("p = {p.value}",
+                  "p (Bonferroni) = {bonferroni}",
+                  "p (Holm) = {holm}"),
+    coef_map = c("minichalw" = "Mini Challenge"),
+    gof_omit = "R2|IC|Log|F|RMSE",
+    fmt = 4)
```

5. Conclusion and comparison

The table-making ecosystem in R is a crowded field. Several packages can produce *model* summaries, including trailblazers like **xtable** and modern alternatives like **gtsummary** (Dahl, Scott, Roosen, Magnusson, and Swinton 2019; Sjoberg, Curry, Hannum, Larmarange, Whiting, and Zabor 2022). Even more packages can produce *data* summaries, such as the excellent **skimr** (Waring *et al.* 2022), **tables** (Murdoch 2020), **table1** (Rich 2021), **tableone** (Yoshida and Bartel 2022), and **furniture** (Barrett and Brignone 2017) packages.

modelsummary is a useful addition to this thriving ecosystem. First, **modelsummary** introduces a powerful set of functions which can produce both *data* and *model* summaries, using a simple and consistent user interface. Second, by using both **broom** and **easystats** to extract estimation results, **modelsummary** supports more model types than any other R package released to date. Third, **modelsummary** makes it easier than most other packages to execute common tasks such as displaying clustered standard errors, or deeply customizing the display of results (via the `tidy_custom` and `glance_custom` mechanisms). Fourth, **modelsummary** can export tables using several specialized table-making packages, which makes its outputs infinitely customizable. Fifth, the modular design of the package ensures that it remains easy to maintain and extend; adding support for new statistical models and table formats should be relatively painless. Finally, the **modelsummary** package is extensively tested, well documented, and accompanied by a rich website full of concrete examples: `https://vincentarelbundock.github.io/modelsummary/`

modelsummary has two main drawbacks. Since the package does not draw tables itself, users sometimes need to call external functions to change the appearance of their outputs. For deep customization, many analysts will thus choose to learn an external package like **kableExtra** or **gt**. Relatedly, **modelsummary** requires dependencies to extract model results and draw tables. Those dependencies do not pose a serious threat to the future of the package because they are actively-maintained, have large user-bases, and because **modelsummary**'s modular design would allow its developers to pivot easily if one upstream package were deprecated. Nevertheless, users who do not want to onboard too many dependencies may want to consider alternative packages.

With respect to *data* summaries, **modelsummary** strikes a balance between two general approaches. The first approach is exemplified by the **tables** package, on which **modelsummary** relies heavily. **tables** offers a general purpose tool to create summary tables which can be exported to LaTeX, HTML, and **kableExtra** formats. The `datasummary` family of functions in **modelsummary** build on that foundation by (a) offering convenient "templates" for common use-cases such as balance tables or crosstabs; (b) expanding the range of output formats; and (c) integrating **tables**'s formula syntax in a wider ecosystem with a harmonized user interface.

The second approach for data summaries can be seen in packages such as **skimr**, **tableone**, **table1**, and **furniture**. These packages overlap with some of the functions introduced in this article; indeed, they have directly inspired many of **modelsummary**'s own features. These packages tend to offer a series of hard-coded "templates" to execute common tasks such as building balance tables or dataset overviews. However, they do not offer the same kind of formula language to create highly customized tables; they can export to fewer output formats; they offer less flexibility to customize the appearance of tables; and they do not share a user interface with functions which can summarize statistical models in addition to raw data.

With respect to *model* summaries, there are several alternative packages to consider. The first are the popular **stargazer** (Hlavac 2022) and **texreg** (Leifeld 2013). Both of these packages offer integrated solutions to extract, reshape, and display statistical results in HTML, LaTeX, and text formats. By handling all of these steps themselves, they obviate the need to call on dependencies. One drawback of this approach is complexity: to master the `stargazer` function, analysts must sift through the documentation for 85 distinct arguments. Similarly, the `texreg` function has 48 distinct arguments. Another cost is flexibility: although package developers have made tremendous efforts to allow customization, the `stargazer` and `texreg` functions remain less flexible and powerful than dedicated table-drawing packages like **kableExtra** or **gt**. Finally, although **texreg** offers a package-specific mechanism to support new models, the **modelsummary** approach is arguably easier, more general, and standardized (see Section 4.1). The **stargazer** package also seems to pose particular challenges for maintainability and development: the whole package appears to consist of a single 7000 lines long function, with a large number of hard-coded variables.

huxtable is a general-purpose table-making package which can also extract and display results from statistical models. **modelsummary** supports this package as one of its output formats, by setting: `output="huxtable"`. This means that **huxtable** functions can be used to customize the appearance of a **modelsummary** table. When used as a standalone regression table-maker, the main drawbacks of **huxtable** are that its results customization functions are less flexible than **modelsummary**'s, and that the HTML and LaTeX code it generates is not designed to be human-readable or hand-editable.

memisc is a package which can summarize the results of statistical models (Elff 2021). It supports fewer model types than **modelsummary**, but produces good-looking text, LaTeX, and HTML tables. This package's main focus area is the analysis of survey data, and it offers many utilities to handle labeled data and to overcome survey-specific challenges, such as displaying clustered variance estimates.

Finally, the **gtsummary** package is emblematic of a new R generation of packages in this space, similar in spirit to **modelsummary**: It uses **broom** to extract results from model objects, it can export to several table-making packages, and it includes many functions to produce data summaries (e.g., "Table 1"). By default, **gtsummary** produces tables that look like those we typically see in peer reviewed journals in the life sciences. If users do not like **modelsummary**, this would be a good place to look next.

In sum, whereas several packages offer functionality that overlaps, **modelsummary** offers an attractive combination of features, thanks to its simplicity, flexibility, robustness, and its strategy to leverage the great work the R community.

Acknowledgments

I thank the authors of **broom** (Robinson *et al.* 2022), **checkmate** (Lang 2017), **generics** (Wickham, Kuhn, and Vaughan 2022), **glue** (Hester 2022), **insight** (Lüdecke, Waggoner, and Makowski 2019), **kableExtra** (Zhu 2021), **parameters** (Lüdecke *et al.* 2020), **performance** (Lüdecke *et al.* 2021), and **tables** (Murdoch 2020).

I thank the Journal of Statistical Software editors, two anonymous reviewers, Garrick Aden-Buie, Lysle Boller, Karl Broman, Benjamin Elbers, Michael E. Flynn, Joachim Gassen, Andrew Heiss, Nick C. Huntington-Klein, Daniel Lüdecke, Grant McDermott, Marco Mendoza Avina, Richard Paquin Morel, Indrajeet Patil, Moritz P. Schwarz, Felix Turbanisch, Florence Vallée-Dubois, Lukas Wallrich, Achim Zeiles.

References

Barrett TS, Brignone E (2017). "Furniture for Quantitative Scientists." *The R Journal*, **9**(2), 142–148. doi:10.32614/rj-2017-037.

Blair G, Cooper J, Coppock A, Humphreys M, Sonnet L (2022). ***estimatr**: Fast Estimators for Design-Based Inference.* R package version 0.30.6, URL https://CRAN.R-project.org/package=estimatr.

Bonferroni C (1936). "Teoria Statistica delle Classi e Calcolo Delle Probabilita." *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, **8**, 3–62.

Dahl DB, Scott D, Roosen C, Magnusson A, Swinton J (2019). ***xtable**: Export Tables to LaTeX or HTML.* R package version 1.8-4, URL https://CRAN.R-project.org/package=xtable.

Elff M (2021). ***memisc**: Management of Survey Data and Presentation of Analysis Results.* R package version 0.99.30.7, URL https://CRAN.R-project.org/package=memisc.

Gohel D (2022). ***flextable**: Functions for Tabular Reporting.* R package version 0.7.2, URL https://CRAN.R-project.org/package=flextable.

Hester J (2022). ***glue**: Interpreted String Literals.* R package version 1.6.2, URL https://CRAN.R-project.org/package=glue.

Hlavac M (2022). ***stargazer**: Well-Formatted Regression and Summary Statistics Tables.* Central European Labour Studies Institute (CELSI), Bratislava, Slovakia. R package version 5.2.3, URL https://CRAN.R-project.org/package=stargazer.

Holm S (1979). "A Simple Sequentially Rejective Multiple Test Procedure." *Scandinavian Journal of Statistics*, pp. 65–70.

Horst AM, Hill AP, Gorman KB (2020). ***palmerpenguins**: Palmer Archipelago (Antarctica) Penguin Data.* R package version 0.1.0, URL https://CRAN.R-project.org/package=palmerpenguins.

Hugh-Jones D (2022). ***huxtable**: Easily Create and Style Tables for LaTeX, HTML and Other Formats.* R package version 5.5.0, URL https://CRAN.R-project.org/package=huxtable.

Iannone R, Cheng J, Schloerke B (2022). ***gt**: Easily Create Presentation-Ready Display Tables.* R package version 0.6.0, URL https://CRAN.R-project.org/package=gt.

Lang M (2017). "**checkmate**: Fast Argument Checks for Defensive R Programming." *The R Journal*, **9**(1), 437–445. doi:10.32614/rj-2017-028.

Leifeld P (2013). "**texreg**: Conversion of Statistical Model Output in R to LaTeX and HTML Tables." *Journal of Statistical Software*, **55**(8), 1–24. doi:10.18637/jss.v055.i08.

Lüdecke D, Ben-Shachar MS, Patil I, Makowski D (2020). "Extracting, Computing and Exploring the Parameters of Statistical Models Using R." *Journal of Open Source Software*, **5**(53), 2445. doi:10.21105/joss.02445.

Lüdecke D, Ben-Shachar MS, Patil I, Waggoner P, Makowski D (2021). "**performance**: An R Package for Assessment, Comparison and Testing of Statistical Models." *Journal of Open Source Software*, **6**(60), 3139. doi:10.21105/joss.03139.

Lüdecke D, Waggoner PD, Makowski D (2019). "**insight**: A Unified Interface to Access Information from Model Objects in R." *Journal of Open Source Software*, **4**(38), 1412. doi:10.21105/joss.01412.

Miller S (2022). ***dragracer**: Data Sets for RuPaul's Drag Race.* R package version 0.1.7, URL https://CRAN.R-project.org/package=dragracer.

Murdoch D (2020). ***tables**: Formula-Driven Table Generation.* R package version 0.9.6, URL https://CRAN.R-project.org/package=tables.

R Core Team (2022). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rich B (2021). ***table1**: Tables of Descriptive Statistics in HTML.* R package version 1.4.2, URL https://CRAN.R-project.org/package=table1.

Robinson D, Hayes A, Couch S (2022). ***broom**: Convert Statistical Objects into Tidy Tibbles.* R package version 0.8.0, URL https://CRAN.R-project.org/package=broom.

Sjoberg DD, Curry M, Hannum M, Larmarange J, Whiting K, Zabor EC (2022). ***gtsummary**: Presentation-Ready Data Summary and Analytic Result Tables.* R package version 1.5.2, URL https://CRAN.R-project.org/package=gtsummary.

Waring E, Quinn M, McNamara A, Arino de la Rubia E, Zhu H, Ellis S (2022). ***skimr**: Compact and Flexible Summaries of Data.* R package version 2.1.4, URL https://CRAN.R-project.org/package=skimr.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis.* 3rd edition. Springer-Verlag.

Wickham H, Kuhn M, Vaughan D (2022). ***generics**: Common S3 Generics Not Provided by Base **R** Methods Related to Model Fitting.* R package version 0.1.2, URL https://CRAN.R-project.org/package=generics.

Yoshida K, Bartel A (2022). ***tableone**: Create "Table 1" to Describe Baseline Characteristics with or without Propensity Score Weights.* R package version 0.13.2, URL https://CRAN.R-project.org/package=tableone.

Zeileis A, Hothorn T (2002). "Diagnostic Checking in Regression Relationships." *R News*, **2**(3), 7–10. URL https://CRAN.R-project.org/doc/Rnews/.

Zeileis A, Köll S, Graham N (2020). "Various Versatile Variances: An Object-Oriented Implementation of Clustered Covariances in R." *Journal of Statistical Software*, **95**(1), 1–36. doi:10.18637/jss.v095.i01.

Zhu H (2021). ***kableExtra**: Construct Complex Table with **kable** and Pipe Syntax.* R package version 1.3.4, URL https://CRAN.R-project.org/package=kableExtra.

Affiliation:

Vincent Arel-Bundock
Université de Montréal
Science Politique, Pavillon Lionel-Groulx
3150 rue Jean-Brillant, C-4020
Montréal, QC
Canada, H3T 1N8
Email: vincent.arel-bundock@umontreal.ca

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/

July 2022, Volume 103, Issue 1 *Submitted:* 2021-08-11
doi:10.18637/jss.v103.i01 *Accepted:* 2021-11-25