Replicable Publication-Ready Model Output with R package apsrtable
Implementation & Extension Guide

Michael Malecki*

Saturday 20th March, 2010

1 Overview

Formatting fitted model objects is not trivial and should not be done at the last minute before submitting a paper. It is too easy for modeling details to get lost, or simply for values to be omitted, transposed, or otherwise misplaced. Thus automation is a boon, if not a requirement, for replicability.

Fitted model objects in R contain a lot of information, but often we want to look at several of them side-by-side. It would be nice to name the models, and include some model-fit information, and to name the covariates. Nested and non-nested models and naming become a problem for cbind and rbind, and last but not least, we want this information in some legible format – either plain text or, prefereably, Latex. My R package *apsrtable*, installable from CRAN, seeks to solve all of these problems, creating tables ready to be published in the *APSR*. I might have named it after *Political Analysis*, but the latter often features other innovative presentations of data and results, such that "a *PA*-style table" is less identifiable.

Like many graduate students – including probably most who read TPM – I have typeset *a lot* of fitted models in the course of methods training, leaving aside the relatively few tables that make their way into final seminar, conference, or journal papers. We were "strongly encouraged" to prepare homework assignments using Latex and create "professional-looking" tables of model output, with the expectation that we would get faster at it, while becoming attuned to the intricacies of Latex's tabular and matrix environments. I found this exercise as tedious as formatting a bibliography by hand. So, I set out to automate R's output of "professional-looking" tables. To complete the bibliography analogy, I also wanted to make the automated solution more user-friendly than bibtex BSTs – though "an order of magnitude easier than BST" is still an extremely low target.

*PhD Candidate, Washington University in St. Louis

2 Usage

To demonstrate `apsrtable`'s features, I'll replicate some of the results from Persson and Tabellini's (2003) *The Economic Effects of Constitutions*[1], in particular models 1–3 of Table 6.1 (p. 159) (Figure 1).

Figure 1. Persson and Tabellini (2003), p.159, detail.

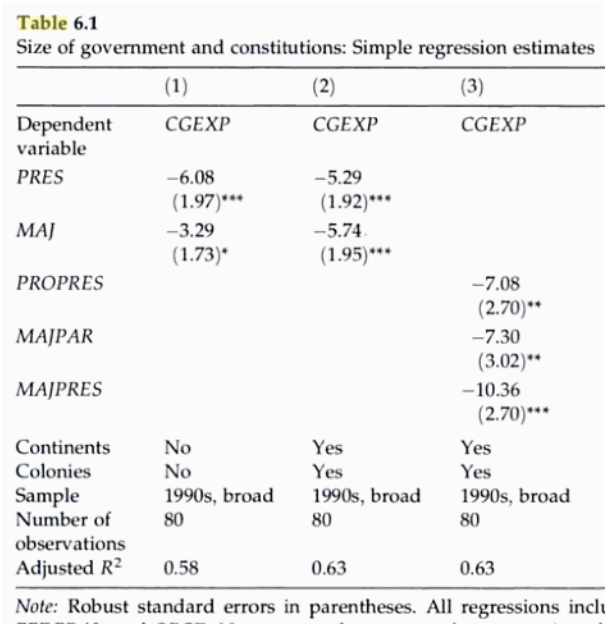

Table 6.1
Size of government and constitutions: Simple regression estimates

	(1)	(2)	(3)
Dependent variable	*CGEXP*	*CGEXP*	*CGEXP*
PRES	−6.08 (1.97)***	−5.29 (1.92)***	
MAJ	−3.29 (1.73)*	−5.74 (1.95)***	
PROPRES			−7.08 (2.70)**
MAJPAR			−7.30 (3.02)**
MAJPRES			−10.36 (2.70)***
Continents	No	Yes	Yes
Colonies	No	Yes	Yes
Sample	1990s, broad	1990s, broad	1990s, broad
Number of observations	80	80	80
Adjusted R^2	0.58	0.63	0.63

Note: Robust standard errors in parentheses. All regressions inclu

```
> pt <- read.csv("pt.csv")
> library(apsrtable)
> controls <- "lyp + gastil + age + trade + prop65 + prop1564 + federal + oecd"
> regions <- "col_uka + col_espa + col_otha + africa + asiae + laam"
> model1 <- lm(as.formula(paste("cgexp ~", paste(c("pres + maj",
                                                   controls),collapse="+"))), data=pt)
> model2 <- lm(as.formula(paste("cgexp ~",paste(c("pres + maj",
                                                  controls,regions),collapse="+"))), data=pt)
> model3 <- lm(as.formula(paste("cgexp ~",paste(c("pro.pres + maj.parl + maj.pres",
                                                  controls,regions),collapse="+"))), data=pt)
```

The result of simply calling `apsrtable(model1, model2, model3)` shouldn't be very far from usable output. We get decimal-aligned columns that are also lined up in the Latex source, errors in parentheses, and a single star indicating $p < .05$. But, we want to include "robust" standard errors, omit a bunch of

[1]The data are available from the authors, though they may contain some coding errors in the institutional interaction variables. A corrected replication dataset is available here.

controls, give the variables less cryptic names, and, to complete the exercise, add more stars. All of these are extremely easy to do with *apsrtable*, but looking at the default output is a helpful baseline and a recommended first step every time.

Alternate Standard Errors However you want to go about estimating alternative standard errors, simply insert a vector or the full new variance-covariance matrix into the fitted model object and name it `se`[2] To get the same standard errors that Persson and Tabellini report (also, I believe, the Stata default), use the *sandwich* package:
`model1$se <- vcovHC(model1,type="HC1")` and so on for each model. By default, apsrtable will include only the robust standard errors when present, but the argument `se` can either ignore them and print the model's original standard errors ("vcov") or both ("both").

Order of Covariates Including nested and non-nested models side-by-side, lining up the covariates that are included and leaving blank those that are not, is the main feature of the package. For this replication (and most of the time) the default "lr" left-to-right incorporation of terms is correct. For special cases, two other options, "rl" and "longest" are provided and discussed briefly under implementation details. For typical use, it is best to nail down the desired order in which the models are passed to the function, and then the value of `order`.

Omitting controls Variables can be omitted from the display either by name or index. It is easiest in this case to supply the argument `omit=c(1,4:17)`. However, the order of coefficients can change, so a list of quoted character names of coefficients to omit may be safer, and certainly makes your code more transparent and thus easier to maintain and replicate. As of version 0.8, you can also provide an `expression`, such as a regular expression. In this case, the argument should take the form:
`expression(grep(pattern,coefnames))`
The internal object `coefnames` is the ordered union of all terms in all models.

To include multiple regular expressions, or a mix of expressions with other types, you may supply a `list`, but you must ensure that the result is a valid subscript list: all character, all numeric, or all logical. For example, if you refer to a specific coefficient by its character name, include the argument `value=TRUE` in any `grep` expressions in the list.

Names of Models and Covariates Models by default are named "Model 1", "Model 2", etc., but numbering can be changed arbitrarily (via argument `model.counter`), or a vector of meaningful names can be supplied as `model.names` (if it comes up short, numbering accounts for the named ones).

Covariate name replacement takes place after the list of included coefficients from all the models has been generated. Therefore, again it is a good idea to look at the default, settle on the values of `order` and `omitcoef`, and only then supply a vector of coefficient display names as the argument `coef.names`.

[2]Most model objects are simply lists rather than formal classes (that is, they use the S3 class system). Models fit with `lmer` are formally *mer* class objects but changes in how *lme4* methods `display` and `summary` work has prevented incorporation of that class of models so far.

Stars The default behavior is to indicate coefficient significance at the level $p < .05$ with a single superscript asterisk. Two arguments allow you to increase the sparkle of almost any table. To set a different level for a single star, supply it to `lev`. To include a dagger for .10, a star for .05, two stars for .01, and three stars (!) for .001, supply the argument `stars="default"`. I admit the name is confusing, but "default" here indicates the R default rather than the APSR and *apsrtable* default.

Other arguments The other arguments are documented in the package, but two are worth mention here. The handling of notes about standard errors and significance indicators is discussed below. Also, `Sweave=TRUE` should be used whenever you include a call to `apsrtable` from within a Sweave document. Otherwise output will include the table environment code and empty caption and label, which should be included in the Sweave document itself if emacs reftex is to keep track of tables. An internal test for Sweave might appear in future versions, but the default output is meant for cutting and pasting from an R session into a Latex document, which seems to be fairly common workflow.

3 Some Implementation Details

Three features of *apsrtable* bear mention for how I implemented them: covariate aggregation across models, table notes, and model summaries, which provides a flexible extension framework.

3.1 Covariate Order

Authors have immense control over the order in which covariate rows are presented in the final table. Through a combination of the order of terms in the model objects, and the choice of `order` in the call to `apsrtable`, one should be able to create tables that never need "rearrangement" of rows between output and publication. First, it builds a list of all variable names, and then notes the position of the variables in each model with respect to this final order. Next, the `omitcoef` list is marked, and finally, the names of remaining terms are replaced with the optional list of `coef.names`, which, of course, may contain Latex markup. The column of variable names is set in text mode, so any math should be delimited by $, and because it's R, backslashes have to be doubled. A label "β_0 Intercept" would be supplied as `"$\\beta_0$ Intercept"`.

The default left-to-right order starts with the first model; any terms included in the second not in the first are appended to the order, and so on. Right-to-left (`order="rl"`) takes the initial order from the rightmost model; and `order="longest"` starts with the order of terms in the model with the most terms, wherever it is in the list of models, then appends any others always left-to-right.

3.2 Table Notes

Some kinds of notes pertaining to the table depend on values used to generate it. In particular, informing readers of "robust" standard errors, and the indicators and "level" of statistical significance lies in the gap

between content and presentation. Authors also commonly indicate the source of data in a note beneath a table (technically a multicolumn span within the tabular environment).

The `notes` argument allows you to specify a list of functions or character strings. R's "lazy evaluation" means that these functions are evaluated only when `eval` is called explicitly inside `apsrtable`. R's variable scoping meant that the variables in the `apsrtable()` call reside 3 levels up the call stack, so custom functions in the `notes` list can only access them by specifying the environment at `sys.frame(-3)`. Of course, simple character strings are valid in the `notes` list and will be typset along with the results of the dynamic functions.

3.3 Extending apsrtable with `modelInfo`

Besides the coefficients, standard errors, and statistical significance star(s), model summaries and some indication of goodness-of-fit should be included with each model, which I call *modelInfo*. But, different researchers prefer to include different statistics describing models as a whole. Therefore, *apsrtable* comes with some reasonable defaults and a simple mechanism to change them. The same mechanism makes extending the package to other types of models relatively easy.

ModelInfo methods are called on the list of model summaries (that is, the result of `summary(model)`). At a minimum, one can select from the information contained therein to generate a list of named character strings that comprise the modelInfo that will be included in the final output. The formatter doesn't care what you include here – it simply matches the names (always left-to-right) across models and prints it all. If the `summary` object does not contain some data that you need for a model summary statistic, see Section 3.4 for creating special `apsrtableSummary` methods that return the information you want.

So, what does a modelInfo method look like? It is a formal S4 method that takes a model summary object as its argument. Therefore to change the list of model information, just change what the modelInfo function returns for a given model class. Future versions will probably include a selection of presets, but changing by a quick call to `setMethod` is straightforward and my aim here is to make clear how to get the modelInfo you want. For `lm` objects, the default prints the N, R^2, adjusted R^2, and residual sd. The example in `?modelInfo` shows you how to print only the N and residual sd; the easiest way to find the default is via `getMethod("modelInfo","summary.lm")`. The defaults for `lm` and `glm` are also included as named private functions, to make reversion easier (see the modelInfo example).

To return to the Persson and Tabellini example, they summarize the controls used (but not displayed) in the modelInfo section. It is simple to write a function for `summary.lm` that tests for the presence of a particular name among coefficients. Besides this, they report the N as "Number of Observations" (so we want to change the name of that element in the list), and the adjusted R^2. So, we just create a custom `modelInfo` method that returns these items.

```
> ## Add robust se to the models
> library(sandwich)
> model1$se <- vcovHC(model1,type="HC1")
> model2$se <- vcovHC(model2,type="HC1")
```

```
> model3$se <- vcovHC(model3,type="HC1")
> ## Create and register custom modelInfo for lm
>
> setMethod("modelInfo", "summary.lm", function(x) {
   env <- parent.frame()
   digits <- evalq(digits,env)
   model.info <- list(
     "Continents"= (
         ifelse(!is.na(charmatch("laam",rownames(coef(x)))),
           "\\multicolumn{1}{c}{Yes}", "\\multicolumn{1}{c}{No}")),
     "Colonies" = (
         ifelse(!is.na(charmatch("col",rownames(coef(x)))),
           "\\multicolumn{1}{c}{Yes}", "\\multicolumn{1}{c}{No}")),
     "Number of\\tabularnewline Observations"= (formatC(sum(x$df[1:2]),format="d")),
     "adj. $R^2$" = (formatC(x$adj.r.squared,format="f",digits=digits)))
 class(model.info) <- "model.info"
 return(model.info)
 } )
```

With the new modelInfo method registered, the final call to `apsrtable()` is shown below, and the results typeset as Table 1. Note that the `omitcoef` list contains a character element and then two expressions that evaluate to character vectors.

```
> apsrtable(model1,model2,model3,
 omitcoef=list("(Intercept)", expression(strsplit(controls," \\+ " )), expression(strsplit(regions," \\+
 coef.names=c("Presidential","Majoritatian",
   "Proportional Presidential", "Majoritarian Parliamentary", "Majoritarian Presidential"),
 align="left",stars="default",
 notes=list(se.note(), stars.note())
 )
```

The example above shows how the `modelInfo` methods can be used to change the display for linear model summary objects. In theory, any model object for which `coef`, `vcov`, and `summary` methods should work with *apsrtable*. Extending means mainly creating `modelInfo` methods for other classes of model summaries.

3.4 Adding model classes with `apsrtableSummary`

Sometimes, the `summary` method for a given object either does not exist, or does not produce output that *apsrtable* knows what to do with. In this case, users have two options: write a replacement summary method and submitting it to package maintainers, or create a summary method that only *apsrtable* will use. Presumably, the latter may be a stopgap on the way to the former.

Table 1. Replication of Models 1–3 from Persson and Tabellini (2003), p. 149: "Size of government and constitutions: Simple regression estimates."

	Model 1	Model 2	Model 3
Presidential	-6.08^{**}	-5.29^{**}	
	(1.97)	(1.92)	
Majoritatian	$-3.29^{\dagger}$	-5.74^{**}	
	(1.73)	(1.95)	
Proportional Presidential			-7.08^{*}
			(2.70)
Majoritarian Parliamentary			-7.30^{*}
			(3.02)
Majoritarian Presidential			-10.36^{***}
			(2.70)
Continents	No	Yes	Yes
Colonies	No	Yes	Yes
Number of Observations	80	80	80
adj. R^2	0.58	0.63	0.63

Robust standard errors in parentheses

† significant at $p < .10$; $^{*}p < .05$; $^{**}p < .01$; $^{***}p < .001$

One user requested support for the *gee* package, specifically `gee.robust` objects. A summary of them suitable for *apsrtable* would have to place the robust standard errors into the `$se` position of the summary object, as well as turn the `summary.gee`'s columns of z-scores into Pr(z) for the bedazzler `apsrstars`.[3] The solution is to write a replacement `apsrtableSummary` which is always tried first, with the model-object's default `summary` called otherwise.

The example manipulates the output of `summary.gee` into a matrix (and `$se` vector) suitable for seamless use with *apsrtable.*

```
> "apsrtableSummary.gee" <- function(x) {
   s <- summary(x)
   newCoef <- coef(s)
   ## which columns have z scores? (two of them in robust case)
   zcols <- grep("z",colnames(newCoef))
   newCoef[,zcols] <- pnorm(abs(newCoef[,zcols]), lower.tail=FALSE)
   colnames(newCoef)[zcols] <- "Pr(z)"
   s$coefficients <- newCoef
   ## put the robust se in $se so that notefunction works automatically
   ## the se checker will overwrite [,4] with pt, but this doesn't matter
   ## because the last column Pr(z) is used by apsrstars() anyway
```

[3]Relatedly, an `apsrtableSummary` may be needed just to *rename* columns in some summary coefficient matricies. The function `apsrstars`, derived from `stats:::printCoefmat`, checks for column names starting with Pr(.

```
  ## and the se are pulled from $se.
  if( class(x)[1] == "gee.robust") {
    s$se <- coef(s)[,4]
  }
  return(s)
}
```

4 Conclusion

I saw a striking need to produce publication-ready output as a part of the modeling process, especially as complexity grew. Not surprisingly, I learned after publishing the package on CRAN and announcing it to PolMeth, that I was not alone. Martin Elff has a similar function (`mtable`) in his *memisc* package, which can also produce non-Latex output. Some of the details of his implementation caused me to refactor parts of my own. My current workflow relies heavily on Sweave to ensure that the code I run is actually connected to both the theoretical arguments I make and the results I present. I rely on *apsrtable* to produce reliable, replicable, relatively painless tables of results. I hope that *TPM* readers can save themselves (and their graduate students!) some typesetting frustration by using *apsrtable*, and I look forward to extensions and improvements that users may provide.