texreg: Conversion of Statistical Model Output in R to \LaTeX{} and HTML Tables

Philip Leifeld
University of Essex

Abstract

A recurrent task in applied statistics is the (mostly manual) preparation of model output for inclusion in \LaTeX{}, Microsoft Word, or HTML documents – usually with more than one model presented in a single table along with several goodness-of-fit statistics. However, statistical models in R have diverse object structures and summary methods, which makes this process cumbersome. This article first develops a set of guidelines for converting statistical model output to \LaTeX{} and HTML tables, then assesses to what extent existing packages meet these requirements, and finally presents the texreg package as a solution that meets all of the criteria set out in the beginning. After providing various usage examples, a blueprint for writing custom model extensions is proposed.

Keywords: reporting, table, coefficients, regression, R, \LaTeX{}, MS Word, HTML, Markdown.

This R package vignette is based on an article in the Journal of Statistical Software (Leifeld 2013). The contents of the article have not been modified, apart from the author affiliation. However, the package has been updated with more arguments, model extensions, and functions. The help pages contain details on any differences to the version documented here.

1. Typesetting R model output in \LaTeX{} and HTML

The primary purpose of the statistical programming language R (R Core Team 2013) is the analysis of data with statistical models. One of the strengths of R is that users can implement their own statistical models. While this flexibility leads to an increased availability of even exotic models and shorter cycles between model development and implementation, there are also downsides of this flexibility. In particular, there is no unified data structure of statistical models and no unified way of representing model output, which makes it hard to re-use coefficients and goodness-of-fit (GOF) statistics in other software packages, especially for the purpose of publishing results.

Several generic functions were developed to provide unified accessor methods for coefficients (the \texttt{coef()} function), GOF statistics (for example, \texttt{AIC()}, \texttt{BIC()}, \texttt{logLik()}, or \texttt{deviance()}), a custom text representation of the fitted model (\texttt{summary()}), and other relevant pieces of information (e.g., \texttt{nobs()} and \texttt{formula()}). Details are provided in Chapter 11 of Venables, Smith, and R Core Team (2013). Nonetheless, many popular packages have only partially implemented methods for these generics, and in some cases they do not even provide accessor functions at all for their coefficients or GOF statistics. Even worse, the model summary methods are usually structured in idiosyncratic ways and do not lend themselves to easy
The design of the texreg package tries to accomplish six goals: it should be capable of dealing with several models in a single table; it should be easily extensible by package writers and users; it should provide options for using the available space in an optimal way; it should
take advantage of advanced layout capabilities in \LaTeX{} and HTML; it should take care of
different model classes and standard errors. There are, however, also cases where
different model types are applied to the same data. This implies that the package must
not only be able to merge the names of coefficients to guarantee comparability of coefficient
columns; it must also be able to deal with different model classes and accommodate different
kinds of GOF statistics in the same table.

Moreover, it must be possible to rename the model terms and GOF statistics. Custom coef-
ficient names not only make the output more easily comprehensible by the readers; renaming
model terms is also mandatory for unifying terms between several models. For example, two
models based upon two different datasets may have different variable names for the same kind
of theoretical construct. This would result in two separate but complementary rows in the
coefficient table. It should be possible to rename coefficients and then conflate any two or
more complementary rows with identical labels.

Finally, it should be possible to assign custom names for the different models, instead of
the default “Model 1”, “Model 2”, etc. While it may be easy to rename them manually in
many applications, particularly \texttt{Sweave()} and \texttt{knitr()} require that no manual manipulation
is necessary.

2.2. Using generics to make \texttt{texreg} easily extensible

Different model classes have different ways how their coefficients and GOF statistics can be
accessed. Hardcoding these extraction rules into the functions of the \texttt{texreg} package would
inhibit customizability. The best way to make \texttt{texreg} extensible is to have a generic \texttt{extract()} function which can be invoked on any kind of model, just like \texttt{plot()} or \texttt{print()} generics in R. Any user—especially model class authors—would then be able to add custom methods
to the \texttt{extract()} function in order to make \texttt{texreg} learn how to cope with new models. For
example, an \texttt{extract()} method for ‘\texttt{lm}’ objects can be written to deal with linear models,
or an \texttt{extract()} method for ‘\texttt{ergm}’ objects can be written to make the generic \texttt{extract()} function understand exponential random graph models. All the user has to do is write a
custom extract function for a specific model type and then register it as a method for the
generic \texttt{extract()} function. Section 6 provides details on how this can be accomplished.

2.3. Use available space efficiently

If a table has many model terms and if standard errors are printed in parentheses below
the coefficients, the table may become too large for a single page. For this reason, it should
be possible to print standard errors right beside the coefficients instead of aligning them
vertically. In \texttt{texreg}, this is achieved with the \texttt{single.row} argument.

If tables grow too large, other measures might prove useful: removing table margins, setting
the table in script size, or setting custom float positions (for \LaTeX{} tables). Very wide tables
should be rotated by 90 degrees using the \texttt{sidewaystable} environment in the \LaTeX{} package.
rotating (Rahtz and Fairbairns 2008) in order to use the available space in an optimal way. The user should also be able to set the table caption and label, decide whether the table should be in a float environment (for \LaTeX tables), align the table horizontally on the page, and set the position of the caption. The \texttt{texreg} package provides arguments to control all of these aspects.

2.4. Beautiful and clean table layout

For \LaTeX tables, the \LaTeX package \texttt{dcolumn} (Carlisle 2001) provides facilities for aligning numbers in table cells nicely, for example at their decimal separators. The \texttt{booktabs} package (Fear 2005) can draw top, mid and bottom rules for tables and produces a cleaner layout than the default horizontal lines. Both packages are supported by \texttt{texreg} and can be switched on or off depending on the availability of the packages in the \LaTeX distribution.

For HTML tables, cascading style sheets (CSS) should be used to adjust the layout, and the user should be able to decide whether CSS markup should be included in the file header or inline.

2.5. Journal- or model-specific requirements

Academic journals may have different requirements regarding the number of digits to be printed, the inclusion of superscripted stars indicating significance, or the removal of leading zeroes. Similarly, there are best practices in different academic communities or for specific model types. For example, it is common practice to attach three stars to coefficients with \( p \) values \( \leq 0.001 \) and small centered dots to coefficients with \( p \) values between 0.05 and 0.1 in exponential random graph models, while less fine-grained significance levels are adopted in many other communities (for example, three stars for \( p \leq 0.01 \), or only one star or bold formatting for one single significance level). In yet other communities, journals or models, \( p \) values or significance stars are not required or even deemed inappropriate (see the \texttt{lme4} package by Bates, Maechler, Bolker, and Walker 2013).

2.6. Customizability and usability

Different users have different applications in mind. For this reason, a solution should be as flexible as possible and offer customization via arguments. For example, inclusion of an HTML table in a \texttt{knitr} Markdown document requires that only the table is printed without any header or document type information, and that significance stars are escaped using backslashes.

In other situations, it may be important to (1) omit certain coefficients (like random or fixed effects or thresholds), (2) reorder the coefficients in the model (e.g., because some models put interaction effects at the end of the list of coefficients), or (3) replace coefficients, standard errors, or \( p \) values by custom vectors, for example when heteroskedasticity-consistent (“robust”) standard errors are used (Zeileis 2004).
<table>
<thead>
<tr>
<th>Argument</th>
<th>\texttt{gtex}</th>
<th>\texttt{gmlt}</th>
<th>\texttt{screenreg}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>file</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>single.row</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>stars</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>custom.model.names</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>custom.coef.names</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>custom.gof.names</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>custom.note</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>digits</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>leading.zero</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>symbol</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>override.coef</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>override.se</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>override.pval</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>omit.coef</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>reorder.coef</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>reorder.gof</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>return.string</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>ci.force</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>ci.level</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>ci.star</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>bold</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>p value below which coefficients are bolded</td>
</tr>
<tr>
<td>center</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>Horizontal alignment on the page</td>
</tr>
<tr>
<td>caption</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>Set the caption of the table</td>
</tr>
<tr>
<td>caption.above</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>Should the caption be placed above the table?</td>
</tr>
<tr>
<td>label</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>Set the label of the table</td>
</tr>
<tr>
<td>booktabs</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>dcolumn</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>sideways</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>use.packages</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>table</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>no.margin</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>scriptsize</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>float.pos</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>star.symbol</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>inline.css</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>doctype</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>html.tag</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>head.tag</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
<tr>
<td>body.tag</td>
<td>\checkmark</td>
<td>\checkmark</td>
<td>\checkmark</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Argument</th>
<th>texreg</th>
<th>htmlreg</th>
<th>screenreg</th>
<th>Short description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column.spacing</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>Number of spaces between columns</td>
</tr>
<tr>
<td>outer.rule</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>Line type for the outer rule</td>
</tr>
<tr>
<td>inner.rule</td>
<td>o</td>
<td>o</td>
<td>•</td>
<td>Line type for the inner rule</td>
</tr>
<tr>
<td>...</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>Additional arguments for the extract functions</td>
</tr>
</tbody>
</table>

Table 1: Arguments of the texreg(), htmlreg() and screenreg() functions.

On the other hand, users should not be required to learn the meaning of all arguments before they can typeset their first table. The default arguments should serve the needs of occasional users. Moreover, adjusting tables based on a complex set of arguments should be facilitated by printing tables to the R console before actually generating the \LaTeX or HTML output. If this screen representation of the table is nicely formatted and aligned using spaces and rules, it can also serve as an occasional replacement for the generic `summary()` method for easy model comparison as part of the statistical modeling workflow.

The texreg package tries to balance these needs for customizability and usability by providing many arguments for layout customization (see Table 1 for a list of arguments), using sensible default values for occasional users, and providing a function for on-screen display of tables for easy model comparison and layout adjustment.

3. Comparison with other packages

Beside texreg, several other packages were designed to convert R model output to \LaTeX or HTML tables.

The xtable package (Dahl 2012) is able to typeset various matrices and data frames from R as \LaTeX or HTML tables. It is very flexible and has its strengths particularly when it comes to tables of summary statistics. However, it was not specifically designed for statistical model output. Similarly, the mat2tex() command from the sfsmisc package (Maechler and others 2012) can export matrices to \LaTeX, and the tex.table() function in the cwhmisc package (Hoffmann 2012) is able to export data frames as \LaTeX tables.

For several years, the outreg() function in the rockchalk package (Johnson 2012) has been available for exporting multiple regression models to \LaTeX. However, the function remains fairly basic and does not provide a great deal of layout options, generics, and custom model types (in fact, only ‘lm’ and ‘glm’ objects).

The apsrtable package (Malecki 2012), the mtable() function from the memisc package (Elff 2012), and the stargazer package (Hlavac 2013) are more advanced and can also merge multiple models in a single table. apsrtable and memisc feature custom functions for the extraction of coefficient and GOF information, and they are based on generics. In this regard, both packages are somewhat similar to the texreg package. texreg, however, offers more straightforward ways of custom model implementation. This important feature is notably absent from stargazer.
<table>
<thead>
<tr>
<th>Class</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>’aftreg’</td>
<td>eha</td>
<td>Accelerated failure time regression</td>
</tr>
<tr>
<td>’betareg’</td>
<td>betareg</td>
<td>Beta regression for rates and proportions</td>
</tr>
<tr>
<td>’brglm’</td>
<td>brglm</td>
<td>Bias-reduced generalized linear models</td>
</tr>
<tr>
<td>’btergm’</td>
<td>btergm</td>
<td>Temporal exponential random graph models</td>
</tr>
<tr>
<td>’clm’</td>
<td>ordinal</td>
<td>Cumulative link models</td>
</tr>
<tr>
<td>’cllogit’</td>
<td>survival</td>
<td>Conditional logistic regression</td>
</tr>
<tr>
<td>’coeftest’</td>
<td>lmtest</td>
<td>Wald tests of estimated coefficients</td>
</tr>
<tr>
<td>’coxph’</td>
<td>survival</td>
<td>Cox proportional hazard models</td>
</tr>
<tr>
<td>’coxph.penal’</td>
<td>survival</td>
<td>Cox proportional hazard with penalty splines</td>
</tr>
<tr>
<td>’dynlm’</td>
<td>dynlm</td>
<td>Time series regression</td>
</tr>
<tr>
<td>’ergm’</td>
<td>ergm</td>
<td>Exponential random graph models</td>
</tr>
<tr>
<td>’gam’</td>
<td>mgcv</td>
<td>Generalized additive models</td>
</tr>
<tr>
<td>’gee’</td>
<td>gee</td>
<td>Generalized estimation equation</td>
</tr>
<tr>
<td>’glm’</td>
<td>stats</td>
<td>Generalized linear models</td>
</tr>
<tr>
<td>’[gl</td>
<td>l</td>
<td>nl]merMod’</td>
</tr>
<tr>
<td>’gls’</td>
<td>nlme</td>
<td>Generalized least squares</td>
</tr>
<tr>
<td>’gmm’</td>
<td>gmm</td>
<td>Generalized method of moments estimation</td>
</tr>
<tr>
<td>’ivreg’</td>
<td>AER</td>
<td>Instrumental-variable regression using 2SLS</td>
</tr>
<tr>
<td>’hurdle’</td>
<td>pscl</td>
<td>Hurdle regression models for count data</td>
</tr>
<tr>
<td>’lm’</td>
<td>stats</td>
<td>Ordinary least squares</td>
</tr>
<tr>
<td>’lme’</td>
<td>nlme</td>
<td>Linear mixed-effects models</td>
</tr>
<tr>
<td>’lme4’</td>
<td>lme4 (≥ 1.0)</td>
<td>Linear mixed-effects models</td>
</tr>
<tr>
<td>’lmrob’</td>
<td>robustbase</td>
<td>MM-type estimators for linear models</td>
</tr>
<tr>
<td>’lnam’</td>
<td>sna</td>
<td>Linear network autocorrelation models</td>
</tr>
<tr>
<td>’lrm’</td>
<td>rms, Design</td>
<td>Logistic regression models</td>
</tr>
<tr>
<td>’maBina’</td>
<td>erer</td>
<td>Marginal effects for binary response models</td>
</tr>
<tr>
<td>’multinom’</td>
<td>nnet</td>
<td>Multinomial log-linear models</td>
</tr>
<tr>
<td>’negbin’</td>
<td>MASS</td>
<td>Negative binomial generalized linear models</td>
</tr>
<tr>
<td>’nlme’</td>
<td>nlme</td>
<td>Nonlinear mixed-effects models</td>
</tr>
<tr>
<td>’phreg’</td>
<td>eha</td>
<td>Parametric proportional hazards regression</td>
</tr>
<tr>
<td>’plm’</td>
<td>plm</td>
<td>Linear models for panel data</td>
</tr>
<tr>
<td>’pmg’</td>
<td>plm</td>
<td>Linear panel models with heterogeneous coefficients</td>
</tr>
<tr>
<td>’polr’</td>
<td>MASS</td>
<td>Ordered logistic or probit regression</td>
</tr>
<tr>
<td>’Relogit’</td>
<td>Zelig</td>
<td>Rare events logistic regression</td>
</tr>
<tr>
<td>’rem.dyad’</td>
<td>relevent</td>
<td>Relational event models for dyadic data</td>
</tr>
<tr>
<td>’rlm’</td>
<td>MASS</td>
<td>Robust fitting of linear models</td>
</tr>
<tr>
<td>’rq’</td>
<td>quantreg</td>
<td>Quantile regression models</td>
</tr>
<tr>
<td>’sclm’</td>
<td>ordinal</td>
<td>Cumulative link models</td>
</tr>
<tr>
<td>’sienaFit’</td>
<td>RSiena</td>
<td>Stochastic actor-oriented models for networks</td>
</tr>
<tr>
<td>’simex’</td>
<td>simex</td>
<td>SIMEX algorithm for measurement error models</td>
</tr>
<tr>
<td>’stergm’</td>
<td>tergm</td>
<td>Temporal exponential random graph models</td>
</tr>
<tr>
<td>’survreg’</td>
<td>survival</td>
<td>Parametric survival regression models</td>
</tr>
<tr>
<td>’survreg.penal’</td>
<td>survival</td>
<td>Fraility survival models</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Class</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘svyglm’</td>
<td>survey</td>
<td>Survey-weighted generalized linear models</td>
</tr>
<tr>
<td>‘systemfit’</td>
<td>systemfit</td>
<td>Linear structural equations</td>
</tr>
<tr>
<td>‘texreg’</td>
<td>texreg</td>
<td>For easy manipulation of texreg tables</td>
</tr>
<tr>
<td>‘tobit’</td>
<td>AER</td>
<td>Tobit regression models for censored data</td>
</tr>
<tr>
<td>‘weibreg’</td>
<td>cha</td>
<td>Weibull regression</td>
</tr>
<tr>
<td>‘zelig’</td>
<td>Zelig</td>
<td>Zelig models (Owen, Imai, King, and Lau 2013)</td>
</tr>
<tr>
<td>‘zeroinfl’</td>
<td>pscl</td>
<td>Zero-inflated regression models</td>
</tr>
</tbody>
</table>

Table 2: List of 52 supported model types (version 1.30).


The `texreg` package (version 1.30), in contrast, can deal with all of the above model types (that is, the union of all three packages, except for some `Zelig` models), is extensible, and offers additional built-in functions for the following model classes: ‘brglm’, ‘btergm’, ‘coxph.penal’, ‘gmm’, ‘lme’, ‘lme4’, ‘lmmam’, ‘maBina’, ‘nlme’, ‘rem.dyad’, ‘rq’, ‘sclm’, ‘stergm’, ‘systemfit’, ‘texreg’ and ‘zelig’ (‘logit’ and ‘relogit’) objects. Table 2 gives an overview of currently implemented model types.

The `texreg` package supports Microsoft Word, HTML, Markdown, and `knitr` whereas the other packages (except for `xtable`) are restricted to `\LaTeX` output. `apsrtable` has an option for Sweave integration, which does not require any argument in `texreg`. In the `memisc` package and in `texreg`, the `booktabs` and `dcolumn` `\LaTeX` packages for table layout (see Section 2.4) can be used, which is not available in `apsrtable` (and only `dcolumn` is supported in `stargazer`).

While `apsrtable` and `texreg` allow for custom GOF measures, `memisc` and `stargazer` only feature a set of hardcoded statistics. Apart from this, all packages presented here are significantly less flexible than `texreg` regarding the utilization of space (Section 2.3), layout options (Section 2.4), outlet- or model-specific requirements (Section 2.5), and customizability (Section 2.6).

4. Under the hood: How `texreg` works

The `texreg` package consists of three main functions:
1. *texreg* for \LaTeX\ output;
2. *htmlreg* for HTML, Markdown-compatible and Microsoft Word-compatible output;
3. *screenreg* for text output to the R console.

There are various internal helper functions, which are called from each of these main functions for purposes of pre- and postprocessing. Moreover, there is a class definition for ‘*texreg*’ objects, and a generic extract() function along with its methods for various statistical models. Figure 1 illustrates the procedure following a call of one of the main functions. Details about each step are provided below.

### 4.1. The generic extract() function and its methods

First, the user hands over a model object or a list of models to the *texreg*, *htmlreg* or *screenreg* function. This main function calls the generic extract() function in order to retrieve all the relevant pieces of information from the models. The extract() function knows how to cope with various model types (see Table 2) because it merely calls the appropriate extract() method designed for the specific model type. For example, if the model is of class ‘*lm*’, extract() calls the extract() method for ‘*lm*’ objects, etc. Custom extract() methods can be easily added (see Section 6).

An extract() method aggregates various required pieces of information, like the coefficients, their names, standard errors, \( p \) values, and several GOF measures. Which measures are used depends on the specific extract() method. It is also possible to let the user decide: beside the model argument, each extract method is allowed to have more arguments. For example, the extract() method for ‘*lme4*’ objects, which are *lme4* multilevel models (Bates *et al.* 2013), has Boolean options like include.variance, which turns on the inclusion of random effect variances in the GOF block, and numeric arguments like conf.level, which sets the confidence level for computation of profile or bootstrapped confidence intervals. When the main function is called in the first place, the user can include these custom arguments to finetune the behavior of the extract() methods.

Once the relevant data have been extracted from a model object, the extract() method creates a ‘*texreg*’ object by calling the createTexreg() function and handing over the extracted data. The ‘*texreg*’ object or the list of ‘*texreg*’ objects is finally returned to the main function.

### 4.2. ‘*texreg*’ objects: An S4 class

There is an S4 class definition for ‘*texreg*’ objects. Such an object contains four vectors for the coefficients—the coefficient values (numeric), their names (character), standard errors (numeric), and \( p \) values (numeric)—, and three vectors for the GOF statistics: the GOF
values (numeric), their names (character), and dummy variables indicating whether it makes sense for the GOF value to have several decimal places (logical); for example, one would not want the number of observations to have any decimal places.

As some types of statistical models report confidence intervals rather than standard errors and p values, the ‘texreg’ class definition can alternatively store lower and upper bounds of confidence intervals instead of standard errors and p values. Which slots of the class are used depends on the extract() method for the specific model. The texreg package checks whether standard errors are present in the ‘texreg’ object and use either standard errors or confidence intervals depending on availability.

The class contains validation rules which make sure that the four coefficient vectors all have the same length and that the three GOF vectors also all have the same length. There are several exceptions to this rule: the p values, the confidence intervals, and the decimal-place vector are optional and may also have a length of zero.

The ‘texreg’ class definition was written to facilitate the handling of the relevant pieces of information. Handing over lists of ‘texreg’ objects between functions is more user-friendly than handing over lists of nested lists of vectors. ‘texreg’ objects are created by the extract() methods and handed over to the texreg function (see Section 4.1).

4.3. Preprocessing the ‘texreg’ objects

Once all ‘texreg’ objects have been returned to the texreg(), htmlreg() or screenreg() function, they have to be preprocessed. This entails two steps: first, coefficients, standard errors or p values must be replaced by user-defined numeric vectors (for example if robust standard errors have been manually computed). The arguments override.coef, override.se, and override.pvalues serve to replace the coefficients, standard errors, and p values, respectively. Second, \LaTeX -specific markup codes are replaced by their HTML or plain-text equivalents if htmlreg() or screenreg() are called instead of texreg().

4.4. Matching the model terms

After preprocessing the ‘texreg’ objects, their contents are arranged in three separate matrices: the coefficient block matrix consists of three columns for each model (coefficient, standard error, and p value); the GOF block matrix consists of one column for each model and one row for each GOF statistic and contains the GOF values; and the decimal matrix has the same dimensions as the GOF block matrix and indicates for each GOF value whether it should have decimal places (e.g., $R^2$, AIC, etc.) or whether it is an integer (e.g., number of observations, number of groups, etc.). All of these matrices are created by matching the names of the coefficients or GOF names of the different models to avoid redundancy. The three matrices are kept separate during the postprocessing stage and are then combined in a single table.

4.5. Postprocessing and rearranging of the matrices

During the postprocessing stage, the coefficient and GOF names are replaced by user-defined names (using the custom.coef.names and custom.gof.names arguments), coefficient rows are removed by applying regular expressions to the row names (using the omit.coef argument), and coefficients/standard errors and GOF statistics are reordered according to the user’s wishes (following the reorder.coef and reorder.gof arguments).
Renaming the coefficients or GOF names may lead to duplicate entries. These duplicate rows must be conflated. For example, there may be one row with the name “duration” (with the duration variable only existing in the first model) and another row with the name “time” (with the time variable only existing in the second model). After renaming both rows to either of the two names, the two rows must be conflated such that there is only one row left with the duration coefficient in the first cell and the time coefficient in the second cell.

Rearranging the matrix also entails checking for rows with duplicate names which are in fact not complementary and rearranging them only by presenting the fullest rows first. Furthermore, there may be more than two duplicate rows with the same name and other complex configurations which are handled by \texttt{texreg}. Finally, rearranged rows are reordered to ensure that models appear as compact as possible in the table.

4.6. Aggregating the table and conflating columns

Before the data are aggregated in the final table, all coefficients, standard errors and GOF values must be formatted according to the specifications of the user: they have to be rounded (following the \texttt{digits} argument), leading zeroes must be removed if desired by the user (as set by the \texttt{leading.zero} argument), and the \texttt{numeric} values are converted into \texttt{character} strings.

The $p$ value column of the coefficient block matrix is then used to add significance stars or bold formatting depending on the \texttt{stars}, \texttt{symbol}, \texttt{star.symbol}, and \texttt{bold} arguments. In the final table, the standard error and $p$ value columns are removed, and the standard errors are either inserted between the coefficient and the stars or in separate rows below the coefficients (depending on the \texttt{single.row} argument).

4.7. Typesetting the final table

The final table is eventually translated into LATEX or HTML code and either printed to the R console or diverted to a file (depending on the \texttt{file} argument). All three functions, \texttt{texreg()}, \texttt{htmlreg()} and \texttt{screenreg()}, have their own custom arguments for the layout of the table. These specific options are listed and explained at the bottom of Table 1.

5. Examples

This section gives some practical examples. All data and model formulae were taken from the help files of the respective models and their packages for the sake of replicability.

5.1. The \texttt{screenreg()} function

First, consider a simple linear model as created by the \texttt{lm()} function:

\begin{verbatim}
R> ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
R> trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
R> group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
R> weight <- c(ctl, trt)
R> m1 <- lm(weight ~ group)
R> m2 <- lm(weight ~ group - 1)
\end{verbatim}
The coefficients, standard errors, \( p \) values etc. of model 2 can be displayed as follows:

\[
R> \text{summary(m2)}
\]

**Call:**
\[
\text{lm(formula = weight} \sim \text{ group - 1)}
\]

**Residuals:**
\[
\begin{array}{cccc}
\text{Min} & 1Q & \text{Median} & 3Q \\
-1.0710 & -0.4938 & 0.0685 & 0.2462 \\
\end{array}
\]

**Coefficients:**
\[
\begin{array}{cccc}
\text{Estimate} & \text{Std. Error} & t \text{ value} & \text{Pr(>|t|)} \\
group\text{Ctl} & 5.0320 & 0.2202 & 22.85 & 9.55e-15 *** \\
group\text{Trt} & 4.6610 & 0.2202 & 21.16 & 3.62e-14 *** \\
\end{array}
\]

---

**Signif. codes:** 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

**Residual standard error:** 0.6964 on 18 degrees of freedom
**Multiple R-squared:** 0.9818,  **Adjusted R-squared:** 0.9798
**F-statistic:** 485.1 on 2 and 18 DF,  **p-value:** < 2.2e-16

Next, load the `texreg` package. The output of the two models can be converted into a plain text table using the following command. The text output is shown below the R code.

\[
R> \text{library("texreg")}
R> \text{screenreg(list(m1, m2))}
\]

<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{(Intercept)} )</td>
<td>5.03 *** (0.22)</td>
<td></td>
</tr>
<tr>
<td>\text{groupTrt}</td>
<td>-0.37   4.66 *** (0.31) (0.22)</td>
<td></td>
</tr>
<tr>
<td>\text{groupCtl}</td>
<td>5.03 *** (0.22)</td>
<td></td>
</tr>
<tr>
<td>( R^2 )</td>
<td>0.07    0.98</td>
<td></td>
</tr>
<tr>
<td>( \text{Adj. R}^2 )</td>
<td>0.02    0.98</td>
<td></td>
</tr>
<tr>
<td>Num. obs.</td>
<td>20      20</td>
<td></td>
</tr>
</tbody>
</table>

*** \( p < 0.001 \), ** \( p < 0.01 \), * \( p < 0.05 \)

An arbitrary number of models can be handed over to the `texreg()`, `htmlreg()` or `screenreg()` function by enclosing them in a `list`. If only one model is converted, the `list` wrapper is not needed.
### Table 3: Two linear models.

<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>5.03***</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(0.22)</td>
<td></td>
</tr>
<tr>
<td>groupTrt</td>
<td>-0.37</td>
<td>4.66***</td>
</tr>
<tr>
<td></td>
<td>(0.31)</td>
<td>(0.22)</td>
</tr>
<tr>
<td>groupCtl</td>
<td></td>
<td>5.03***</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.22)</td>
</tr>
<tr>
<td>R²</td>
<td>0.07</td>
<td>0.98</td>
</tr>
<tr>
<td>Adj. R²</td>
<td>0.02</td>
<td>0.98</td>
</tr>
<tr>
<td>Num. obs.</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

***p < 0.001, **p < 0.01, *p < 0.05

5.2. `texreg()`, table environments, and layout packages

The same table can be typeset in \LaTeX{} by exchanging `screenreg()` for `texreg()`. In the following example, several additional arguments are demonstrated. The \LaTeX{} output code is shown below the R code that generates the table. The resulting table is shown in Table 3.

```r
R> texreg(list(m1, m2), dcolumn = TRUE, booktabs = TRUE,  
   + use.packages = FALSE, label = "tab:3", caption = "Two linear models.",  
   + float.pos = "h")
```

```latex
\begin{table}[h]
\begin{center}
\begin{tabular}{l D{.}{.}{2.5}@{} D{.}{.}{2.5}@{} }
\toprule
& \multicolumn{1}{c}{Model 1} & \multicolumn{1}{c}{Model 2} \\
\midrule
(Intercept) & 5.03^{***} & \ \\
& (0.22) & \ \\
groupTrt & -0.37 & 4.66^{***} \\
& (0.31) & (0.22) \\
groupCtl & & 5.03^{***} \\
& & (0.22) \\
R^2 & 0.07 & 0.98 \\
Adj. R^2 & 0.02 & 0.98 \\
Num. obs. & 20 & 20 \\
\bottomrule
\multicolumn{3}{l}{\scriptsize{\textsuperscript{***}p < 0.001, \textsuperscript{**}p < 0.01, \textsuperscript{*}p < 0.05}}
\end{tabular}
\caption{Two linear models.}
\label{tab:3}
\end{center}
\end{table}
```
The caption, label, and float position of the table are set explicitly. The `dcolumn` package is used to align coefficients at their decimal separators, and the `booktabs` package is employed to create professional horizontal rules. These arguments can be omitted if the two packages are not available (in this case, top, mid and bottom rules are replaced by conventional horizontal rules, and numeric values are horizontally aligned at the center of the column). The \texttt{\usepackage{}} declarations for the two packages are suppressed because the code has to be processed by \texttt{Sweave()}. 

In order to omit the \texttt{\begin{table}} and \texttt{\end{table}} as well as the \texttt{\begin{center}} and \texttt{\end{center}} code, the `table` and `center` arguments can be used. If `table = FALSE` and `center = FALSE` are set, only the `tabular` environment is printed, not the `table` and `center` environments. In effect, the resulting table would be printed in-line in the text. Another reason for skipping the table environment could be to finetune the environment manually.

Alternatively, the argument `sideways = TRUE` can be used to rotate the table by 90 degrees using the `sidewaystable` environment in the `rotating` package (Rahtz and Fairbairns 2008) instead of the default `table` environment.

### 5.3. Custom names, omission of terms, and customization of coefficients

Another example demonstrates how the \LaTeX{} code can be saved in an object using the `return.string` argument. The result is shown in Table 4.

```r
R> mytable <- texreg(list(m1, m2), label = "tab:4",
+                     caption = "Bolded coefficients, custom notes, three digits.",
+                     float.pos = "h", return.string = TRUE, bold = 0.05, stars = 0,
+                     custom.note = "Coefficients with $p < 0.05$ in \textbf{bold}.",
+                     digits = 3, leading.zero = FALSE, omit.coef = "Inter")
```

The table can be printed to the R console later using the \texttt{cat()} function.

The example presented above introduced several additional arguments: `bold = 0.05` formats all coefficients with \emph{p} values < 0.05 in bold; `stars = 0` means that only coefficients with \emph{p} values < 0 are decorated with a star, which effectively suppresses all significance stars in the table because negative \emph{p} values are not possible. Note that bold formatting cannot
### Table 5: Multiple model types, custom names, and single row.

<table>
<thead>
<tr>
<th></th>
<th>OLS model</th>
<th>GLS model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>5.03 (0.22)**</td>
<td>12.22 (0.66)**</td>
</tr>
<tr>
<td>sin(2 \cdot \pi \cdot \text{time})</td>
<td>-2.77 (0.65)**</td>
<td></td>
</tr>
<tr>
<td>cos(2 \cdot \pi \cdot \text{time})</td>
<td>-0.90 (0.70)</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>-0.37 (0.31)</td>
<td></td>
</tr>
</tbody>
</table>

R²: 0.07  
Num. obs: 20 308  
AIC: 1571.45  
Log Likelihood: -780.73  

**p < 0.001, *p < 0.01

Several interesting things can be noted. First, the `custom.coef.names` argument was used to relabel the coefficient rows. If there were repetitions of coefficient names in the

be used in combination with the `dcolumn` argument, so decimal mark alignment is switched off in Table 4. The `booktabs` argument was also left out to show the difference between conventional horizontal lines in Table 4 and `booktabs` rules in Table 3. The `custom.note = "Coefficients with $p < 0.05$ in bold."` argument changes the significance note below the table. The `digits = 3` argument sets three decimal places, `leading.zero = FALSE` suppresses leading zeroes before the decimal separator, and `omit.coef = "Inter"` causes all rows containing the regular expression “Inter” to be skipped from the output (here: the “(Intercept)” term). Note that more complex regular expressions are possible; for example, `omit.coef = "(Trt)|(Ctl)"` would remove all rows matching either “Trt” or “Ctl”.

5.4. Multiple model types, single.row, and custom names

Another example shows how `texreg` can deal with multiple kinds of models in the same table. The following code shows how ordinary least squares (OLS) and generalized least squares (GLS) models are matched in a single output table. The output is shown in Table 5.

```r
R> library("nlme")
R> m3 <- gls(follicles ~ sin(2 * pi * Time) + cos(2 * pi * Time), Ovary,
+   correlation = corAR1(form = ~ 1 | Mare))
R> table <- texreg(list(m1, m3), custom.coef.names = c(
+   "Intercept",
+   "Control",
+   "$\sin(2 \cdot \pi \cdot \text{time})$",
+   "$\cos(2 \cdot \pi \cdot \text{time})$",
+   custom.model.names = c("OLS model", "GLS model"),
+   reorder.coef = c(1, 3, 4, 2),
+   caption = "Multiple model types, custom names, and single row.",
+   label = "tab:5", stars = c(0.01, 0.001), dcolumn = TRUE,
+   booktabs = TRUE, use.packages = FALSE, single.row = TRUE,
+   include.adjrs = FALSE, include.bic = FALSE)
```

Several interesting things can be noted. First, the `custom.coef.names` argument was used to relabel the coefficient rows. If there were repetitions of coefficient names in the
custom.coef.names vector, texreg would try to conflate rows with identical names. In the case shown here, the two models are only matched on the intercept and the number of observations because all other rows have unique names.

Second, the custom names include \$\LaTeX$ code. Within the code, in-line math code is allowed. \$\LaTeX$ commands have to be marked by an additional backslash as an escape character, e.g., `\pi` instead of `\pi`. Text within math blocks can be included in `\mbox{}` commands.

Third, custom names were also provided for the models. Using the custom.model.names argument, the default “Model 1”, “Model 2” etc. are replaced by “OLS model” and “GLS model” in this case.

Fourth, the order of the coefficients was changed using the reorder.coef argument. The “Control” term was moved to the last position in the table.

Fifth, two significance levels (and, accordingly, a maximum of two stars) are used in the table. The stars argument takes at most four values, and when four values are specified, the lowest significance level (usually $0.05 \leq p < 0.1$) is denoted by the character specified in the symbol argument (by default a centered dot).

Sixth, the single.row argument causes the table to consume less vertical and more horizontal space because the standard errors are inserted right after the coefficients.

And seventh, the include.adjRs and include.bic arguments suppress the inclusion of the adjusted R$^2$ and BIC GOF statistics. These are model-specific arguments, which are defined in the extract() methods for `lm` and `gls`. More information about model-specific arguments can be found on the help page of the generic extract() function.

5.5. An example with robust standard errors

A common task in econometrics is to report robust—i.e., (Eicker-)Huber-White-corrected, or heteroskedasticity-consistent—standard errors using the sandwich (Zeileis 2004, 2006) and lmtest (Zeileis and Hothorn 2002) packages. The following code shows how this can be accomplished in combination with the texreg package. The resulting table is not reported here.

```R
R> library("sandwich")
R> library("lmtest")
R> hc <- vcovHC(m2)
R> ct <- coeftest(m2, vcov = hc)
R> se <- ct[, 2]
R> pval <- ct[, 4]
R> texreg(m2, override.se = se, override.pvalues = pval)
```

The standard errors and $p$ values are first extracted from the `hc` matrix and then handed over to the `texreg()` function using the override.se and override.pvalues arguments.

5.6. htmlreg(), Microsoft Word, knitr, and Markdown

The following examples show how the htmlreg() function can be used. The output code for these examples is not reported here.

The output of any texreg(), htmlreg() or screenreg() call can be written directly to a file.
<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>5.03 (0.22)***</td>
<td><strong>5.03</strong> [4.60; 5.46]*</td>
<td><strong>5.03</strong> [4.60; 5.46]*</td>
</tr>
<tr>
<td>groupTrt</td>
<td>−0.37 (0.31)</td>
<td>−0.37 [−0.98; 0.24]</td>
<td><strong>4.66</strong> [4.23; 5.09]*</td>
</tr>
<tr>
<td>groupCtl</td>
<td></td>
<td></td>
<td>5.03 [4.60; 5.46]*</td>
</tr>
</tbody>
</table>

***p < 0.001, **p < 0.01, *p < 0.05 (or 0 outside the confidence interval).

Table 6: Enforcing confidence intervals.

by adding the file argument. This is especially handy because HTML files can be read by Microsoft Word if a “.doc” file extension is added.

If the table is exported to a file, it is advisable to include the full header information of the HTML file to make sure that Microsoft Word or other programs can parse the file. An example:

```r
R> htmlreg(list(m1, m2, m3), file = "mytable.doc", inline.css = FALSE, 
+ doctype = TRUE, html.tag = TRUE, head.tag = TRUE, body.tag = TRUE)
```

The doctype argument adds the document type declaration to the first line of the HTML document. The inline.css = FALSE argument causes the function to write cascading style sheets (the table formatting code) into the <head> tag rather than into the table code. The head.tag argument actually adds such a <head> tag to the document. Similarly, the body.tag argument wraps the table in a <body> tag, and the html.tag argument encloses both—the <head> and the <body> tag—in an <html> tag. In other words, these arguments create a whole HTML document rather than merely the table code. The resulting file can be read by Microsoft Word because the HTML file has a “.doc” extension.

The htmlreg() function also works well with the knitr package for dynamic report generation (Xie 2013c). The default arguments are compatible with knitr and HTML. In addition to HTML, knitr is also compatible with Markdown, a simplified markup language. texreg can work with Markdown as well, but an additional argument should be provided to make it work:

```r
R> htmlreg(list(m1, m2, m3), star.symbol = "\*", center = TRUE)
```

The star.symbol = "\*" argument makes sure that Markdown does not interpret the significance stars as special Markdown syntax. The additional (and optional) center = TRUE argument centers the table horizontally on the page.

5.7. Confidence intervals instead of standard errors

Most model types implemented in texreg report standard errors and p values. However, some model types report confidence intervals by default. The btergm package (?) and the lme4 package (Bates et al. 2013) are two examples. If confidence intervals are preferred to standard errors but they are not available by default, the ci.force argument allows conversion of standard errors to confidence intervals. The ci.force.level argument determines at which confidence level the interval should be computed.

A star is added to estimates where the confidence interval does not contain the value given by ci.test (to remove significance stars, ci.test = NULL can be set). When the bold argument
is used in conjunction with confidence intervals, \texttt{bold} values greater than 0 cause \texttt{texreg} to print estimates in bold where the \texttt{ci.test} value is outside the confidence interval, regardless of the actual value of the \texttt{bold} argument (see Table 6):

```R
R> texreg(list(m1, m1, m2), ci.force = c(FALSE, TRUE, TRUE), ci.test = 0,
+       ci.force.level = 0.95, bold = 0.05, float.pos = "h",
+       caption = "Enforcing confidence intervals.",
+       booktabs = TRUE, use.packages = FALSE, single.row = TRUE)
```

6. Writing extensions for new models

The previous examples have demonstrated how the \texttt{texreg} package can be used to convert statistical model output into plain-text, \LaTeX, HTML, and Markdown tables. Yet, this only works with model types known to \texttt{texreg}. Accordingly, this section shows how methods for new model types can be devised and registered.

6.1. Simple extensions

A custom extract function can be easily implemented. For any model type, there exists a function which extracts the relevant information from a model. For example, the \texttt{extract()} method for \texttt{lm} objects provides coefficients and GOF statistics for \texttt{lm} objects, the \texttt{extract()} method for \texttt{lm} objects extracts $R^2$, Adj. $R^2$ and Num. obs.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coef.names</td>
<td>The names of the independent variables or coefficients.</td>
</tr>
<tr>
<td>coef</td>
<td>The actual coefficients. These values must be in the same order as the \texttt{coef.names}.</td>
</tr>
<tr>
<td>se</td>
<td>\textit{(optional)} The standard errors, which will later be put in parentheses. These values must be in the same order as the \texttt{coef.names}.</td>
</tr>
<tr>
<td>pvalues</td>
<td>\textit{(optional)} The $p$ values. They are used to add significance stars. These values must be in the same order as the \texttt{coef.names}.</td>
</tr>
<tr>
<td>ci.low</td>
<td>\textit{(optional)} Lower bounds of the confidence intervals. An alternative to the \texttt{se} slot.</td>
</tr>
<tr>
<td>ci.up</td>
<td>\textit{(optional)} Upper bounds of the confidence intervals. An alternative to the \texttt{se} slot.</td>
</tr>
<tr>
<td>gof.names</td>
<td>The names of some GOF statistics to be added to the table. For example, the \texttt{extract()} method for \texttt{lm} objects extracts $R^2$, Adj. $R^2$ and Num. obs.</td>
</tr>
<tr>
<td>gof</td>
<td>A vector of GOF statistics to be added to the table. These values must be in the same order as the \texttt{gof.names}.</td>
</tr>
<tr>
<td>gof.decimal</td>
<td>\textit{(optional)} A vector of logical (Boolean) values indicating for every GOF value whether the value should have decimal places in the output table. This is useful to avoid decimal places for the number of observations and similar count variables.</td>
</tr>
</tbody>
</table>

Table 7: Arguments of the \texttt{createTexreg()} function.
method for ‘ergm’ objects provides this information for ‘ergm’ objects, etc.
To get an overview of the model type one is interested in, it is recommended to fit a model and
examine the resulting object using the \texttt{str(model)} command, the \texttt{summary(model)} command,
the \texttt{summary(model)$coef} command, and related approaches.

Any new extract function should retrieve the data shown in Table 7 from a statistical model.
Note that \texttt{pvalues} and \texttt{gof.decimal} are optional and can be omitted. Either the \texttt{se} slot or
the \texttt{ci.low} and \texttt{ci.up} slots must contain values.

Once these data have been located and extracted, a ‘\texttt{texreg}’ object can be created and
returned to the \texttt{texreg()} function. The following code provides a simple example for ‘\texttt{lm}’
objects:

\begin{verbatim}
R> extract.lm <- function(model) {
+   s <- summary(model)
+   names <- rownames(s$coef)
+   co <- s$coef[, 1]
+   se <- s$coef[, 2]
+   pval <- s$coef[, 4]
+   +   rs <- s$r.squared
+   +   adj <- s$adj.r.squared
+   +   n <- nobs(model)
+   +   +   gof <- c(rs, adj, n)
+   +   gof.names <- c("R^2", "Adj. R^2", "Num. obs.")
+   +   +   tr <- createTexreg(
+   +       coef.names = names,
+   +       coef = co,
+   +       se = se,
+   +       pvalues = pval,
+   +       gof.names = gof.names,
+   +       gof = gof
+   +   )
+   +   return(tr)
+   }
\end{verbatim}

First, the names of the model terms, the coefficient values, the standard errors, and the
\textit{p} values are extracted from the model or its summary (they can be computed if not available).
Second, various summary statistics and GOF measures are extracted from the model object
(in this case: \textit{R}^2, Adj. \textit{R}^2 and Num. obs.) and saved in a \texttt{numeric} vector.
Third, the names of these statistics should be defined in a \texttt{character} vector. All vectors so
far should have the same length.
Fourth, a new ‘\texttt{texreg}’ object should be created, with the information extracted before in-
cluded as arguments.
Fifth, the ‘\texttt{texreg}’ object must be returned. This is necessary for the \texttt{texreg()} function to
continue processing the model.
After writing a custom function, the function has to be registered as a method for the generic
`extract()` function. In the above example, this can be achieved with the following code:

```R
R> setMethod("extract", signature = className("lm", "stats"),
+ definition = extract.lm)
```

Assume, for instance, that an extension for `clogit` objects called `extract.clogit()` is written. The `clogit()` function (and the corresponding class definition) can be found in the `survival` package (Therneau and Grambsch 2000; Therneau 2012). Then the code above should be changed as follows:

```R
R> setMethod("extract", signature = className("clogit", "survival"),
+ definition = extract.clogit)
```

After executing the definition of the function and the adjusted `setMethod()` command, `texreg` can be used with `clogit` models.

### 6.2. A complete example

The following code shows the complete `extract.lm()` function as included in the `texreg` package.

```R
R> extract.lm <- function(model, include.rsquared = TRUE,
+ include.adjrs = TRUE, include.nobs = TRUE, ...) {
+ s <- summary(model, ...)
+ names <- rownames(s$coef)
+ co <- s$coef[, 1]
+ se <- s$coef[, 2]
+ pval <- s$coef[, 4]
+ 
+ gof <- numeric()
+ gof.names <- character()
+ gof.decimal <- logical()
+ if (include.rsquared == TRUE) {
+ rs <- s$r.squared
+ gof <- c(gof, rs)
+ gof.names <- c(gof.names, "R$^2$")
+ gof.decimal <- c(gof.decimal, TRUE)
+ }
+ if (include.adjrs == TRUE) {
+ adj <- s$adj.r.squared
+ gof <- c(gof, adj)
+ gof.names <- c(gof.names, "Adj.\ R$^2$")
+ gof.decimal <- c(gof.decimal, TRUE)
+ }
+ if (include.nobs == TRUE) {
+ n <- nobs(model)
+ ...
+ ```
gof <- c(gof, n)
gof.names <- c(gof.names, "Num.\ obs.")
gof.decimal <- c(gof.decimal, FALSE)
}

tr <- createTexreg(
  coef.names = names,
  coef = co,
  se = se,
  pvalues = pval,
  gof.names = gof.names,
  gof = gof,
  gof.decimal = gof.decimal
)

return(tr)

R> setMethod("extract", signature = className("lm", "stats"),
  definition = extract.lm)

In addition to the simple example code shown above, this function has several arguments, which can be used to include or exclude various GOF or summary statistics. Additional arguments can also be used in other contexts. For example, the user can decide whether random effect variances should be included in texreg tables of ‘mer’ objects (from the lme4 package, see Bates et al. 2013) by setting the include.variance argument. Similarly, the output of ‘stergm’ models (Hunter, Handcock, Butts, Goodreau, and Morris 2008; Handcock, Hunter, Butts, Goodreau, Krivitsky, and Morris 2012) or ‘hurdle’ or ‘zeroinfl’ models (Zeileis, Kleiber, and Jackman 2008) can be typeset in two columns using the beside argument.

New extract functions and methods can be easily used locally. Once they work well, submission of new extract functions to the online forum of texreg is encouraged.

Existing functions can also be manipulated and overwritten locally in order to change the GOF statistics block.

Acknowledgments

The author would like to thank Oleg Badunenko, Tom Carsey, S. Q. Chang, Skyler Cranmer, Sebastian Daza, Christopher Gandrud, Lena Koerber, Johannes Kutsam, Fabrice Le Lec, Florian Oswald, Markus Riester, Francesco Sarracino, Matthieu Stigler, Sebastian Ugbaje, Gábor Uhrin, Antoine Vernet, Yanghao Wang, and Yihui Xie for their valuable input and ideas.

References


Dahl DB (2012). *xtable: Export Tables to \LaTeX{} or HTML*. \R{} package version 1.7-0, URL http://CRAN.R-project.org/package=xtable.


Rahtz S, Fairbairns R (2008). \textit{rotating}: A Package for Rotated Objects in \LaTeX. \LaTeX package version 2.16b, URL \url{http://www.CTAN.org/pkg/rotating}.


Therneau T (2012). \textit{survival}: A Package for Survival Analysis in S. R package version 2.36-14, URL \url{http://CRAN.R-project.org/package=survival}.


\textbf{Affiliation:}

Philip Leifeld  
Department of Government  
University of Essex  
Wivenhoe Park  
Colchester, CO4 6BW  
United Kingdom  
E-mail: philip.leifeld@essex.ac.uk  
URL: \url{http://www.philipleifeld.com}