Introduction to broom

2018-12-04

broom: let's tidy up a bit

The broom package takes the messy output of built-in functions in R, such as `lm`, `nls`, or `t.test`, and turns them into tidy data frames.

The concept of "tidy data", as introduced by Hadley Wickham, offers a powerful framework for data manipulation and analysis. That paper makes a convincing statement of the problem this package tries to solve (emphasis mine):

> **While model inputs usually require tidy inputs, such attention to detail doesn't carry over to model outputs. Outputs such as predictions and estimated coefficients aren't always tidy. This makes it more difficult to combine results from multiple models.** For example, in R, the default representation of model coefficients is not tidy because it does not have an explicit variable that records the variable name for each estimate, they are instead recorded as row names. In R, row names must be unique, so combining coefficients from many models (e.g., from bootstrap resamples, or subgroups) requires workarounds to avoid losing important information. **This knocks you out of the flow of analysis and makes it harder to combine the results from multiple models. I'm not currently aware of any packages that resolve this problem.**

broom is an attempt to bridge the gap from untidy outputs of predictions and estimations to the tidy data we want to work with. It centers around three S3 methods, each of which take common objects produced by R statistical functions (`lm`, `t.test`, `nls`, etc) and convert them into a data frame. broom is particularly designed to work with Hadley's dplyr package (see the broom+dplyr vignette for more).

broom should be distinguished from packages like reshape2 and tidyr, which rearrange and reshape data frames into different forms. Those packages perform critical tasks in tidy data analysis but focus on manipulating data frames in one specific format into another. In contrast, broom is designed to take format that is *not* in a data frame (sometimes not anywhere close) and convert it to a tidy data frame.

Tidying model outputs is not an exact science, and it's based on a judgment of the kinds of values a data scientist typically wants out of a tidy analysis (for instance, estimates, test statistics, and p-values). You may lose some of the information in the original object that you wanted, or keep more information than you need. If you think the tidy output for a model should be changed, or if you're missing a tidying function for an S3 class that you'd like, I strongly encourage you to open an issue or a pull request.

Tidying functions

This package provides three S3 methods that do three distinct kinds of tidying.

- `tidy`: constructs a data frame that summarizes the model's statistical findings. This includes coefficients and p-values for each term in a regression, per-cluster information in clustering applications, or per-test information for `multtest` functions.
- `augment`: add columns to the original data that was modeled. This includes predictions, residuals, and cluster assignments.
- `glance`: construct a concise *one-row* summary of the model. This typically contains values such as R^2, adjusted R^2, and residual standard error that are computed once for the entire model.

Note that some classes may have only one or two of these methods defined.

Consider as an illustrative example a linear fit on the built-in `mtcars` dataset.


```
lmfit <- lm(mpg ~ wt, mtcars)
lmfit
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Coefficients:
## (Intercept)           wt
##      37.285       -5.344
```

```
summary(lmfit)
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

This summary output is useful enough if you just want to read it. However, converting it to a data frame that contains all the same information, so that you can combine it with other models or do further analysis, is not trivial. You have to do `coef(summary(lmfit))` to get a matrix of coefficients, the terms are still stored in row names, and the column names are inconsistent with other packages (e.g. `Pr(>|t|)` compared to `p.value`).

Instead, you can use the `tidy` function, from the broom package, on the fit:

```
library(broom)
tidy(lmfit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    37.3      1.88      19.9  8.24e-19
## 2 wt             -5.34     0.559     -9.56 1.29e-10
```

This gives you a data.frame representation. Note that the row names have been moved into a column called `term`, and the column names are simple and consistent (and can be accessed using `$`).

Instead of viewing the coefficients, you might be interested in the fitted values and residuals for each of the original points in the regression. For this, use `augment`, which augments the original data with information

from the model:

```
augment(lmfit)
```

```
## # A tibble: 32 x 10
##    .rownames   mpg    wt .fitted .se.fit .resid   .hat .sigma .cooksd
##  * <chr>     <dbl> <dbl>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
##  1 Mazda RX4  21    2.62    23.3   0.634 -2.28  0.0433   3.07 1.33e-2
##  2 Mazda RX~  21    2.88    21.9   0.571 -0.920 0.0352   3.09 1.72e-3
##  3 Datsun 7~  22.8  2.32    24.9   0.736 -2.09  0.0584   3.07 1.54e-2
##  4 Hornet 4~  21.4  3.22    20.1   0.538  1.30  0.0313   3.09 3.02e-3
##  5 Hornet S~  18.7  3.44    18.9   0.553 -0.200 0.0329   3.10 7.60e-5
##  6 Valiant    18.1  3.46    18.8   0.555 -0.693 0.0332   3.10 9.21e-4
##  7 Duster 3~  14.3  3.57    18.2   0.573 -3.91  0.0354   3.01 3.13e-2
##  8 Merc 240D  24.4  3.19    20.2   0.539  4.16  0.0313   3.00 3.11e-2
##  9 Merc 230   22.8  3.15    20.5   0.540  2.35  0.0314   3.07 9.96e-3
## 10 Merc 280   19.2  3.44    18.9   0.553  0.300 0.0329   3.10 1.71e-4
## # ... with 22 more rows, and 1 more variable: .std.resid <dbl>
```

Note that each of the new columns begins with a `.` (to avoid overwriting any of the original columns).

Finally, several summary statistics are computed for the entire regression, such as R^2 and the F-statistic. These can be accessed with the `glance` function:

```
glance(lmfit)
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
## *     <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl>
## 1     0.753         0.745  3.05      91.4 1.29e-10     2  -80.0  166.  170.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

This distinction between the `tidy`, `augment` and `glance` functions is explored in a different context in the k-means vignette.

Other Examples

Generalized linear and non-linear models

These functions apply equally well to the output from `glm`:

```
glmfit <- glm(am ~ wt, mtcars, family="binomial")
tidy(glmfit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic p.value
##   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)    12.0       4.51      2.67 0.00759
## 2 wt             -4.02      1.44     -2.80 0.00509
```

```
augment(glmfit)
```



```
## # A tibble: 32 x 10
##    .rownames    am    wt .fitted .se.fit .resid   .hat .sigma .cooksd
##  * <chr>     <dbl> <dbl>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
##  1 Mazda RX4     1  2.62   1.50    0.918  0.635 0.126   0.803 0.0184
##  2 Mazda RX~     1  2.88   0.471   0.676  0.985 0.108   0.790 0.0424
##  3 Datsun 7~     1  2.32   2.70    1.28   0.360 0.0963  0.810 0.00394
##  4 Hornet 4~     0  3.22  -0.897   0.601 -0.827 0.0744  0.797 0.0177
##  5 Hornet S~     0  3.44  -1.80    0.749 -0.553 0.0681  0.806 0.00647
##  6 Valiant       0  3.46  -1.88    0.767 -0.532 0.0674  0.807 0.00590
##  7 Duster 3~     0  3.57  -2.33    0.878 -0.432 0.0625  0.809 0.00348
##  8 Merc 240D     0  3.19  -0.796   0.593 -0.863 0.0755  0.796 0.0199
##  9 Merc 230      0  3.15  -0.635   0.586 -0.922 0.0776  0.793 0.0242
## 10 Merc 280      0  3.44  -1.80    0.749 -0.553 0.0681  0.806 0.00647
## # ... with 22 more rows, and 1 more variable: .std.resid <dbl>
```

```
glance(glmfit)
```

```
## # A tibble: 1 x 7
##   null.deviance df.null logLik   AIC   BIC deviance df.residual
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int>
## 1          43.2      31  -9.59  23.2  26.1     19.2          30
```

Note that the statistics computed by `glance` are different for `glm` objects than for `lm` (e.g. deviance rather than R^2):

These functions also work on other fits, such as nonlinear models (`nls`):

```
nlsfit <- nls(mpg ~ k / wt + b, mtcars, start=list(k=1, b=0))
tidy(nlsfit)
```

```
## # A tibble: 2 x 5
##   term  estimate std.error statistic  p.value
##   <chr>    <dbl>     <dbl>     <dbl>    <dbl>
## 1 k         45.8      4.25      10.8  7.64e-12
## 2 b          4.39     1.54       2.85 7.74e- 3
```

```
augment(nlsfit, mtcars)
```

```
## # A tibble: 32 x 14
##    .rownames   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear
##    <chr>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Mazda RX4  21       6  160    110  3.9   2.62  16.5     0     1     4
##  2 Mazda RX~  21       6  160    110  3.9   2.88  17.0     0     1     4
##  3 Datsun 7~  22.8     4  108     93  3.85  2.32  18.6     1     1     4
##  4 Hornet 4~  21.4     6  258    110  3.08  3.22  19.4     1     0     3
##  5 Hornet S~  18.7     8  360    175  3.15  3.44  17.0     0     0     3
##  6 Valiant    18.1     6  225    105  2.76  3.46  20.2     1     0     3
##  7 Duster 3~  14.3     8  360    245  3.21  3.57  15.8     0     0     3
##  8 Merc 240D  24.4     4  147.    62  3.69  3.19  20       1     0     4
##  9 Merc 230   22.8     4  141.    95  3.92  3.15  22.9     1     0     4
## 10 Merc 280   19.2     6  168.   123  3.92  3.44  18.3     1     0     4
## # ... with 22 more rows, and 3 more variables: carb <dbl>, .fitted <dbl>,
## #   .resid <dbl>
```

```
glance(nlsfit)
```

```
## # A tibble: 1 x 8
##   sigma isConv       finTol logLik   AIC   BIC deviance df.residual
##   <dbl> <lgl>         <dbl>  <dbl> <dbl> <dbl>    <dbl>       <int>
## 1  2.77 TRUE   0.0000000288  -77.0  160.  164.     231.          30
```

Hypothesis testing

The `tidy` function can also be applied to `htest` objects, such as those output by popular built-in functions like `t.test`, `cor.test`, and `wilcox.test`.

```
tt <- t.test(wt ~ am, mtcars)
tidy(tt)
```

```
## # A tibble: 1 x 10
##   estimate estimate1 estimate2 statistic p.value parameter conf.low
##      <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>
## 1     1.36      3.77      2.41      5.49 6.27e-6      29.2    0.853
## # ... with 3 more variables: conf.high <dbl>, method <chr>,
## #   alternative <chr>
```

Some cases might have fewer columns (for example, no confidence interval):

```
wt <- wilcox.test(wt ~ am, mtcars)
tidy(wt)
```

```
## # A tibble: 1 x 4
##   statistic   p.value method                                  alternative
##       <dbl>     <dbl> <chr>                                   <chr>
## 1      230. 0.0000435 Wilcoxon rank sum test with continuity ~ two.sided
```

Since the `tidy` output is already only one row, `glance` returns the same output:

```
glance(tt)
```

```
## # A tibble: 1 x 10
##   estimate estimate1 estimate2 statistic p.value parameter conf.low
##      <dbl>     <dbl>     <dbl>     <dbl>   <dbl>     <dbl>    <dbl>
## 1     1.36      3.77      2.41      5.49 6.27e-6      29.2    0.853
## # ... with 3 more variables: conf.high <dbl>, method <chr>,
## #   alternative <chr>
```

```
glance(wt)
```

```
## # A tibble: 1 x 4
##   statistic   p.value method                                  alternative
```



```
##       <dbl>     <dbl> <chr>                                   <chr>
## 1      230. 0.0000435 Wilcoxon rank sum test with continuity ~ two.sided
```

`augment` method is defined only for chi-squared tests, since there is no meaningful sense, for other tests, in which a hypothesis test produces output about each initial data point.

```
chit <- chisq.test(xtabs(Freq ~ Sex + Class, data = as.data.frame(Titanic)))
tidy(chit)
```

```
## # A tibble: 1 x 4
##   statistic  p.value parameter method
##       <dbl>    <dbl>     <int> <chr>
## 1      350. 1.56e-75         3 Pearson's Chi-squared test
```

```
augment(chit)
```

```
## # A tibble: 8 x 9
##   Sex   Class .observed  .prop .row.prop .col.prop .expected .residuals
##   <fct> <fct>     <dbl>  <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
## 1 Male  1st         180 0.0818    0.104     0.554      256.       -4.73
## 2 Fema~ 1st         145 0.0659    0.309     0.446       69.4       9.07
## 3 Male  2nd         179 0.0813    0.103     0.628      224.       -3.02
## 4 Fema~ 2nd         106 0.0482    0.226     0.372       60.9       5.79
## 5 Male  3rd         510 0.232     0.295     0.722      555.       -1.92
## 6 Fema~ 3rd         196 0.0891    0.417     0.278      151.        3.68
## 7 Male  Crew        862 0.392     0.498     0.974      696.        6.29
## 8 Fema~ Crew         23 0.0104    0.0489    0.0260     189.      -12.1
## # ... with 1 more variable: .stdres <dbl>
```

Conventions

In order to maintain consistency, we attempt to follow some conventions regarding the structure of returned data.

All functions

- The output of the `tidy`, `augment` and `glance` functions is *always* a data frame.
- The output never has rownames. This ensures that you can combine it with other tidy outputs without fear of losing information (since rownames in R cannot contain duplicates).
- Some column names are kept consistent, so that they can be combined across different models and so that you know what to expect (in contrast to asking "is it `pval` or `PValue`?" every time). The examples below are not all the possible column names, nor will all tidy output contain all or even any of these columns.

tidy functions

- Each row in a `tidy` output typically represents some well-defined concept, such as one term in a regression, one test, or one cluster/class. This meaning varies across models but is usually self-evident. The one thing each row cannot represent is a point in the initial data (for that, use the `augment` method).
- Common column names include:

    - `term`"" the term in a regression or model that is being estimated.
    - `p.value`: this spelling was chosen (over common alternatives such as `pvalue`, `PValue`, or `pval`) to be consistent with functions in R's built-in `stats` package
    - `statistic` a test statistic, usually the one used to compute the p-value. Combining these across many sub-groups is a reliable way to perform (e.g.) bootstrap hypothesis testing
    - `estimate`
    - `conf.low` the low end of a confidence interval on the `estimate`
    - `conf.high` the high end of a confidence interval on the `estimate`
    - `df` degrees of freedom

augment functions

- `augment(model, data)` adds columns to the original data.
    - If the `data` argument is missing, `augment` attempts to reconstruct the data from the model (note that this may not always be possible, and usually won't contain columns not used in the model).
- Each row in an `augment` output matches the corresponding row in the original data.
- If the original data contained rownames, `augment` turns them into a column called `.rownames`.
- Newly added column names begin with `.` to avoid overwriting columns in the original data.
- Common column names include:
    - `.fitted`: the predicted values, on the same scale as the data.
    - `.resid`: residuals: the actual y values minus the fitted values
    - `.cluster`: cluster assignments

glance functions

- `glance` always returns a one-row data frame.
    - The only exception is that `glance(NULL)` returns an empty data frame.
- We avoid including arguments that were *given* to the modeling function. For example, a `glm` glance output does not need to contain a field for `family`, since that is decided by the user calling `glm` rather than the modeling function itself.
- Common column names include:
    - `r.squared` the fraction of variance explained by the model
    - `adj.r.squared` R^2 adjusted based on the degrees of freedom
    - `sigma` the square root of the estimated variance of the residuals