Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

dotwhisker: Dot-and-Whisker Plots of Regression Results

Frederick Solt
University of Iowa

Yue Hu
Tsinghua University

Abstract

The dotwhisker package provides a quick and easy way to create highly customizable dot-and-whisker plots for presenting and comparing the output of regression models. It can be used to plot estimates of coefficients or other quantities of interest (e.g., predicted probabilities) within a single model or across different models.

Keywords: dotwhisker, dot-and-whisker plots, regression models, estimates, confidence intervals, dwplot, ggplot.

1. Introduction: Creating highly customizable dot-and-whisker plots

Graphs have long been known to be a more compact and effective means of conveying the results of regression models than tables (Gelman, Pasarica, and Dodhia 2002; Kastellec and Leoni 2007), but many researchers continue to list these results in tables. The reason, (Kastellec and Leoni 2007) surmised, is "simply put, it takes much greater effort to produce a quality graph than a table." The 'dotwhisker' package provides a quick and easy way to create highly customizable dot-and-whisker plots for presenting and comparing the output of regression models. It can be used to plot estimates of coefficients or other quantities of interest (e.g., predicted probabilities) within a single model or across different models: the estimates are presented as dots and their confidence intervals as whiskers (see Kastellec and Leoni 2007, 765-767).

Users can easily customize the content of their plots: presenting multiple models or results for a subset of variables is easy. Moreover, by outputting 'ggplot' objects (Wickham 2009), 'dotwhisker' allows users to further modify the format of their plots in nearly infinite ways.

This article illustrates basic use of the package's mainstay function, `dwplot`, for creating dot-and-whisker plots from model objects; more advanced uses of `dwplot` that employ tidy data frames as input; and, finally, some useful variations of dot-and-whisker plots that are easily made using other functions in the `dotwhisker` package.

2. Basic Use: Plotting Results from One or More Regression Models

Generating dot-and-whisker plots from model objects generated by the most commonly used regression functions is straightforward. To make a basic dot-and-whisker plot of any single model object of a class supported by parameters::parameters, simply pass it to `dwplot`. For these examples, we'll use the `mtcars` dataset extracted from the 1974 volume of the US magazine, *Motor Trend*.

```
#Package preload
library(dotwhisker)
```

```
Loading required package: ggplot2
```

```
library(dplyr)
```

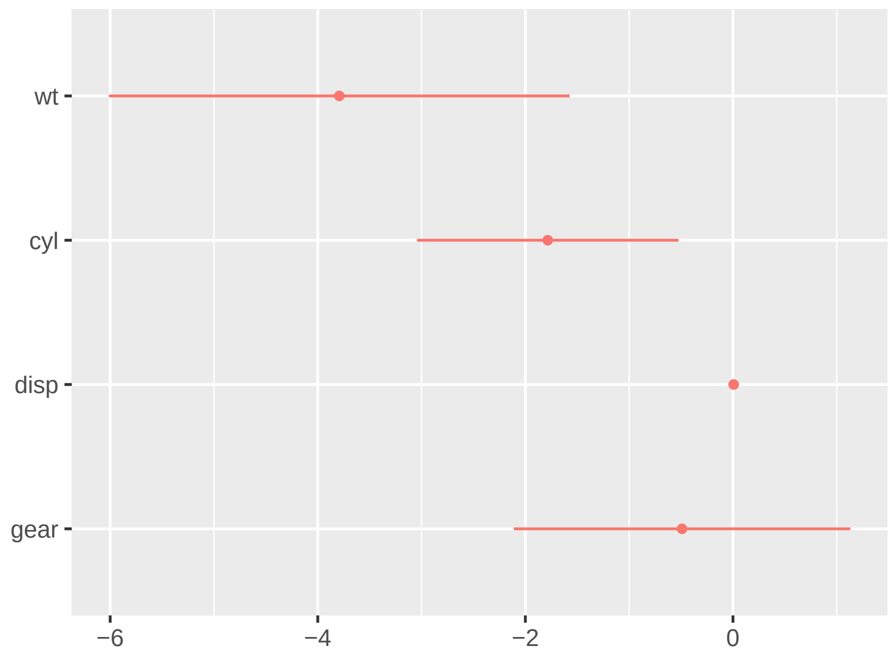
```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

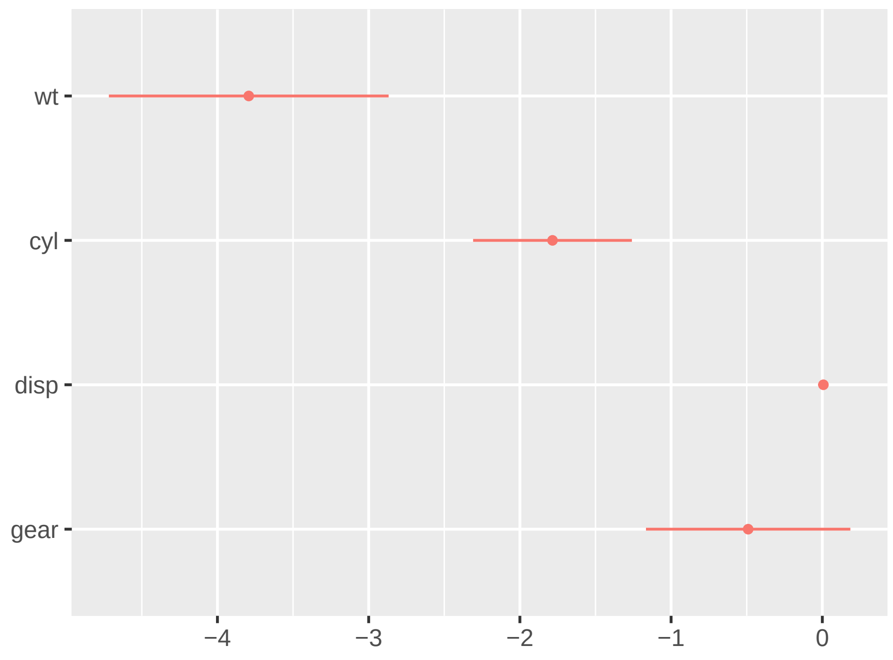
```
# run a regression compatible with tidy
m1 <- lm(mpg ~ wt + cyl + disp + gear, data = mtcars)
```

```
# draw a dot-and-whisker plot
dwplot(m1)
```



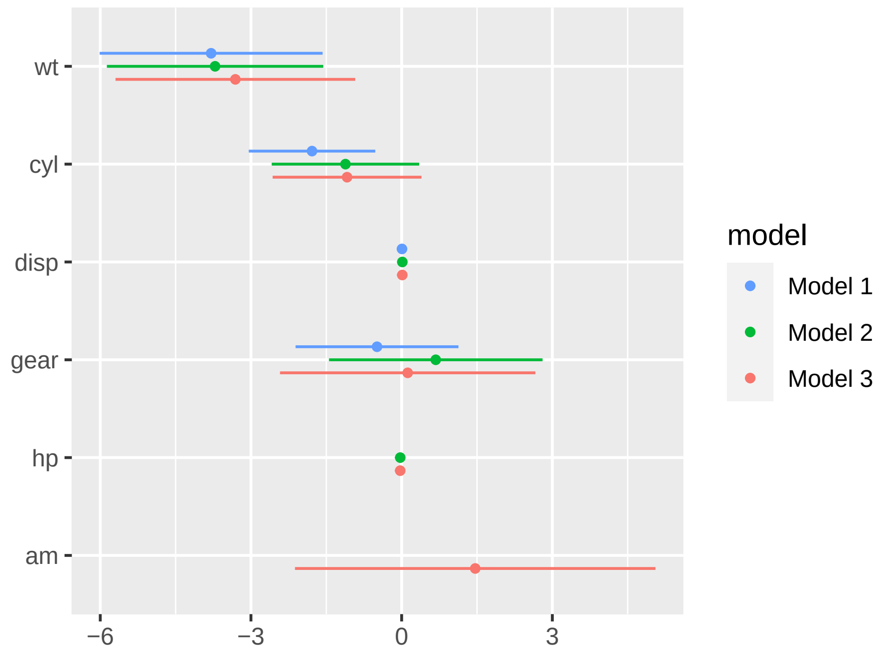
By default, the whiskers span the 95% confidence interval. To change the width of the confidence interval, specify a `ci` argument to pass to `parameters::parameters()`:

```
dwplot(m1, ci = .60) +   # using 60% of confidence intervals
    theme(legend.position = "none")
```



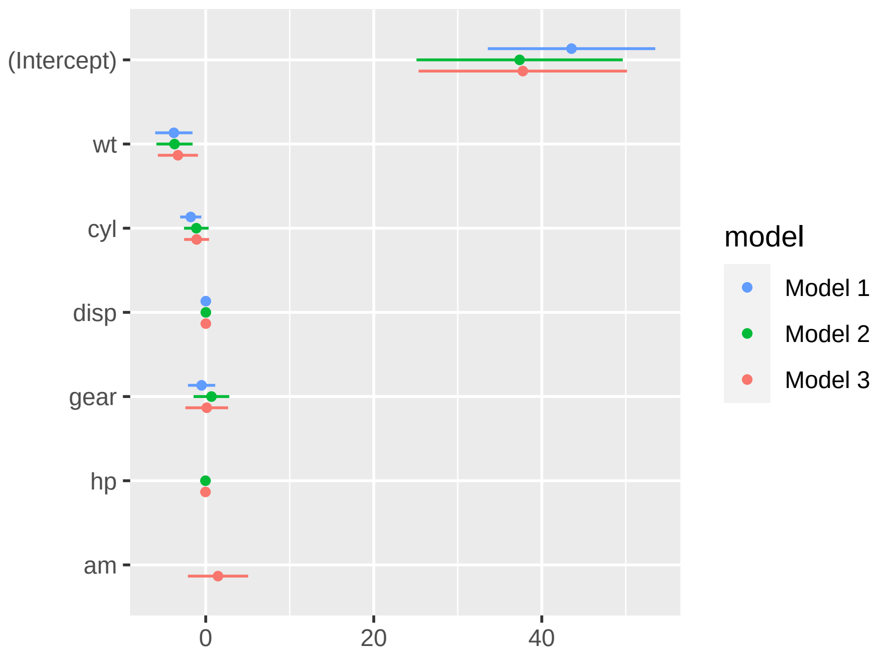
Plotting the results of more than one regression model is just as easy. Just pass the model objects to `dwplot` as a list. The `dodge_size` argument is used to adjust the space between the estimates of one variable when multiple models are presented in a single plot. Its default value of .4 will usually be fine, but, depending on the dimensions of the desired plot, more pleasing results may be achieved by setting `dodge_size` to lower values when the plotted results include a relatively small number of predictors or to higher values when many models appear on the same plot.

```
m2 <- update(m1, . ~ . + hp) # add another predictor
m3 <- update(m2, . ~ . + am) # and another

dwplot(list(m1, m2, m3))
```



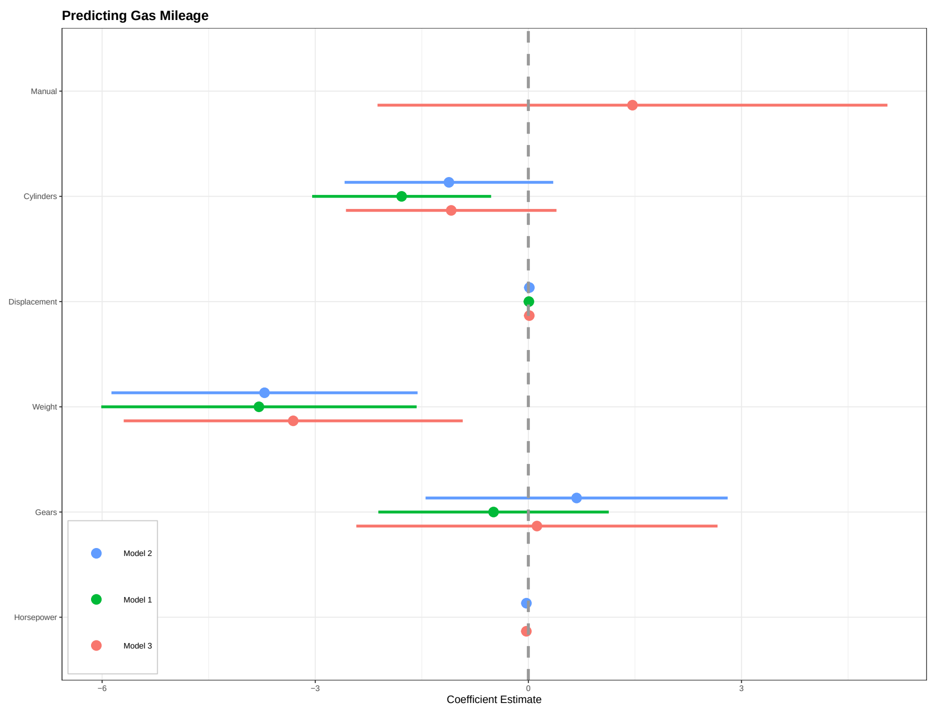
Model intercepts are rarely theoretically interesting (see Kastellec and Leoni 2007, 765), so they are excluded by `dwplot` by default. They are easy to include if desired, however, by setting the `show_intercept` argument to true.

```
dwplot(list(m1, m2, m3), show_intercept = TRUE)
```



Users are free to customize the order of the models and variables to present with the arguments `model_order` and `vars_order`. Moreover, the output of `dwplot` is a `ggplot` object. Add or change any `ggplot` layers after calling `dwplot` to achieve the desired presentation. Users can provide a named character vector to `relabel_predictors`, a `dotwhisker` function, conveniently renames the predictors. Note that both `vars_order` and `relabel_predictors` changes the presenting order of variables. When both are used, the later overwrites the former.

```
dwplot(list(m1, m2, m3),
       vline = geom_vline(
           xintercept = 0,
           colour = "grey60",
           linetype = 2
       ),
       vars_order = c("am", "cyl", "disp", "gear", "hp", "wt"),
       model_order = c("Model 2", "Model 1", "Model 3")
       ) %>% # plot line at zero _behind_coefs
    relabel_predictors(
        c(
            am = "Manual",
            cyl = "Cylinders",
            disp = "Displacement",
            wt = "Weight",
            gear = "Gears",
            hp = "Horsepower"
        )
    ) +
    theme_bw(base_size = 4) +

    # Setting `base_size` for fit the theme
    # No need to set `base_size` in most usage
    xlab("Coefficient Estimate") + ylab("") +

    geom_vline(xintercept = 0,
               colour = "grey60",
               linetype = 2) +

    ggtitle("Predicting Gas Mileage") +

    theme(
        plot.title = element_text(face = "bold"),
        legend.position = c(0.007, 0.01),
        legend.justification = c(0, 0),
        legend.background = element_rect(colour = "grey80"),
        legend.title = element_blank()
    )
```



There are many other packages (e.g., `coefplot`) that have the ability to draw dot-and-whisker plots of at least a single set of regression results taking model objects as input. While this is very convenient, it also comes with some severe limitations. First, many less common model objects are not supported. Second, rescaling coefficients, reordering them, or just plotting a subset of results is typically impossible. And third, quantities of interest beyond coefficient estimates cannot be plotted. The dotwhisker package avoids all of these limitations by optionally taking as its input a tidy data frame of estimates drawn from a model object rather than the model object itself.

3. Advanced Use: Decoration and Modification

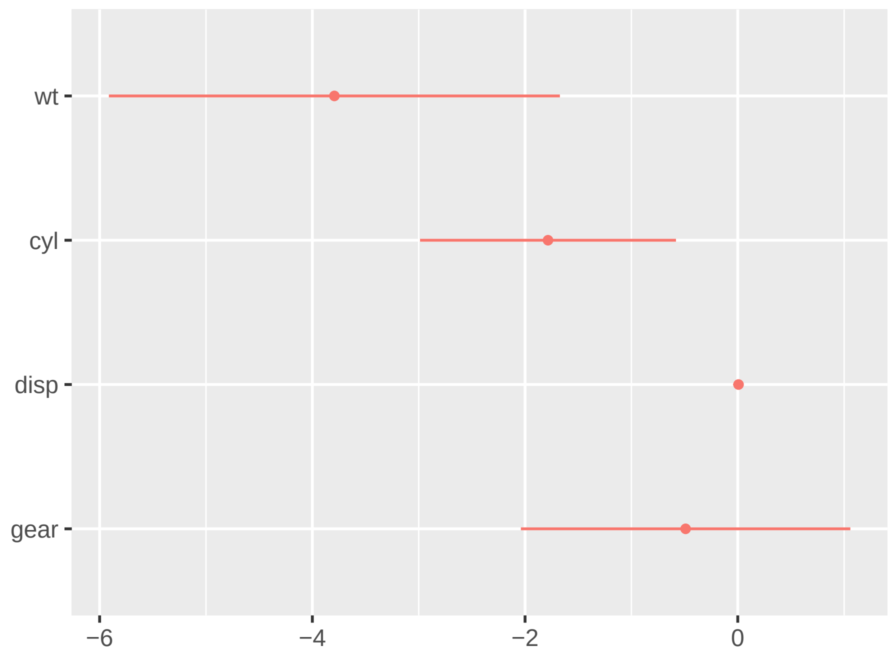
3.1. Plotting Results Stored in a Tidy Data Frame

In addition to model objects, the input for `dwplot` may be a tidy data frame that includes three columns: `term`, that is, the variable name; `estimate`, the regression coefficients or other quantity of interest; and `std.error`, the standard errors associated with these estimates. In place of `std.error` one may substitute `conf.low`, the lower bounds of the confidence intervals of the estimates, and `conf.high`, the corresponding upper bounds. As noted above, "tidy data" (Robinson 2015) refers such a data frame of estimates for many common classes of model objects (indeed, `dwplot` was written to expect a data.frame with the columns `term`, `estimate`, and `std.error`). When more than one model's results are to be plotted, an additional column `model` that identifies the two models must be added to the data frame (alternate names for this last column may be specified by using the `model_name` argument).

```
# regression compatible with tidy
m1_df <- broom::tidy(m1) # create data.frame of regression results

m1_df # a tidy data.frame available for dwplot
```

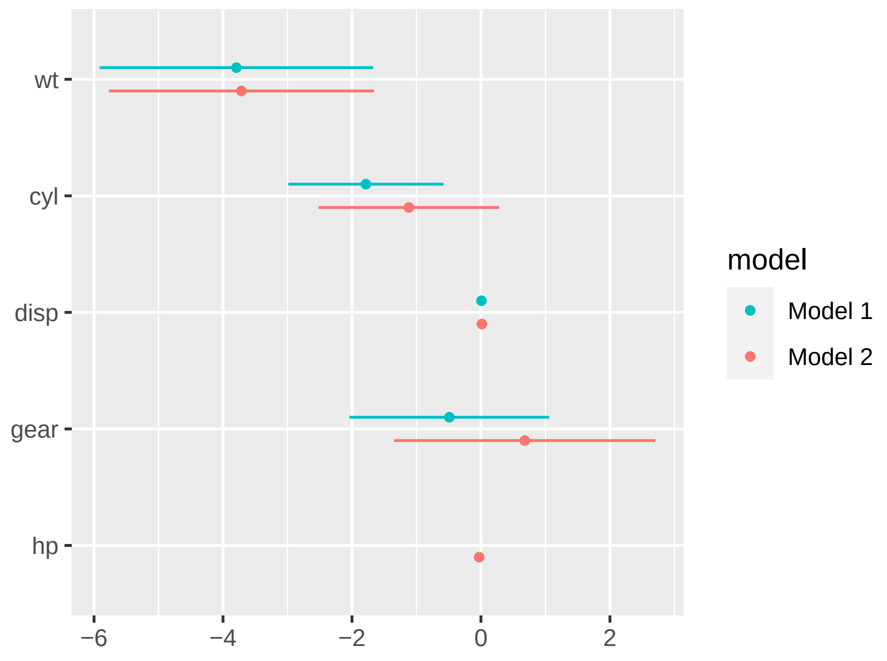
```
# A tibble: 5 x 5
  term        estimate std.error statistic       p.value
  <chr>          <dbl>     <dbl>     <dbl>         <dbl>
1 (Intercept) 43.5        4.86       8.96  0.00000000142
2 wt          -3.79       1.08      -3.51  0.00161
3 cyl         -1.78       0.614     -2.91  0.00722
4 disp         0.00694    0.0120     0.578 0.568
5 gear        -0.490      0.790     -0.621 0.540
```

```
dwplot(m1_df) #same as dwplot(m1)
```



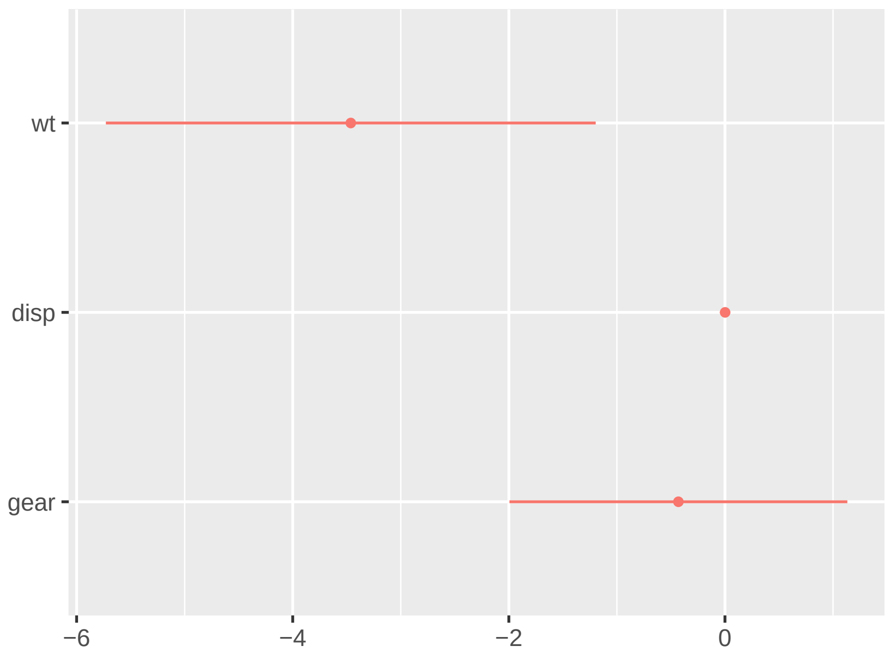
Using `tidy` can be helpful when one wishes to omit certain model estimates from the plot. To illustrate, we drop the intercept (although this is in fact done by `dwplot` automatically by default):

```
m1_df <-
    broom::tidy(m1) %>% filter(term != "(Intercept)") %>% mutate(model = "Model 1")

m2_df <-
    broom::tidy(m2) %>% filter(term != "(Intercept)") %>% mutate(model = "Model 2")


two_models <- rbind(m1_df, m2_df)


dwplot(two_models)
```



You can also filter by regular expressions. This can be helpful, for instance, if a model contains a factor with many levels (e.g., a dummy variable for each country) which you might not want to include in your plot.

```
# Transform cyl to factor variable in the data
m_factor <-
    lm(mpg ~ wt + cyl + disp + gear, data = mtcars %>% mutate(cyl = factor(cyl)))


# Remove all model estimates that start with cyl*
m_factor_df <- broom::tidy(m_factor) %>%
    filter(!grepl('cyl*', term))


dwplot(m_factor_df)
```

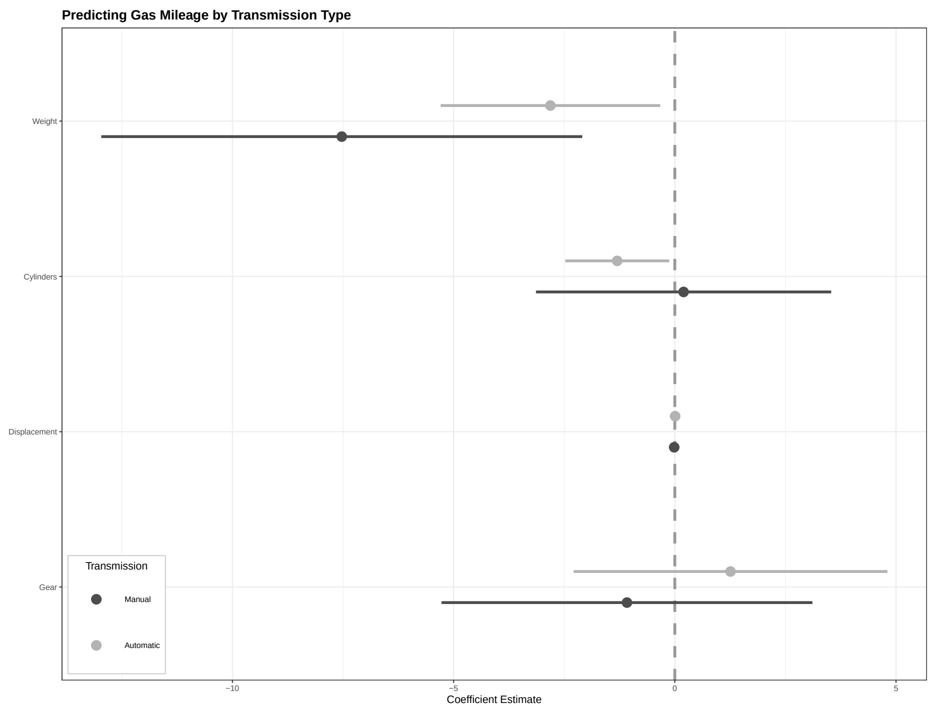


It can also be convenient to build a tidy data frame of regression results directly, that is, without first creating model objects:

```
# Run model on subsets of data, save results as tidy df, make a model variable, and relabe
by_trans <- mtcars %>%
    group_by(am) %>%                                          # group data by trans
    do(broom::tidy(lm(mpg ~ wt + cyl + disp + gear, data = .))) %>% # run model on each gr
    rename(model = am) %>%                                      # make model variable
    relabel_predictors(c(
        wt = "Weight",
        # relabel predictors
        cyl = "Cylinders",
        disp = "Displacement",
        gear = "Gear"
    ))

by_trans
```

```
# A tibble: 10 x 6
# Groups:   model [2]
   model term         estimate std.error statistic  p.value
   <dbl> <fct>           <dbl>     <dbl>     <dbl>    <dbl>
 1     0 Weight       -2.81        1.27      -2.22  0.0434
 2     0 Cylinders    -1.30        0.599     -2.17  0.0473
 3     0 Displacement  0.00692     0.0129     0.534 0.601
 4     0 Gear          1.26        1.81       0.696 0.498
 5     0 (Intercept)  30.7         7.41       4.15  0.000986
 6     1 Weight       -7.53       2.77      -2.71  0.0265
 7     1 Cylinders     0.198      1.70       0.116 0.910
 8     1 Displacement -0.0146     0.0315    -0.464 0.655
 9     1 Gear         -1.08       2.14      -0.506 0.627
10     1 (Intercept)  48.4       11.1        4.34  0.00247
```

```
dwplot(by_trans,
       vline = geom_vline(
           xintercept = 0,
           colour = "grey60",
           linetype = 2
       )) + # plot line at zero _behind_ coefs
    theme_bw(base_size = 4) + xlab("Coefficient Estimate") + ylab("") +
    ggtitle("Predicting Gas Mileage by Transmission Type") +
    theme(
        plot.title = element_text(face = "bold"),
        legend.position = c(0.007, 0.01),
        legend.justification = c(0, 0),
        legend.background = element_rect(colour = "grey80"),
        legend.title.align = .5
    ) +
    scale_colour_grey(
        start = .3,
        end = .7,
        name = "Transmission",
        breaks = c(0, 1),
        labels = c("Automatic", "Manual")
    )
```



Also note in the above example the additional manner of using the `relabel_predictors` function: in addition to being used on the `ggplot` object created by `dwplot` before further customization, it may also be used on a tidy data frame before it is passed to `dwplot`.

Additionally, one can change the shape of the point estimate instead of using different colors. This can be useful, for example, when a plot needs to be printed in black and white. Here we also vary the linetype of the whiskers.

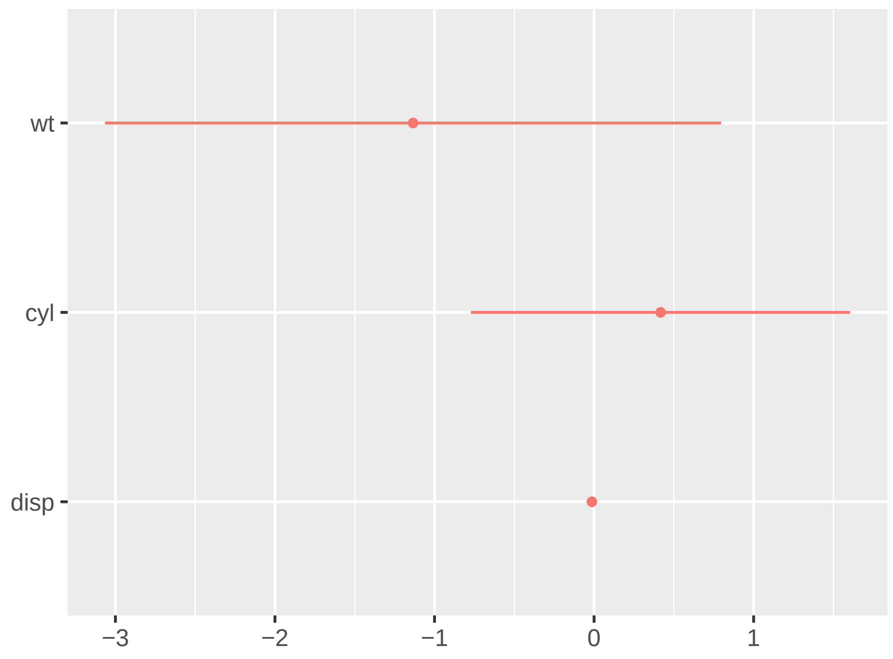
```
    vline = geom_vline(
        xintercept = 0,
        colour = "grey60",
        linetype = 2
    ),
    # plot line at zero _behind_ coefs
    dot_args = list(aes(shape = model)),
    whisker_args = list(aes(linetype = model))
) +
    theme_bw(base_size = 4) + xlab("Coefficient Estimate") + ylab("") +
    ggtitle("Predicting Gas Mileage by Transmission Type") +
    theme(
        plot.title = element_text(face = "bold"),
        legend.position = c(0.007, 0.01),
        legend.justification = c(0, 0),
        legend.background = element_rect(colour = "grey80"),
        legend.title.align = .5
    ) +
    scale_colour_grey(
        start = .1,
        end = .1,
        # if start and end same value, use same colour for all models
        name = "Model",
        breaks = c(0, 1),
        labels = c("Automatic", "Manual")
    ) +
    scale_shape_discrete(
        name = "Model",
        breaks = c(0, 1),
        labels = c("Automatic", "Manual")
    ) +
    guides(
        shape = guide_legend("Model"),
        colour = guide_legend("Model")
    ) # Combine the legends for shape and color
```

It is also easy to plot classes of model objects that are not supported by `tidy` or `parameters::parameters`: one simply extracts the results from the model object and builds the data frame to pass to `dwplot` oneself. Many functions generate results that can be extracted by `coef()`.

```
# the ordinal regression model is not supported by tidy
m4 <- ordinal::clm(factor(gear) ~ wt + cyl + disp, data = mtcars)

m4_df <- coef(summary(m4)) %>%
    data.frame() %>%
    tibble::rownames_to_column("term") %>%
    rename(estimate = Estimate, std.error = Std..Error)

m4_df
```

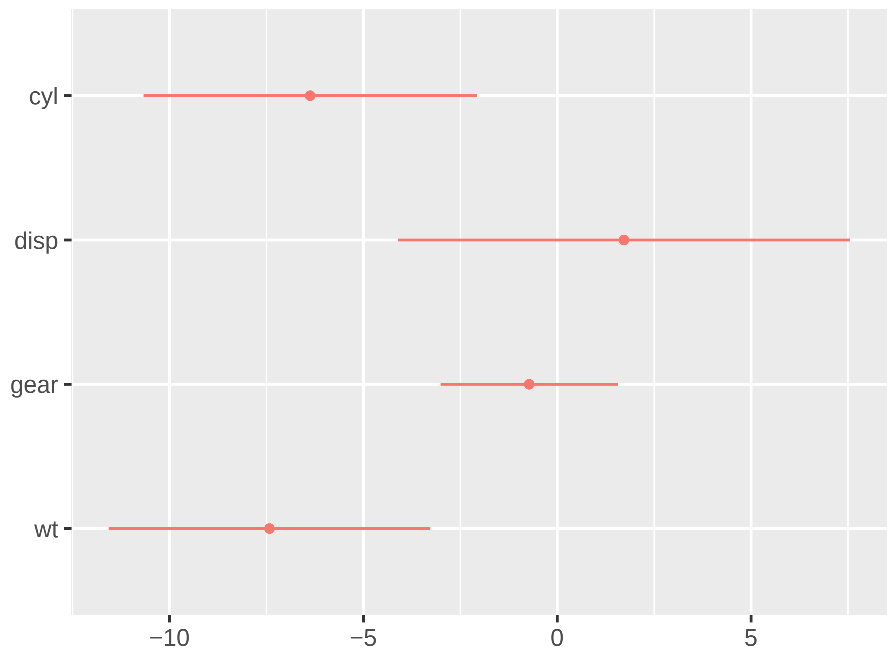
```
  term    estimate  std.error    z.value  Pr...z..
1  3|4 -4.03517968 2.45869171 -1.6411898 0.1007580
2  4|5 -1.37662018 2.28622404 -0.6021370 0.5470829
3   wt -1.13452561 0.98498075 -1.1518252 0.2493929
4  cyl  0.41701081 0.60620009  0.6879095 0.4915098
5 disp -0.01343896 0.01188167 -1.1310664 0.2580271
```

```
dwplot(m4_df)
```



Working with a tidy data frame, it is similarly straightforward to plot just a subset of results or to rescale or reorder coefficients. One often desirable manipulation is to standardize the scales of variables. (Gelman 2008), for example, suggests rescaling ordinal and continuous predictors by two standard deviations to facilitate comparison with dichotomous predictors. Although this can of course be done before model estimation, it can be more convenient to simply rescale the coefficients afterwards; the `by_2sd` function, which takes as arguments a data frame of estimates along with the original data frame upon which the model was based, automates this calculation.

```
# Customize the input data frame
m1_df_mod <-
    m1_df %>%                          # the original tidy data.frame
    by_2sd(mtcars) %>%                 # rescale the coefficients
    arrange(term)                      # alphabetize the variables

m1_df_mod  # rescaled, with variables reordered alphabetically

# A tibble: 4 x 7
  term  estimate std.error statistic p.value model   by_2sd
  <chr>    <dbl>     <dbl>     <dbl>   <dbl> <chr>   <lgl>
1 cyl     -6.37       2.19    -2.91  0.00722 Model 1 TRUE
2 disp     1.72       2.98     0.578 0.568   Model 1 TRUE
3 gear    -0.724      1.17    -0.621 0.540   Model 1 TRUE
4 wt      -7.42       2.12    -3.51  0.00161 Model 1 TRUE

dwplot(m1_df_mod)
```

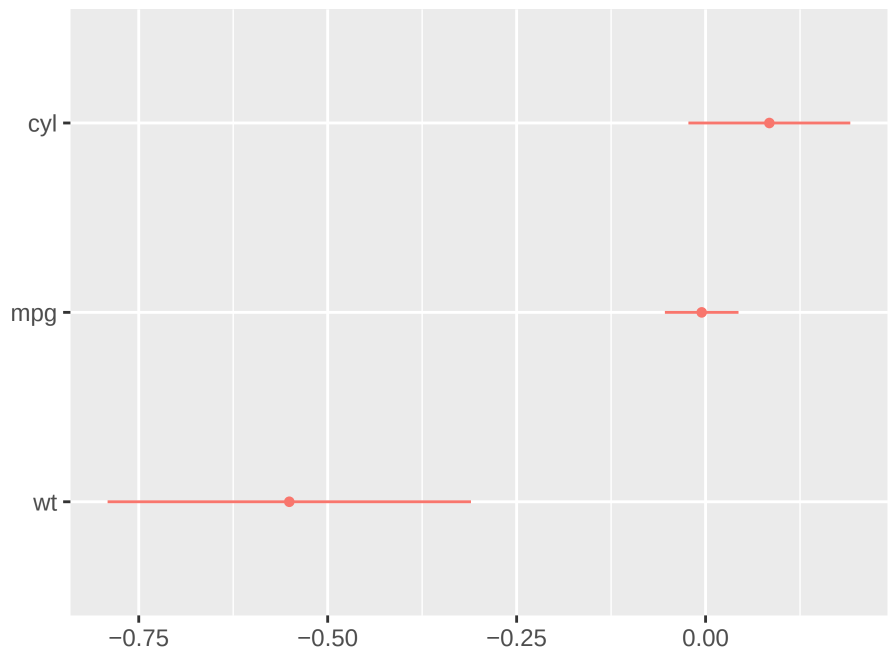


An input data frame can also be constructed from estimates of other quantities of interest, such as margins, odds ratios, or predicted probabilities, rather than coefficients.

```
# Create a data.frame of marginal effects
m5 <- glm(am ~ wt + cyl + mpg, data = mtcars, family = binomial)

m5_margin <- margins::margins(m5) %>%
    summary() %>%
    dplyr::rename(
        term = factor,
        estimate = AME,
        std.error = SE,
        conf.low = lower,
        conf.high = upper,
        statistic = z,
        p.value = p
    )

m5_margin
```

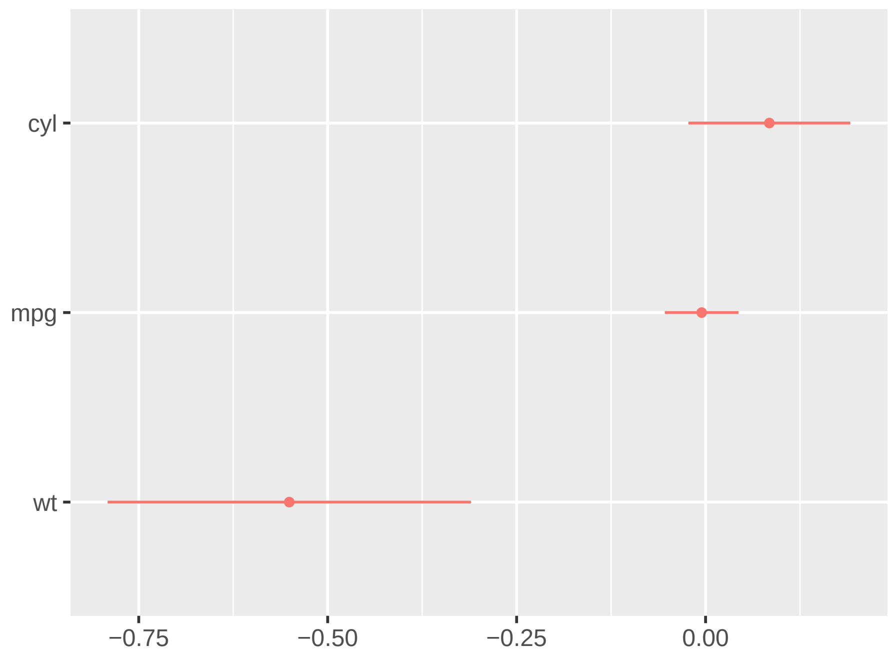
```
 term   estimate std.error statistic   p.value conf.low conf.high
  cyl   0.084403   0.05464    1.5447 1.224e-01 -0.02269   0.19150
  mpg  -0.005115   0.02485   -0.2058 8.369e-01 -0.05382   0.04359
   wt  -0.550800   0.12266   -4.4904 7.107e-06 -0.79121  -0.31039
```

```
dwplot(m5_margin)
```

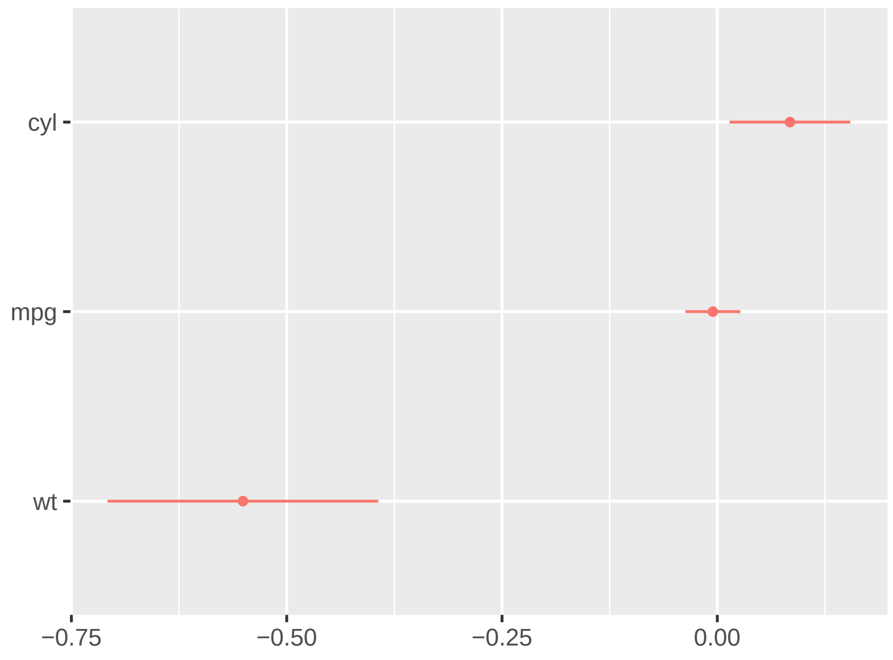


Since the marginal effects are widely used in nowadays social science, `dwplot` offers a convenient shortcut `margins`. Users can plot the average marginal effects (AME) instead of regression coefficients by setting `margins` to TRUE. The confidence intervals of the AME can be set by the argument `ci`.

```
dwplot(m5, margins = TRUE)
```



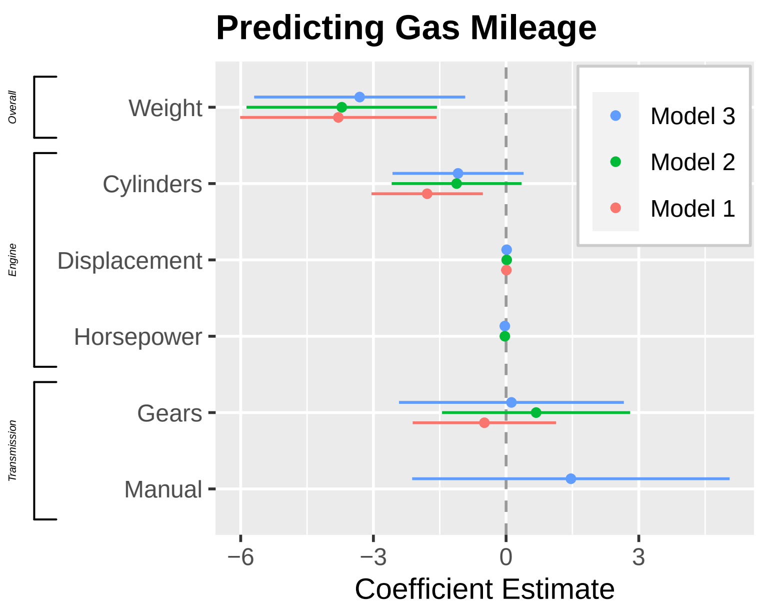
```
dwplot(m5, margins = TRUE, ci = .8)
```



3.2. Grouping Predictors

It is frequently desirable to convey that the predictors in a model depicted in a dot-and-whisker plot form groups of some sort. This can be achieved by passing the finalized plot to the `add_brackets` function. To pass the finalized plot to `add_brackets` without creating an intermediate object, simply wrap the code that generates it in braces (`{` and `}`):

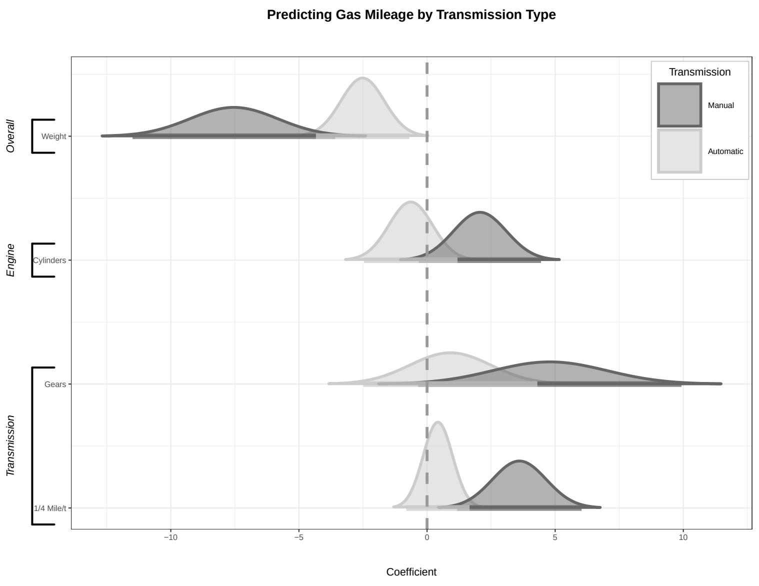
```
# Create list of brackets (label, topmost included predictor, bottommost included predicto
three_brackets <- list(
    c("Overall", "Weight", "Weight"),
    c("Engine", "Cylinders", "Horsepower"),
    c("Transmission", "Gears", "Manual")
)

{
    dwplot(list(m1, m2, m3),
           vline = geom_vline(
               xintercept = 0,
               colour = "grey60",
               linetype = 2
           )) %>% # plot line at zero _behind_ coefs
        relabel_predictors(
            c(
                wt = "Weight",
                # relabel predictors
                cyl = "Cylinders",
                disp = "Displacement",
                hp = "Horsepower",
                gear = "Gears",
                am = "Manual"
            )
        ) + xlab("Coefficient Estimate") + ylab("") +
        ggtitle("Predicting Gas Mileage") +
        theme(
            plot.title = element_text(face = "bold"),
            legend.position = c(0.993, 0.99),
            legend.justification = c(1, 1),
            legend.background = element_rect(colour = "grey80"),
            legend.title = element_blank()
        )
} %>%
    add_brackets(three_brackets, fontSize = 0.3)
```



3.3. Presenting Regression Results as Normal Distributions

Inspired by the way (Edwards, Jacobs, and Forrest 2016) displayed regression coefficients as normal distributions, `dotwhisker` now provides an easy way to make similar plots. To create such plots, call `dwplot` as always but include the argument `style = "distribution"`, then customize with other `dotwhisker` functions and `ggplot` additions as usual:

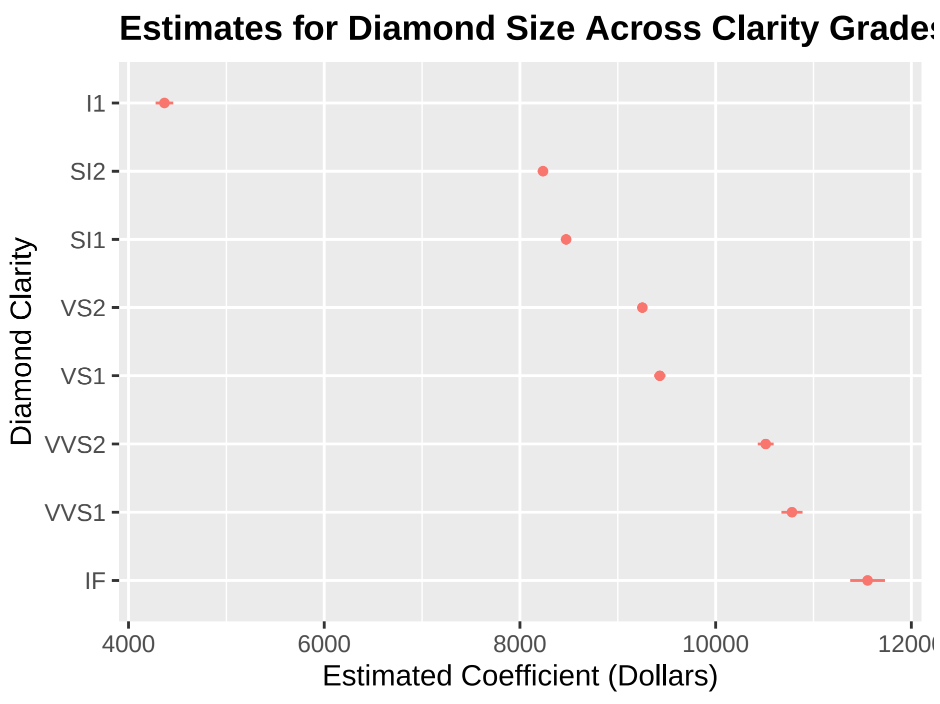
```
by_transmission_brackets <- list(
    c("Overall", "Weight", "Weight"),
    c("Engine", "Cylinders", "Horsepower"),
    c("Transmission", "Gears", "1/4 Mile/t")
)


{
    mtcars %>%
        split(.$am) %>%
        purrr::map( ~ lm(mpg ~ wt + cyl + gear + qsec, data = .x)) %>%
        dwplot(style = "distribution") %>%
        relabel_predictors(
            wt = "Weight",
            cyl = "Cylinders",
            disp = "Displacement",
            hp = "Horsepower",
            gear = "Gears",
            qsec = "1/4 Mile/t"
        ) +
        theme_bw(base_size = 4) + xlab("Coefficient") + ylab("") +
        geom_vline(xintercept = 0,
                   colour = "grey60",
                   linetype = 2) +
        theme(
            legend.position = c(.995, .99),
            legend.justification = c(1, 1),
            legend.background = element_rect(colour = "grey80"),
            legend.title.align = .5
        ) +
        scale_colour_grey(
            start = .8,
            end = .4,
            name = "Transmission",
            breaks = c("Model 0", "Model 1"),
            labels = c("Automatic", "Manual")
        ) +
        scale_fill_grey(
            start = .8,
            end = .4,
            name = "Transmission",
            breaks = c("Model 0", "Model 1"),
            labels = c("Automatic", "Manual")
        ) +
        ggtitle("Predicting Gas Mileage by Transmission Type") +
    theme(plot.title = element_text(face = "bold", hjust = 0.5))
} %>%
    add_brackets(by_transmission_brackets, fontSize = 0.3)
```



3.4. The 'Secret Weapon' and 'Small Multiple' Plots {#sec-the-‘secret-weapon’-and-‘small-multiple’-plots}

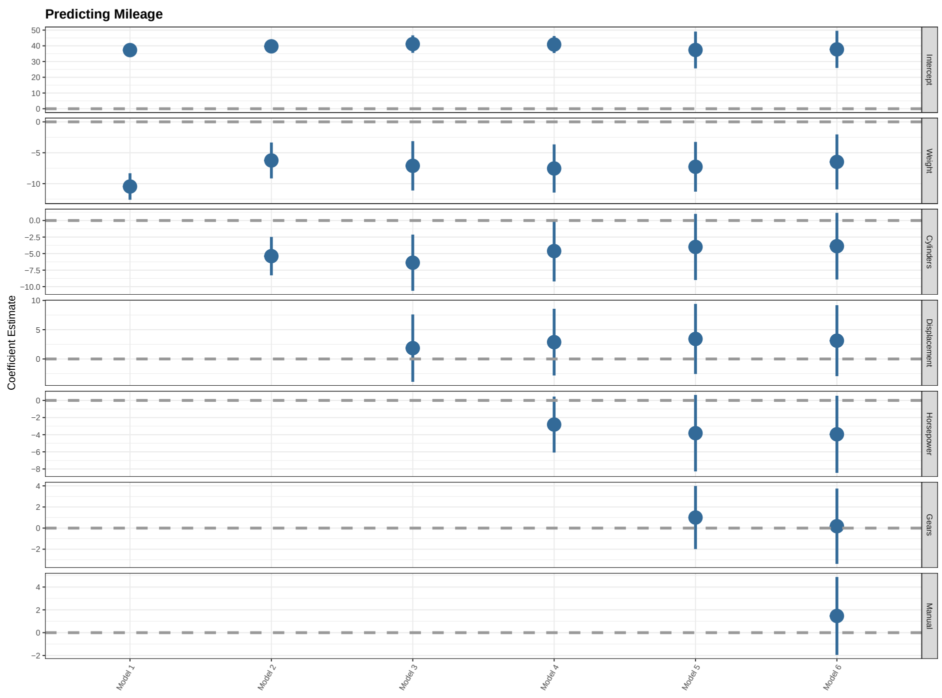
A variation of dot-and-whisker plot is used to compare the estimated coefficients for a single predictor across many models or datasets: Andrew Gelman calls such plots the 'secret weapon'. They are easy to make with the `secret_weapon` function. Like `dwplot`, the function accepts both lists of model objects and tidy data frames as input. The `var` argument is used to specify the predictor for which results are to be plotted.

```
data(diamonds)


# Estimate models for many subsets of data, put results in a tidy data.frame
by_clarity <- diamonds %>%
    group_by(clarity) %>%
    do(broom::tidy(lm(price ~ carat + cut + color, data = .), conf.int = .99)) %>%
    ungroup %>% rename(model = clarity)


# Deploy the secret weapon
secret_weapon(by_clarity, var = "carat") +
    xlab("Estimated Coefficient (Dollars)") + ylab("Diamond Clarity") +
    ggtitle("Estimates for Diamond Size Across Clarity Grades") +
    theme(plot.title = element_text(face = "bold"))
```



A final means of presenting many models' results at once in a particularly compact format is the ”small multiple” plot of regression results (Kastellec and Leoni 2007, 766). Small-multiple plots present estimates in multiple panels, one for each variable: they are similar to a stack of secret weapon plots. The `small_multiple` function makes generating these plots simple. Here, we pass a tidy data frame of six models to the function so we can to rescale the coefficients first, but the function can accept a list of model objects as well.

```
# Generate a tidy data frame of regression results from six models
m <- list()
ordered_vars <- c("wt", "cyl", "disp", "hp", "gear", "am")
m[[1]] <- lm(mpg ~ wt, data = mtcars)
m123456_df <- m[[1]] %>%
    broom::tidy() %>%
    by_2sd(mtcars) %>%
    mutate(model = "Model 1")
for (i in 2:6) {
    m[[i]] <- update(m[[i - 1]], paste(". ~ . +", ordered_vars[i]))
    m123456_df <- rbind(m123456_df,
                        m[[i]] %>%
                            broom::tidy() %>%
                            by_2sd(mtcars) %>%
                            mutate(model = paste("Model", i)))
}


# Relabel predictors (they will appear as facet labels)
m123456_df <- m123456_df %>%
    relabel_predictors(
        c(
            "(Intercept)" = "Intercept",
            wt = "Weight",
            cyl = "Cylinders",
            disp = "Displacement",
            hp = "Horsepower",
            gear = "Gears",
            am = "Manual"
        )
    )


# Generate a 'small multiple' plot
small_multiple(m123456_df) +
    theme_bw(base_size = 4) + ylab("Coefficient Estimate") +
    geom_hline(yintercept = 0,
               colour = "grey60",
               linetype = 2) +
    ggtitle("Predicting Mileage") +
    theme(
        plot.title = element_text(face = "bold"),
        legend.position = "none",
        axis.text.x = element_text(angle = 60, hjust = 1)
    )
```

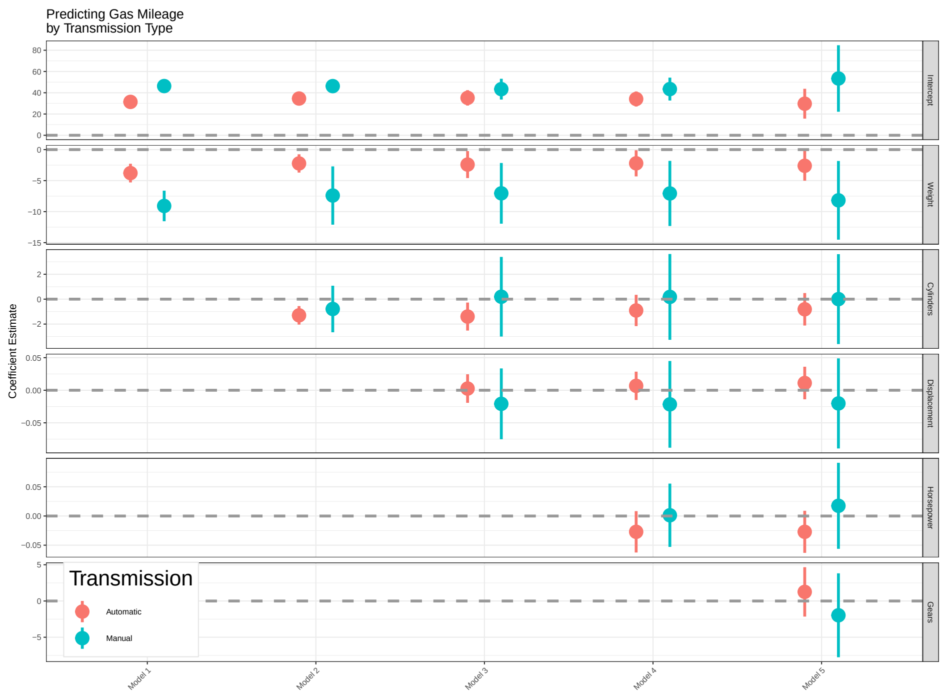


To facilitate comparisons across, e.g., results generated across different samples, one can cluster the results presented in a small multiple plot. To do so, results that should be clustered should have the same value of `model`, but should be assigned different values of an additional `submodel` variable included in the tidy data frame passed to `small_multiple`.

```
# Generate a tidy data frame of regression results from five models on
# the mtcars data subset by transmission type

ordered_vars <- c("wt", "cyl", "disp", "hp", "gear")
mod <- "mpg ~ wt"


by_trans2 <- mtcars %>%
    group_by(am) %>%                          # group data by transmission
    do(broom::tidy(lm(mod, data = .))) %>%           # run model on each group
    rename(submodel = am) %>%                 # make submodel variable
    mutate(model = "Model 1") %>%             # make model variable
    ungroup()


for (i in 2:5) {
    mod <- paste(mod, "+", ordered_vars[i])
    by_trans2 <- rbind(
        by_trans2,
        mtcars %>%
            group_by(am) %>%
            do(broom::tidy(lm(mod, data = .))) %>%
            rename(submodel = am) %>%
            mutate(model = paste("Model", i)) %>%
            ungroup()
    )
}


# Relabel predictors (they will appear as facet labels)
by_trans2 <- by_trans2 %>%
    select(-submodel, everything(), submodel) %>%
    relabel_predictors(
        c(
            "(Intercept)" = "Intercept",
            wt = "Weight",
            cyl = "Cylinders",
            disp = "Displacement",
            hp = "Horsepower",
            gear = "Gears"
        )
    )


by_trans2
```

```
# A tibble: 40 x 7
   term      estimate std.error statistic  p.value model   submodel
   <fct>        <dbl>     <dbl>     <dbl>    <dbl> <chr>      <dbl>
 1 Intercept   31.4       2.95     10.7   6.01e- 9 Model 1        0
 2 Intercept   46.3       3.12     14.8   1.28e- 8 Model 1        1
 3 Weight      -3.79      0.767    -4.94  1.25e- 4 Model 1        0
 4 Weight      -9.08      1.26     -7.23  1.69e- 5 Model 1        1
 5 Intercept   34.6       2.48     13.9   2.31e-10 Model 2        0
 6 Intercept   46.2       3.17     14.6   4.51e- 8 Model 2        1
 7 Weight      -2.23      0.752    -2.96  9.19e- 3 Model 2        0
 8 Weight      -7.40      2.40     -3.09  1.15e- 2 Model 2        1
 9 Cylinders   -1.30      0.379    -3.43  3.43e- 3 Model 2        0
10 Cylinders   -0.789     0.953    -0.828 4.27e- 1 Model 2        1
# i 30 more rows
```

```
small_multiple(by_trans2) +

    theme_bw(base_size = 4) +

    ylab("Coefficient Estimate") +

    geom_hline(yintercept = 0,
               colour = "grey60",
              linetype = 2) +

    theme(
        axis.text.x  = element_text(angle = 45, hjust = 1),
        legend.position = c(0.02, 0.008),
        legend.justification = c(0, 0),
        legend.title = element_text(size = 8),
        legend.background = element_rect(color = "gray90"),
        legend.spacing = unit(-4, "pt"),
        legend.key.size = unit(10, "pt")
    ) +

    scale_colour_hue(
        name = "Transmission",
        breaks = c(0, 1),
        labels = c("Automatic", "Manual")
    ) +

    ggtitle("Predicting Gas Mileage\nby Transmission Type")
```



4. Remaking the Examples from Kastellec and Leoni (2007)

Since (Kastellec and Leoni 2007) came out, and by putting their code for creating graphs from actually-published tables online, the authors allowed me and many others to adapt it for our own needs. But there can be no doubt that the code is cumbersome and that it takes a lot of fiddly work to get it to produce the desired results, work that likely contributes to the continued reliance in political science on tables to present regression results. As they acknowledged then, ”it simply takes more work to produce graphs” than tables (Kastellec and Leoni 2007, 757). Just eight-plus years later, the `dotwhisker` package, building on many other developments in the `R` ecosystem, arrived to make producing effective plots of regression results nearly effortless.

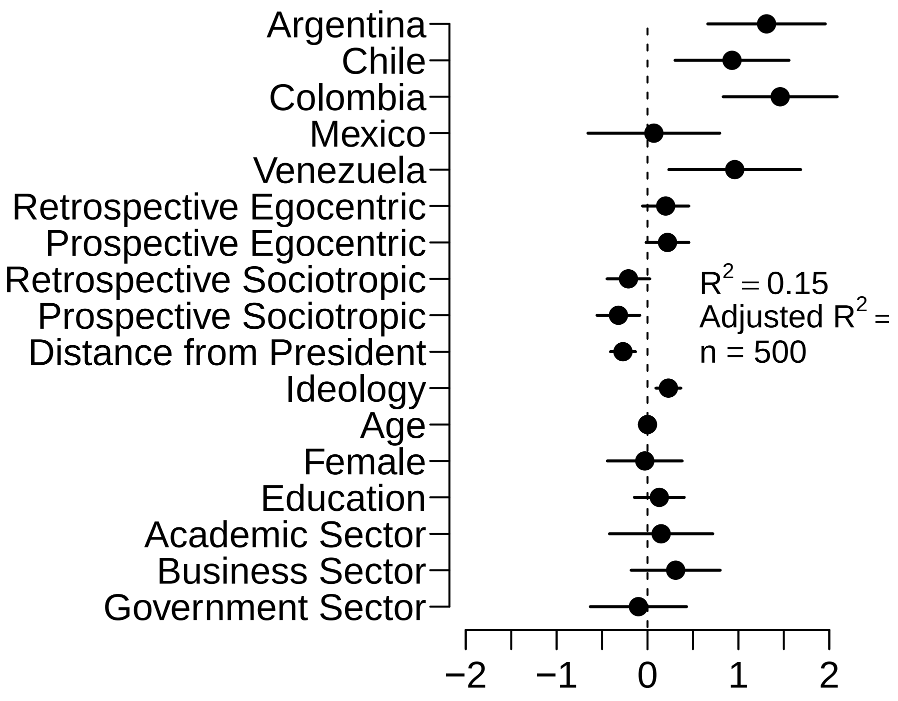
I present original code for their three examples of plotting regression results , along with similar plots done using `dotwhisker`.

4.1. Presenting a Single Regression Model Using a Dot Plot with Error Bars.

Kastellec and Leoni's original code:

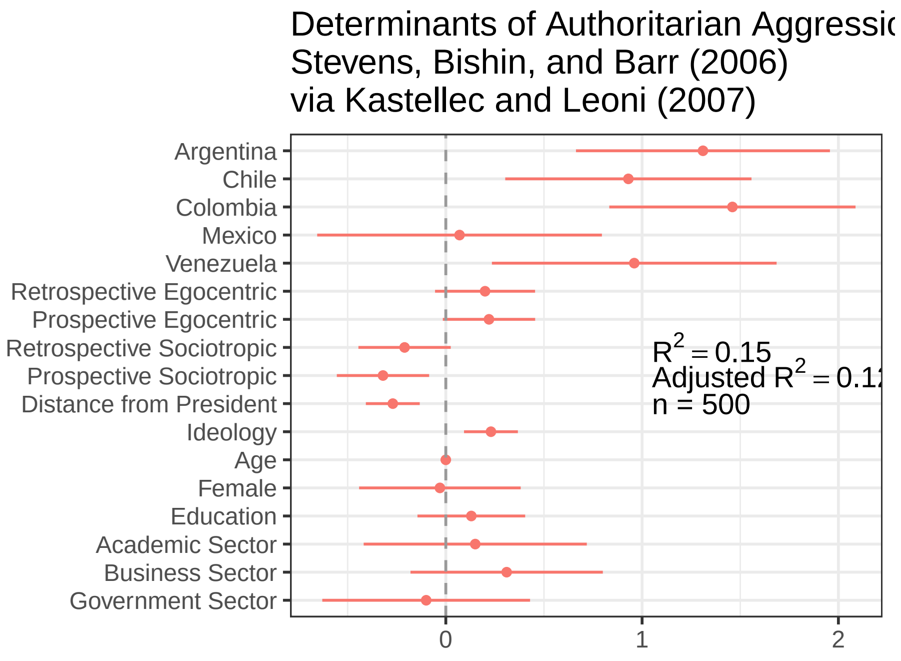
```
#Create vectors for coefficients, standard errors and variable names
    #we place coefficient as last element in each vector rather than 1st
    #since it is least important predictor, and thus we place it at the bottom of the grap
#note: we exclude the constant, since it is substantively meaningless


coef.vec <- c( 1.31, .93, 1.46, .07, .96, .2, .22, -.21, -.32, -.27,.23,
    0, -.03, .13, .15, .31, -.10)
se.vec <- c( .33, .32, .32, .37, .37, .13, .12, .12, .12, .07, .07, .01, .21,
    .14, .29, .25, .27)


var.names <- c("Argentina", "Chile", "Colombia", "Mexico", "Venezuela", #for longer names,
    "Retrospective Egocentric", "Prospective Egocentric",
    "Retrospective Sociotropic", "Prospective Sociotropic",
    "Distance from President", "Ideology", "Age", "Female", "Education",
    "Academic Sector", "Business Sector", "Government Sector")


y.axis <- c(length(coef.vec):1)#create indicator for y.axis, descending so that R orders v


par(mar=c(2, 13, 0, 0))#set margins for plot, leaving lots of room on left-margin (2nd num
plot(coef.vec, y.axis, type = "p", axes = F, xlab = "", ylab = "", pch = 19, cex = 1.2,#pl
    xlim = c(-2,2.5), xaxs = "r", main = "") #set limits of x-axis so that they include mi
        #coefficients + .95% confidence intervals and plot is symmetric; use "internal axe
#the 3 lines below create horiztonal lines for 95% confidence intervals, and vertical tick
segments(coef.vec-qnorm(.975)*se.vec, y.axis, coef.vec+qnorm(.975)*se.vec, y.axis, lwd =


axis(1, at = seq(-2,2,by=.5), labels = NA, tick = T,#draw x-axis and labels with tick mark
    cex.axis = 1.2, mgp = c(2,.7,0))#reduce label size, moves labels closer to tick marks
```

```
axis(1, at = seq(-2,2,by=1), labels =  c(-2,  -1,  0, 1,2), tick = T,#draw x-axis and labe
    cex.axis = 1.2, mgp = c(2,.7,0))#reduce label size, moves labels closer to tick marks


axis(2, at = y.axis, label = var.names, las = 1, tick = T, mgp = c(2,.6,0),
    cex.axis = 1.2) #draw y-axis with tick marks, make labels perpendicular to axis and cl
segments(0,0,0,17,lty=2) # draw dotted line through 0
#box(bty = "l") #place box around plot


#use following code to place model info into plot region
x.height <- .57
text(x.height, 10, expression(R^{2} == .15), adj = 0, cex = 1) #add text for R-squared
text(x.height, 9, expression(paste("Adjusted ", R^{2} == ".12", "")), adj = 0, cex = 1)#ad
text(x.height, 8, "n = 500", adj = 0, cex = 1)#add text for sample size
```



Redone using `dotwhisker`:

```
#install.packages("dotwhisker") # uncomment to install from CRAN
library(dplyr)
library(dotwhisker)
library(dplyr)


# Format data as tidy dataframe
results_df <- data.frame(term=var.names, estimate=coef.vec,
                                                std.error=se.vec)


# Draw dot-and-whisker plot
results_df %>% dwplot + theme_bw() + theme(legend.position="none") +
    ggtitle("Determinants of Authoritarian Aggression\nStevens, Bishin, and Barr (2006)\nv
    geom_vline(xintercept = 0, colour = "grey60", linetype = 2) +
    annotate("text", x = 1.05, y = 10, size = 4, hjust = 0,
             label = "R^2 == .15", parse = TRUE) +
    annotate("text", x = 1.05, y = 9, size = 4, hjust = 0,
             label = "Adjusted~R^2 == .12", parse = TRUE) +
    annotate("text", x = 1.05, y = 8, size = 4, hjust = 0,
              label = "n = 500")
```

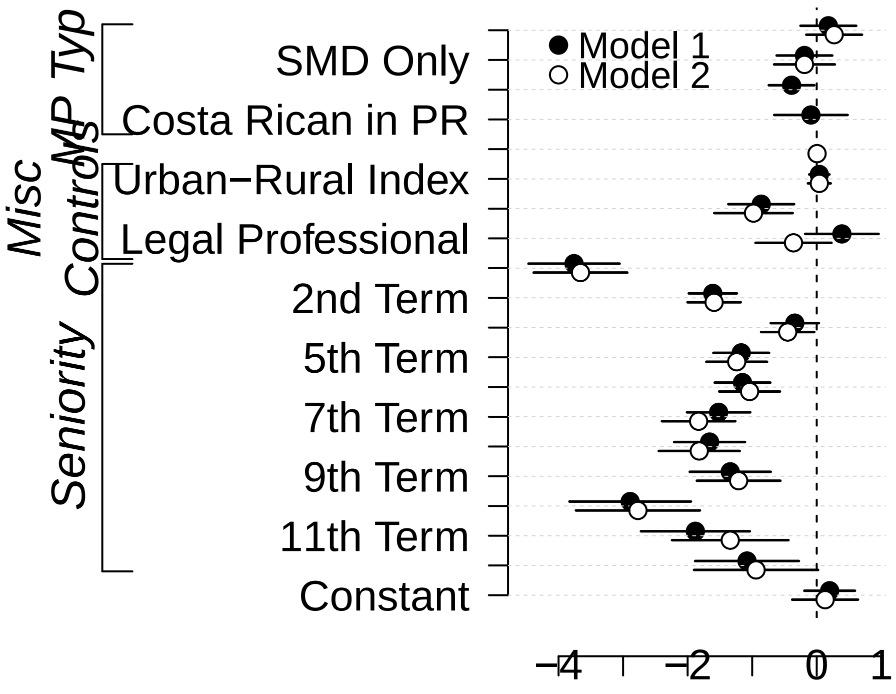


4.2. Using Parallel Dot Plots with Error Bars to Present Two Regression Models.

Kastellec and Leoni's original code:

```
#Create Vectors for coefs and standard errors for each model, and variable names
    #note that we  exclude "margin squared" since it doesn't appear in either model

coef.matrix <- matrix(c(-.039, NA, .048, -.133, .071, -.795, 1.47,
                        -.036, NA, .036, -.142, .07, -.834, 1.70,
                        -.051, NA, .017, .05, .011, -.532, .775,
                        -.037, -.02, .047, -.131,.072, -.783, 1.45,
                        -.034, -.018, -.035, -.139, .071, -.822, 1.68,
                        -.05, -.023, .016,-.049, .013, -.521, .819),nr=7)

## R2 of the models
R2<-  c(0.910,  0.910,  0.940,  0.910,  0.910,  0.940)


##standard error matrix, n.variables x n.models
se.matrix <- matrix(c(.003, NA, .011, .013, .028, .056, .152, .003, NA, .012, .014, .029,
                      .01, .013, .024, .044, .124, .003, .005, .011, .013, .028, .055, .15
                      .029, .059, .17, .003,.006, .01, .013, .024, .044, .127),nr=7)


##variable names
coef.vec.1<- c(0.18, -0.19,-0.39,-0.09, NA, 0.04,-0.86, 0.39,-3.76, -1.61,
    -0.34, -1.17, -1.15,-1.52, -1.66, -1.34,-2.89,-1.88,-1.08, 0.20)
se.vec.1 <-  c(0.22, 0.22, 0.18,.29, NA, 0.08,0.26,0.29,0.36,.19,0.19, 0.22,
    0.22,0.25,0.28,0.32,0.48, 0.43,0.41, 0.20)
coef.vec.2 <-  c(0.27,-0.19, NA, NA, 0.005, 0.04,-0.98,-.36,-3.66, -1.59,
     -0.45, -1.24, -1.04, -1.83, -1.82, -1.21, -2.77, -1.34, -0.94, 0.13)
se.vec.2 <- c(0.22,0.24, NA, NA, 0.004, 0.09 , .31 , .30 , .37 , .21 , .21 , .24 , .24,
     .29 , .32 , .33 , .49 , .46 , .49 , .26)
var.names <- c("Zombie" , "SMD Only", "PR Only", "Costa Rican in PR",
    "Vote Share Margin", "Urban-Rural Index","No Factional\nMembership",
    "Legal Professional", "1st Term", "2nd Term", "4th Term",
    "5th Term","6th Term","7th Term","8th Term","9th Term","10th Term",
    "11th Term","12th Term", "Constant")


y.axis <- length(var.names):1#create indicator for y.axis, descending so that R orders var
adjust <- .15 #create object that we will use to adjust points and lines up and down to di


layout(matrix(c(2,1),1,2), #in order to add variable categories and braces to left side of
    widths = c(1.5, 5))#we use layout command, create a small second panel on left side.

               #using c(2,1) in matrix command tells R to create right panel 1st
#layout.show(2) #can use this command to check results of layout command (but it must be c


par(mar=c(2,8,.5,1), lheight = .8)#set margins for regression plot
plot(coef.vec.1, y.axis+adjust, type = "p", axes = F, xlab = "", ylab = "", pch = 19, cex
    xlim = c(min((coef.vec.1-qnorm(.975)*se.vec.1 -.1), (coef.vec.2-qnorm(.975)*se.vec.2 -
    max((coef.vec.1+qnorm(.975)*se.vec.1 -.1), (coef.vec.2+qnorm(.975)*se.vec.2 -.1), na.r
    ylim = c(min(y.axis), max(y.axis)), main = "")
axis(1,at = seq(-4,1, by = 1), label = seq(-4,1, by = 1), mgp = c(2,.8,1), cex.axis = 1.3)
axis(2, at = y.axis, label = var.names, las = 1, tick = T, cex.axis =1.3)#add y-axis and l
```

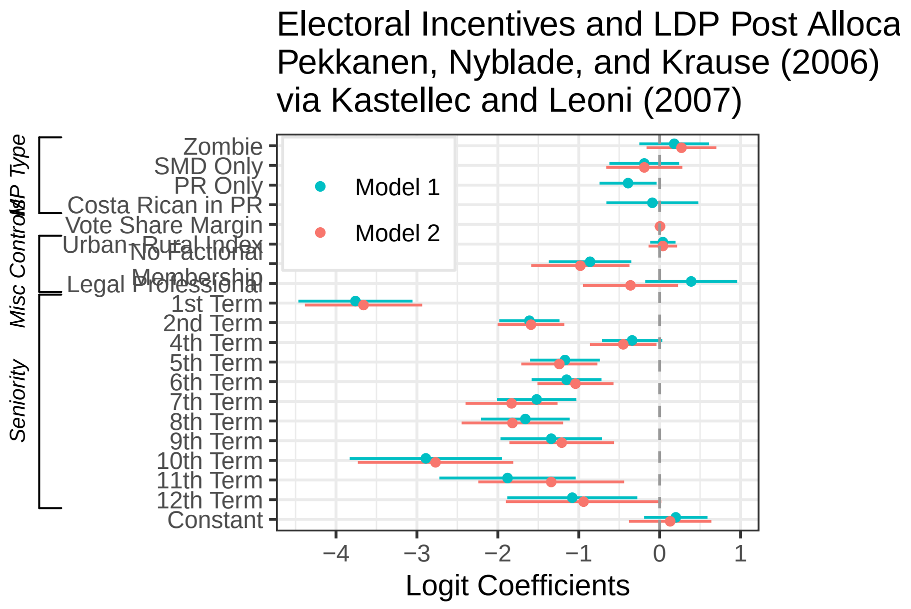
```
#axis(3,pretty(coef.vec.1, 3))#same as x-axis, but on top axis
abline(h = y.axis, lty = 2, lwd = .5, col = "light grey")#draw light dotted line at each v
#box(bty="l")#draw box around plot
segments(coef.vec.1-qnorm(.975)*se.vec.1, y.axis+adjust, coef.vec.1+qnorm(.975)*se.vec.1,


abline(v=0, lty = 2) # draw dotted line through 0 for reference line for null significance


#add 2nd model
    #because we are using white points and do want the lines to go "through" points rather
        #we draw lines first and the overlay points
segments(coef.vec.2-qnorm(.975)*se.vec.2, y.axis-adjust, coef.vec.2+qnorm(.975)*se.vec.2,
points(coef.vec.2, y.axis-adjust, pch = 21, cex = 1.2, bg = "white" ) #add point estimates


#add legend (manually) to identify which dots denote model 1 and which denote model 2
#legend(-4.5, 20, c("Model 1", "Model 2"), pch = c(19,21),bty = "n")
points(-4, 19.5, pch = 19, cex  = 1.2)
text(-3.7, 19.5, "Model 1", adj = 0,cex  = 1.2)#left-justify text using adj  = 0
points(-4, 18.5, pch = 21,cex  = 1.2)
text(-3.7, 18.5, "Model 2", adj = 0,cex  = 1.2)#left-justify text using adj  = 0


#Create Variable Categories and Braces to go in 2nd plot

par(mar=c(2,0,.5,0)) #set margins--- bottom (1st number) and top (3rd number) must be the
plot(seq(0,1,length=length(var.names)), y.axis, type = "n", axes = F,  xlab = "", ylab = "
    #use a sequence of length 20 so that x and y have same length

left.side <- .55#use this to manipulate how far segments are from y-axis
    #note:  getting braces and text in proper place requires much trial and error
segments(left.side,20.2,left.side,16.5) #add brackets around MP Type vars
segments(left.side,20.2,left.side+.15,20.2) #1 segment at a time
segments(left.side,16.5,left.side+.15,16.5)
text(.4, 18.5, "MP Type", srt = 90, font = 3, cex  = 1.5)#Add text; "srt" rotates to 90 de


#don't add "Electoral Strength" since it's only 1 variable

segments(left.side,15.5,left.side,12.3) #add brackets around "Misc Controls"
segments(left.side,15.5,left.side+.15,15.5) #one segment at a time
segments(left.side,12.3,left.side+.15,12.3)
text(.3, 14, "Misc\nControls", srt = 90, font = 3, cex  = 1.5)#Add text; "srt" rotates to


segments(left.side,12.15,left.side,1.8) #add brackets around "Seniority"
segments(left.side,12.15,left.side+.15,12.15)   #one segment at a time
segments(left.side,1.8,left.side+.15,1.8)
text(.4, 7, "Seniority", srt = 90, font = 3, cex  = 1.5)#Add text; "srt" rotates to 90 deg
```



Redone using `dotwhisker`:

```
# Format data as tidy dataframe
results_df <- data.frame(term = rep(var.names, times = 2),
                         estimate = c(coef.vec.1, coef.vec.2),
                         std.error = c(se.vec.1, se.vec.2),
                         model = c(rep("Model 1", 20), rep("Model 2", 20)),
                         stringsAsFactors = FALSE)


# Draw dot-and-whisker plot
p <- dwplot(results_df) +
    theme_bw() +
    theme(legend.justification=c(.02, .993), legend.position=c(.02, .99),
           legend.title = element_blank(), legend.background =
              element_rect(color="gray90")) +
    xlab("Logit Coefficients") +
    geom_vline(xintercept = 0, colour = "grey60", linetype = 2) +
    ggtitle("Electoral Incentives and LDP Post Allocation\nPekkanen, Nyblade, and Krause (


# Add brackets
```

```
p %>% add_brackets(list(c("MP Type", "Zombie", "Costa Rican in PR"),
                        c("Misc Controls", "Urban-Rural Index", "Legal Professional"),
                        c("Seniority", "1st Term", "12th Term")))
```



4.3. Using ”Small Multiple” Plots to Present Regression Results from Several Models. {#sec-using-“small-multiple”-plots-to-present-regression-results-from-several-models.}

Kastellec and Leoni's original code:

```
library(grid)
library(gridExtra)
```

```

Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```
##point estimates, in a n.variables, n.variables x n.models
coef.matrix <- matrix(c(-.039, NA, .048, -.133, .071, -.795, 1.47,
                        -.036, NA, .036, -.142, .07, -.834, 1.70,
                        -.051, NA, .017, .05, .011, -.532, .775,
                        -.037, -.02, .047, -.131,.072, -.783, 1.45,
                        -.034, -.018, -.035, -.139, .071, -.822, 1.68,
                        -.05, -.023, .016,-.049, .013, -.521, .819),nr=7)
```

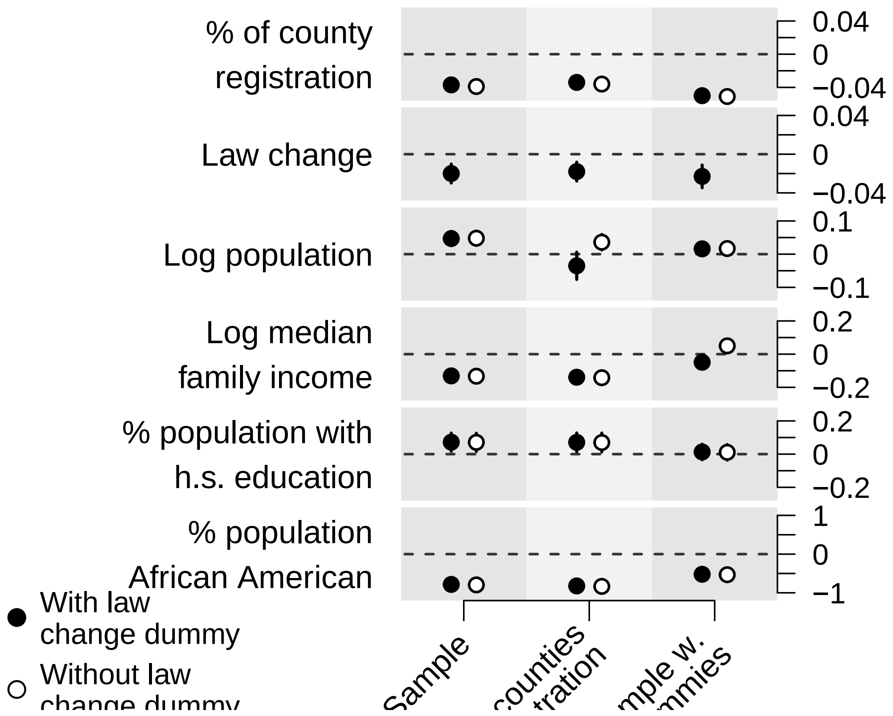
```
## R2 of the models
R2<-  c(0.910,  0.910,  0.940,  0.910,  0.910,  0.940)

##standard error matrix, n.variables x n.models
se.matrix <- matrix(c(.003, NA, .011, .013, .028, .056, .152, .003, NA, .012, .014, .029,
                      .01, .013, .024, .044, .124, .003, .005, .011, .013, .028, .055, .15
                      .029, .059, .17, .003,.006, .01, .013, .024, .044, .127),nr=7)

##variable names
varnames<- c("% of county\nregistration", "Law change", "Log population", "Log median\nfam
             "% population with\nh.s. education" ,"% population\nAfrican American" ,"Const


##exclude intercept
coef.matrix<-coef.matrix[-(7),]
se.matrix<-se.matrix[-7,]

## each panel has at most six models, plotted in pairs.
## in each pair, solid circles will be the models with "law change" in the specification
## empty circles, those without "law change"

##we are making a list, define it first as empty
Y1 <- vector(length=0,mode="list")
#estimates with law change (in the 4th to 6th columns)
Y1$estimate <- coef.matrix[,4:6]
##95% confidence intervals
Y1$lo <- coef.matrix[,4:6]-qnorm(0.975)*se.matrix[,4:6]
Y1$hi <- coef.matrix[,4:6]+qnorm(0.975)*se.matrix[,4:6]
##90% confidence intervals
Y1$lo1 <- coef.matrix[,4:6]-qnorm(0.95)*se.matrix[,4:6]
Y1$hi1 <- coef.matrix[,4:6]+qnorm(0.95)*se.matrix[,4:6]
##name the rows of Y1 estimate
rownames(Y1$estimate) <- varnames[-7] ##no intercept


#estimates without law change
Y2 <- vector(length=0,mode="list")
Y2$estimate <- coef.matrix[,1:3]
Y2$lo <- coef.matrix[,1:3]-qnorm(.975)*se.matrix[,1:3]
Y2$hi <- coef.matrix[,1:3]+qnorm(.975)*se.matrix[,1:3]
Y2$lo1 <- coef.matrix[,1:3]-qnorm(.95)*se.matrix[,1:3]
Y2$hi1 <- coef.matrix[,1:3]+qnorm(.95)*se.matrix[,1:3]
rownames(Y2$estimate) <- varnames[-7]

##code from http://svn.tables2graphs.com/tables2graphs/Rcode/plotReg.R
##The main function is plot.reg. It is called as follows
plot.reg <- function(Y,
                                        #a list composed by three or five matrices, all k
                                        #where k is the number of independent variables
                                        #and m is the number of models
                                        #the matrices are: estimate, lo and hi;
                                        #and optionally lo1 and hi1
                                        #lo and hi are the low and upper bounds of the
                                        #confidence intervals
                                        #similarly
                                        #lo1 and hi1 are the inner confidence intervals
                                        #which will be plotted as cross hairs
                     Y2=NULL,
                                        #specified just as Y
                                        #so one can plot the models in pairs (see examples
                     legend=NULL,
                                        #if there both Y and Y2 are specified, legend
                                        #is an optional character vector of length 2
                                        #giving the legends for Y and Y2
                     print=TRUE,       # print the plot or just create the object
                     refline=NA,        # a vector with the reference lines for each inde
                                        # put NA if you don't want ref lines
                     hlast=.1,         # the amount of space (in proportion) left at the
                                        # for the x-axis labels
                     lwd.fact=1,
                                        # a multiplier for the line width and character si
                     length.arrow=unit(0,"mm"),
                                        # length of the cross hair
                     widths=c(.45,.45,.1),
                                        # widths in proportion of the graph.
                                        # (space for the independent variable labels,
                                        # space for the panels,
                                        # space for the y-axis labels)
                     rot.label.y=0,    # rotation of the independent variable labels
                     just.label.y="right", # justification of the  independent variable la
                     pos.label.y=.97,  # x position of the independent variable labels
                     pch.size=0.5,     # size of the symbols
                     h.grid=FALSE,      # plot horizontal grid
                     v.grid=FALSE,      # plot vertical grid
                     expand.factor=0.25, # factor by which to extend the plot range
                     expand.factor2=0.1, # factor by which to extend the plot range
                     leg.mult=.7,  #rel size of legend
                     leg.fontsize=8, ## font size of legend
                     yaxis.at=NULL, ## list with y axis tick-mark points,
                     ylabel=NULL, ## list with y axis labels
                     ##with length equal to the number of plots
                     ...               # other options passed to the grid.Vdotplot functi
                     ) {
  ## the function gets the variable names from Y$estimate rownames
```

```
label.vec.vars <- rownames(Y$estimate)
## number of independent variables
n.plots <- nrow(Y$estimate)
if ((!is.null(yaxis.at))&(length(yaxis.at)!=n.plots)) {
  stop("length of yaxis.at must equal the number of plots")
}
hbet <- .01 # amound of vertical space between plots
hit <- (1-hlast-hbet*n.plots)/n.plots #height of each plot
sp.now <- 0 #i f sp.now > 0, the x-axis with labels is plotted
index <- seq(1,n.plots*2,2) # index of the plots and between spaces
grid.newpage() # create a new page
##a frame graphical object
## it has k*2 vertical slots (one for each variable + one for each space between plots)
fg <- frameGrob(layout=grid.layout(n.plots*2,
                  ## and 3 horizontal slots (space for ind. variables labels,
                  ## space for plots, space for yaxis labels)
                  3,widths=unit(widths,"npc"),
                  heights=unit(c(rep(c(hit,hbet),n.plots-1),hit,hlast),"npc")))
## loop to create panels
## j indexes independent variables
j <- 1
for (i in index) {  ## i is the vertical slot position
  #create a dataframe with the data to plot now
  Y.now <- data.frame(estimate=Y$estimate[j,],
                      lo=Y$lo[j,],
                      hi=Y$hi[j,],
                      lo1=Y$lo1[j,],
                      hi1=Y$hi1[j,])
  ##similartly for Y2
  if (!is.null(Y2)) Y2.now <- data.frame(estimate=Y2$estimate[j,],
                                         lo=Y2$lo[j,],
                                         hi=Y2$hi[j,],
                                         lo1=Y2$lo1[j,],
                                         hi1=Y2$hi1[j,])
  else Y2.now <- NULL
  ## if it is the bottom row, set sp.now to a positive value
  if (i==max(index)) sp.now <- .1
  ##are we drawing a reference line?
  drawRef <- !is.na(refline[j])
  ##place the plot
  ##the actual plot object is created by the function grid.Vdotplot
  fg <- placeGrob(fg,
                  grid.Vdotplot(Y.now,
                                Y2.now,
                                sp=c(.1,sp.now),draw=FALSE,lwd.fact=lwd.fact,
                                refline=ifelse(drawRef,refline[j],0) ## if refline is NA
                                ,drawRef=drawRef,
                                  length.arrow=length.arrow,
                                  pch.size=pch.size,
                                  h.grid=h.grid,
                                  v.grid=v.grid,
                                  expand.factor=expand.factor,
                                  expand.factor2=expand.factor2,
                                  aty=yaxis.at[[j]],
                                  labely=ylabel[[j]],
                                  ...)
                    ,col=2,row=i)
    ##the independent variables labels
    fg <- placeGrob(fg,textGrob(x=pos.label.y,label.vec.vars[j]
                                ,rot=rot.label.y,gp=gpar(cex=.75*lwd.fact),just=just.label
                                ),col=1,row=i)
    j <- j+1
  }
  ## if Y2 exists and a legend is specified, draw it using the legendGrob function
  if (!is.null(Y2)&!is.null(legend)) {
    fg <- placeGrob(fg,legendGrob(c(21,21),legend,cex=leg.mult,leg.fontsize=leg.fontsize,f
  }
  if (print) {
    grid.arrange(fg)
  } else {
    ## if we are not printing, return the graphical object
    fg
  }
}

### grid.Vdotplot is what actually draws the plots
### the arguments are explained in the plot.reg function
grid.Vdotplot <- function(Y,Y2=NULL,x=NULL,sp=c(.1,.1),draw=TRUE,refline=0,label.x=NULL,dr
  ## function to plot point estimates
  estimates.grob <- function(x, #x coordinates
                             Y, #Y$estimate has the y coordinates
                             fill="black" #color to fill the symbol
                             ) {
    ## pointsGrob is a grid function
    pointsGrob(x,Y$estimate,pch=21,size=unit(pch.size,"char"),gp=gpar(fill=fill,lwd=lwd.fa
  }
  ## function to plot confidence intervals
  ci.grob <- function(ylo,yhi,x,lwd=2.5,name="ci",plot.arrow=FALSE) {
    ##do we want the cross hairs at the ends?
    if (plot.arrow) {
      arrow.now <- arrow(angle=90,length=length.arrow,ends="both")
    } else {
      arrow.now <- NULL
    }
    ## use the segmentsGrob function of grid to plot the error bars
    segmentsGrob(x0=x,x1=x,y0=ylo,y1=yhi,
                 default.units="native",
                 name=name,gp=gpar(lwd=lwd)
                 ,arrow=arrow.now
                 )
  }

  if (is.null(aty)) {
    ##  tick-mark not supplied

    ## calculate y axis ticks (and labels)
    ## create a vector with all values in the plot
    aty <- unique(c(unlist(Y),unlist(Y2),refline))
    ## if there is a refline, we want to make the plot symmetric around it
    ##aty <- pretty(aty,5,min.n=5,high.u.bias=5)
    if (!is.na(refline)) {
      ##maximum distance
      ##mdist <- max(abs(aty-refline),na.rm=TRUE)
      aty <- pretty(aty,2,min.n=2,high.u.bias=1)
      ##cat(aty,"is aty a \n")
      aty <- unique(sort(c(aty,2*refline-aty)))
      ##cat(aty,"is aty b \n")
    }
    else {
      aty <- pretty(aty,5,min.n=5,high.u.bias=5)
    }
    ## take out the highest and the lowest value, to minimize whitespace
    if (length(aty)>5)  {
      aty <- aty[-c(1,length(aty))]
      r.y <- range.e(aty,expand.factor) ## expand the range, so as to include everything
    } else {
      r.y <- range.e(aty,expand.factor2)
    }
    ##cat(aty,"is aty c \n")
    ##aty <- c(aty,-max(abs(aty)),max(abs(aty)))
    ##r.y <- range(aty)
    ## make sure we draw the horizontal grid in every interval in the plot
    ## but not outside the plot area
    ## some manual adjustment might be necessary
    aty <- ifelse(aty<min(r.y)|aty>max(r.y),NA,aty)
  } else {
    r.y <- range.e(aty,expand.factor) ## expand the range, so as to include everything
  }
  ## x axis. might have to change this. the default is simply
  ## an index of the models in the x - axis
  ## later we possibly want to make this continuous
```

```
if (is.null(x)) x <- 1:nrow(Y)
##if (is.null(x)) x <- c(.5,1.75,3)
## save x values
x.o <- x
l.x <- length(x)
## if there is no label, we create one with the index
if (is.null(label.x)) label.x <- paste("(",x.o,")",sep="")
## if there is a second set of values create x2=x+e and decrease x to x-e
if (!is.null(Y2)) {
  x2 <- x+y1y2sep
  x <- x-y1y2sep
}
## horizontal grid
if (h.grid) {
  hgrid <-   segmentsGrob(x0=unit(rep(0,length(aty)),"npc"),
                          x1=unit(rep(1,length(aty)),"npc"),
                          y0=unit(aty,"native"),
                          y1=unit(aty,"native"),
                          gp=gpar(lty="dotted",lwd=lwd.fact,col="lightgrey"))
} else {
  hgrid <- NULL
}
## vertical grid
if (v.grid) {
  ##     vgrid <-   segmentsGrob(y0=unit(rep(0,l.x),"npc"),
  ##                             y1=unit(rep(1,l.x),"npc"),
  ##                             x0=unit(x.o,"native"),
  ##                             x1=unit(x.o,"native"),
  ##                             gp=gpar(lty="dotted",lwd=lwd.fact,col="lightgrey"))
  vgrid <-   rectGrob(y=unit(rep(0.5,l.x),"npc"),
                      ##x=unit(x.o[seq(2,l.x,1)],"native"),
                      x=unit(x.o,"native"),
                      width=unit(1,"native"),
                      gp=gpar(lty="dotted",lty=0,fill=c("gray90","gray95")))
} else {
  vgrid <- NULL
}
## ref line
if (drawRef)  {
  refline <- segmentsGrob(x0=unit(0.01,"npc"),x1=unit(.99,"npc"),y0=unit(refline,"native
} else {
  refline <- NULL
}
## store ci
ci1a <- NULL
ci1b <- NULL
ci2a <- NULL
```

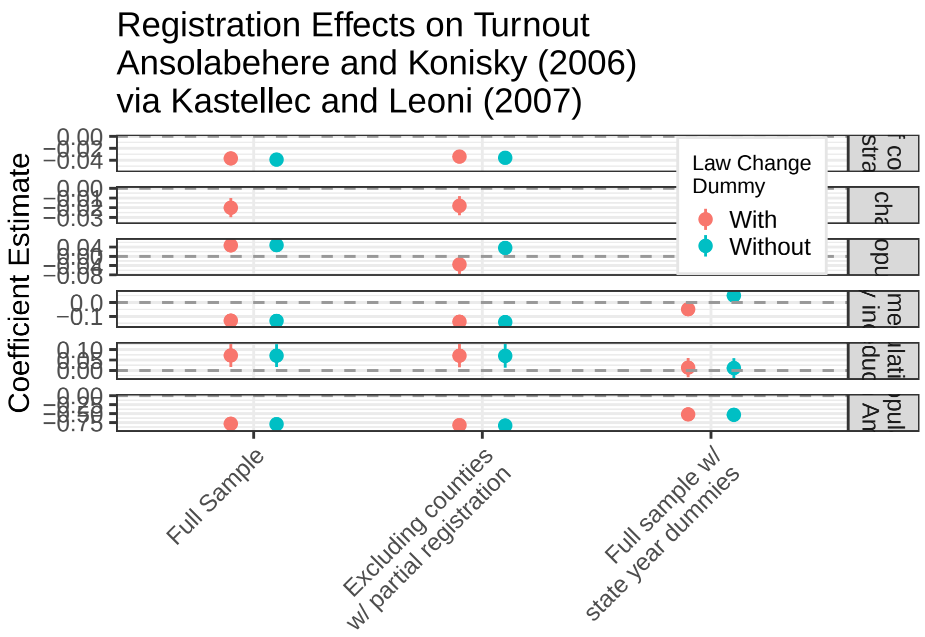
```
ci2b <- NULL
points2 <- NULL
## if ncol(Y)=5 there are overlapping CIs. the second one here.
if(ncol(Y)==5) ci1b <-        ci.grob(Y$hi1,Y$lo1,x,lwd=.8*lwd.fact,name="ci1b",plot.ar
## the first one here.
ci1a <- ci.grob(Y$hi,Y$lo,x,lwd=1.2*lwd.fact,name="ci1a")
if (!is.null(Y2)) {
  ## if ncol(Y2)=5 there are overlapping CIs. the second one here.
  if(ncol(Y2)==5) ci2b <- ci.grob(Y2$hi1,Y2$lo1,x2,lwd=.8*lwd.fact,name="ci2b",plot.arro
  ## the first one here.
  ci2a <- ci.grob(Y2$hi,Y2$lo,x2,lwd=1.2*lwd.fact,name="ci2a")
  ## point estimates here
  points2 <- estimates.grob(x2,Y2,fill="white")
}
if (is.null(labely)) {
  labely <- aty
  ##print(paste("labely is ",labely,is.null(labely)))
}
gplot <- with(Y,
              ## gTree is a graphical object with the whole plot
              gTree(
                    children=gList(
                      hgrid,vgrid,
                      refline,
                      ci1a,
                      ci1b,
                      estimates.grob(x,Y),
                      ## if Y2
                      ci2a,
                      ci2b
                      ,points2
                      ## box/rectangle around the plot area
                      ##,rectGrob(gp=gpar(lwd=.5*lwd.fact))
                      ## plot x axis if sp2>0 (we name it xaxis, so we can refer to it l
                      ,if(sp[2]!=0) xaxisGrob(at=x.o,label=label.x,name="xaxis",
                              gp=gpar(cex=0.8*lwd.fact,lwd=0.6*lwd.fact))
                      ## plot x axis with no labels if it is not the bottom plot
                      ,if((sp[2]==0)&(!v.grid)) xaxisGrob(at=x.o,label=rep("",length(x.o
                              gp=gpar(cex=0.8*lwd.fact,lwd=0.6*lwd.fact))
                      ## plot y-axis if sp1>0
                      ,if(sp[1]!=0) yaxisGrob(at=aty,label=labely,
                              gp=gpar(cex=0.7*lwd.fact,lwd=0.6*lwd.fact),main=FALSE,name
                    ## definition of the viewport (plot area)
                    vp=viewport(width=unit(1, "npc"),
                      height=unit(1, "npc"),
                      ##xscale=c(1,nrow(Y)),
                      xscale=c(1-.5,nrow(Y)+.5),
                       yscale=r.y
                       ##yscale=c(-.1,.1)
                       ,clip=FALSE)
                     ))
  if (draw==TRUE) {
    ##draw the plot
    grid.newpage()
    fg <- frameGrob(layout=grid.layout(2,2,widths=unit(c(sp[1],1-sp[1]),"npc"),heights=uni
    fg <- placeGrob(fg,gplot,col=2,row=1)
    grid.arrange(fg)
  } else {
    gplot
  }
}

##function to create legend (adapted from Murrell R Graphics book)
legendGrob <- function(pch, ## what symbol to use
                       labels, ## the text
                       hgap = unit(0.1, "lines"), #horizontal gap
                       vgap = unit(0.5, "lines"), #vertical gap
                       default.units = "lines", #default units
                       vp = NULL, #what viewport to use
                       cex=1, #character expansion
                       leg.fontsize=8,
                       fill=NULL)
{
  nkeys <- length(labels)
  gf <- frameGrob(vp = vp)
  for (i in 1:nkeys) {
    if (i == 1) {
      symbol.border <- unit.c(vgap, hgap,
                              vgap, hgap)
      text.border <- unit.c(vgap, unit(0,
                                       "npc"), vgap, hgap)
    }
    else {
      symbol.border <- unit.c(vgap, hgap,
                              unit(0, "npc"), hgap)
      text.border <- unit.c(vgap, unit(0,
                                       "npc"), unit(0, "npc"), hgap)
    }
    gf <- packGrob(gf, pointsGrob(0.5, 0.5,
                                  pch = pch[i],gp=gpar(cex=cex,fill=fill[i])), col = 1, ro
                   width = unit(1, "lines"), height = unit(1,
                                                           "lines"), force.width = TRUE)
 gf <- packGrob(gf, textGrob(labels[i],
                             x = 0, y = 0.5, just = c("left", "centre"),gp=gpar(fontsize=l
 col = 2, row = i, border = text.border)
 }
 gf
 }

##function to plot the legend
grid.legend <- function(pch, labels, frame = TRUE,
                         hgap = unit(1, "lines"), vgap = unit(1, "lines"),
                         default.units = "lines", draw = TRUE, vp = NULL) {
  gf <- legendGrob(pch, labels, frame, hgap,
                    vgap, default.units, vp)
   if (draw)
     grid.arrange(gf)
   gf
 }



range.e <- function(x,xp=.1) {
  ##expand the range by a fixed proportion
  r <- range(x,na.rm=TRUE)
  r.e <- (r[2]-r[1])*xp
  c(r[1]-r.e,r[2]+r.e)
}

## create the graph (do not print it yet)
tmp <- plot.reg(Y1,Y2,#the lists
                #the model labels
                label.x=c("Full Sample","Excluding counties\nw. partial registration",
                  "Full sample w. \nstate year dummies"),
                ## reference lines
                refline=c(0,0,0,0,0,0),
                ## space left in the bottom (for the x-axis labels)
                hlast=.15,
                ## print the graph?
                print=FALSE,
                ## line width / character size multiplier
                lwd.fact=1.3,
                ## length of the cross- hairs
                length.arrow=unit(0,"mm"),
                ## legend
                ##legend=c("without law change","with law change"),
                ## widths: variable names, plot size, y-axis
                widths=c(.6,.4,.3),
                ## rotation of the variable name labes
                rot.label.y=0,
                ## justification of the variable name labels
```

```
                    just.label.y="right",
                    ## position (x-axis) of the variable name labels)
                    pos.label.y=0.95,
                    ## size of the symbol
                    pch.size=0.6,expand.factor=.2,expand.factor2=0.1,
                    ##legend
                    legend=c("With law\nchange dummy","Without law\nchange dummy"),leg.mult=.7
                    ##legend font size
                    leg.fontsize=11,
                    v.grid=TRUE,
                    yaxis.at=list(
                    NULL,
                    NULL,
                    seq(-.1,.1,.05),
                    seq(-.2,.2,.1),
                    seq(-.2,.2,.1),
                    NULL##seq(-1,1,.5)
                    )
                    )
## we rotate the labels of the x-axis 45 degrees. The grid utilities allow
## this modification "on the fly", and it is easy if you are careful at naming the paths
tmp <- editGrob(tmp,gPath("xaxis","labels"),rot=45,just="right",gp=gpar(lineheight=.75))
##tmp is the object we have just created,"xaxis" is the name of element in the object with
##elements, and "labels" is the actual object in xaxis that we want to rotate
##just is the justification of the text
grid.arrange(tmp) ## print the graph
```



Redone using `dotwhisker`:

```
# Format data as tidy dataframe
model_names <- c("Full Sample\n",
                       "Excluding counties\nw/ partial registration\n",
                       "Full sample w/\nstate year dummies\n")

submodel_names <- c("With","Without")


model_order <- c(4, 1, 5, 2, 6, 3)


results_df <- data.frame(term = rep(varnames[1:6], times = 6),
                         estimate = as.vector(coef.matrix[, model_order]),
                         std.error = as.vector(se.matrix[, model_order]),
                         model = as.factor(rep(model_names, each = 12)),
                         submodel = rep(rep(submodel_names, each = 6), times = 3),
                         stringsAsFactors = FALSE)



small_multiple(results_df) +
    scale_x_discrete(limits = model_names) + # order the models
    theme_bw() + ylab("Coefficient Estimate") +
    geom_hline(yintercept = 0, colour = "grey60", linetype = 2) +
    theme(axis.text.x  = element_text(angle = 45, hjust = 1),
          legend.position=c(.97, .99), legend.justification=c(1, 1),
          legend.title = element_text(size=8),
          legend.background = element_rect(color="gray90"),
          legend.spacing = unit(-3, "pt"),
          legend.key.size = unit(10, "pt")) +
    scale_colour_hue(name = "Law Change\nDummy") +
    ggtitle("Registration Effects on Turnout\nAnsolabehere and Konisky (2006)\nvia Kastell
```



5. Conclusion

The `dotwhisker` package provides a flexible and convenient way to visualize regression results and to compare them across models. This article offers an overview of its use and features. We encourage users to consult the help files for more details.

References

Edwards B, Jacobs J, Forrest S (2016). "Risky Business: Assessing Security with External Measurements." Pre-print.

Gelman A (2008). "Scaling Regression Inputs by Dividing by Two Standard Deviations." *Statistics in medicine*, **27**(15), 2865–2873.

Gelman A, Pasarica C, Dodhia R (2002). "Let's Practice What We Preach: Turning Tables into Graphs." *American Statistician*, **56**(2), 121–130.

Kastellec JP, Leoni EL (2007). "Using Graphs Instead of Tables in Political Science." *Perspectives on Politics*, **5**(4), 755–771.

Robinson D (2015). *broom: Convert Statistical Analysis Objects into Tidy Data Frames*. R package version 0.3.7, URL https://CRAN.R-project.org/package=broom.

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer, New York. ISBN 978-0-387-98140-6. URL https://ggplot2-book.org/.

Affiliation:

Frederick Solt
E-mail: `frederick-solt@uiowa.edu`

Yue Hu
E-mail: `yuehu@tsinghua.edu.cn`

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
`doi:10.18637/jss.v000.i00` *Accepted:* yyyy-mm-dd