SPEC REU R Resources: Visualizing Regression Results with dot-and-whisker Plots

Yeiyoung Choo and Claudia Salas Gimenez

Summer 2024

In this module, we will learn how to create dot-and-whisker plots, which is one of the common ways to visualize results from regression analysis.

Initial Setup

Set your R environment to the location of the data files, and load the necessary libraries and dataset. The data comes from a study by Appel & Loyle (2012) on post-conflict justice institutions (truth commissions in particular) and foreign direct investment.

Please note that for part of this module, we will use the `dotwhisker` package. As of April 2024, `dotwhisker` has been removed from CRAN, so you need to install its dependencies, `prediction` and `margins`, from GitHub to load it.

```
# Set working directory
#setwd("YourFolderPath")

# Load required libraries
library(tidyverse)
library(ggplot2)
library(broom)

# As of April 2024, "dotwhisker" is no longer on CRAN and must be installed with its
# dependencies "prediction" and "margins" from GitHub:
# remotes::install_github("leeper/prediction", force = TRUE)
# remotes::install_github("leeper/margins", force = TRUE)
# remotes::install_github("fsolt/dotwhisker", force = TRUE)

library(dotwhisker)

# Load the data
al2012 <- readRDS("al2012.rds")
```

Approach 1: Utilizing Regression Output as Input for ggplot2 Visualization

In this section, we'll use regression model outputs to generate dot-and-whisker plots using the `ggplot` package to illustrate the impact and uncertainties of the predictors.

Create a regression object

Recall that the basic syntax for linear regression is `lm(y ~ x, data = data)`. We use `+` to add predictors, such as: `lm(y ~ x1 + x2 + x3, data = data)`. Store each model as a regression object, e.g., `m1 <- lm(y ~ x, data = data)`.

Let's run our first regression model and store it as an object `m1`. The outcome variable is `fdiflow`, capturing inflows of foreign direct investment. The variable `truthvictim` is our key predictor, which is implementation of a post-conflict justice institution. Include variables `damage`, `peaceagr` and `victory` as well. Check regression results using `summary()` and save them as `summary_m1`.

```
# Fit a linear regression model
## DV: foreign direct investment inflows
m1 <-lm(fdiflow ~ truthvictim + damage + duration + peaceagr + victory,
        data = al2012)

# Display regression results and save them as `summary_m1`
summary_m1 <- summary(m1)
```

Calculate confidence intervals

Next, we'll calculate the confidence intervals using the regression results saved in `summary_m1`. Confidence intervals estimate the uncertainty around a measurement, typically within a 95% confidence level. The formula for a 95% confidence interval is:

`Lower Confidence Interval <= Estimate <= Upper Confidence Interval`

`Upper CI = Estimate + (1.96 × Standard Error)`

`Lower CI = Estimate - (1.96 × Standard Error)`

Where:

- **Estimate** is the coefficient value from the regression output.
- **Standard Error** measures the accuracy of the estimate.
- **1.96** standard value used to capture the middle 95% of the normal distribution.

By applying this formula, we can determine the range within which the true parameter lies, assuming a normal distribution of errors.

First, we'll extract the names of regression coefficients (except the intercept) as `term`, the coefficient estimates as `estimate`, and the standard errors as `std.error` from `summary_m1`. Then, we'll calculate the lower and upper bounds of the 95% confidence interval for each coefficient. Let's store them in a dataframe called `results_df`.

```
# Create a dataframe for plotting based on the results stored in `summary_m1`
results_df <- data.frame(
  ## Extract the names of regression coefficients except for the intercept.
  ## [-1] skips the first element, which is the intercept.
  term = rownames(summary_m1$coefficients)[-1],
  ## Extract the coefficient estimates for each predictor, excluding the intercept.
  ## [-1, "Estimate"] selects all rows except the intercept and the 'Estimate' column.
  estimate = summary_m1$coefficients[-1, "Estimate"],
  ## Extract the standard errors for the coefficient estimates, excluding the intercept.
  ## [-1, "Std. Error"] selects all rows except the intercept and the 'Std. Error' column.
  std.error = summary_m1$coefficients[-1, "Std. Error"],
  ## Calculate the lower bound of the 95% confidence interval for each coefficient.
  conf.low = summary_m1$coefficients[-1, "Estimate"] -
```

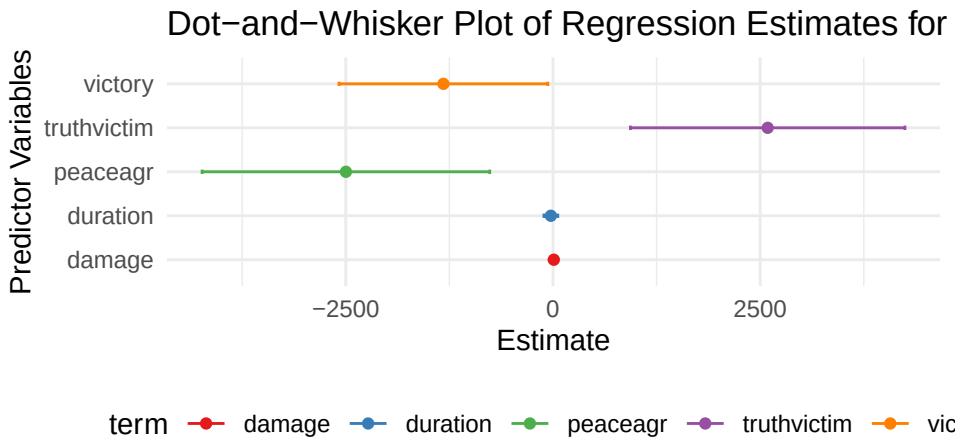
```
    1.96 * summary_m1$coefficients[-1, "Std. Error"],
  ## Calculate the upper bound of the 95% confidence interval for each coefficient.
  conf.high = summary_m1$coefficients[-1, "Estimate"] +
    1.96 * summary_m1$coefficients[-1, "Std. Error"])
```

Generate a dot-and-whisker plot

Finally, we'll create the dot-and-whisker plots using `ggplot2`. At the bare minimum, we need the key variables (`term`), their coefficient estimates (`Estimate`), and their uncertainty, included with the confidence intervals (`conf.low` and `conf.high`).

For an overview of the most important functions and geoms available through **`ggplot2`**, a handy ggplot2 cheatsheet is available.

```
# Create the dot-and-whisker plot using ggplot2
dw_plot <- ggplot(results_df, aes(x = estimate, y = term, color = term)) +
  geom_point() +
  geom_errorbar(aes(xmin = conf.low, xmax = conf.high), width = 0.1) +
  ## Sets the horizontal span of the error bars based on the confidence intervals calculated,
  ## showing the estimate's uncertainty.
  labs(title = "Dot-and-Whisker Plot of Regression Estimates for FDI inflows",
       x = "Estimate", y = "Predictor Variables") +
  theme_minimal() +
  theme(legend.position = "bottom") +
  scale_color_brewer(palette = "Set1")
  ## Apply the 'Set1' color palette from 'ColorBrewer' to differentiate the terms.

# Print the plot
print(dw_plot)
```



Approach 2: *dotwhisker* Visualization

An alternative to using `ggplot2` is using the `dotwhisker` package, introduced by Frederick Solt and Yue Hu in September 2021. The package provides a quick and easy way to generate the coefficient plots we want; and pairs well with `dplyr` as it builds on `ggplot2` architecture.

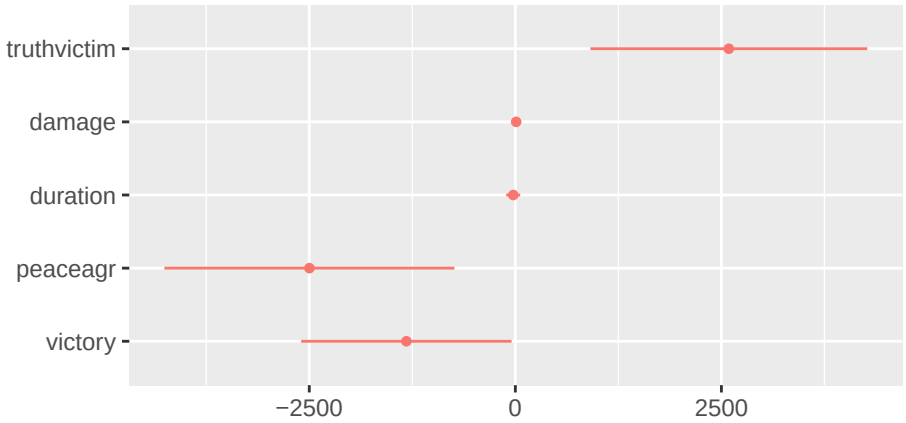
With `dotwhisker`, we can either use results directly from regression objects or tidied results (`broom::tidy()`) stored in dataframes.

Approach 2.1: Using Results from Regression Objects as Input for `dotwhisker` Visualization

Similar to Approach 1, we'll use the regression model `m1` to create a dot-and-whisker plot. Use the `dwplot()` function from the dotwhisker package, specifying the regression object in the parenthesis. The coefficient estimates on predictor variables will be shown as dots, with the whiskers representing the 95% confidence intervals, automatically calculated by `dwplot()`.

Note: When using `dwplot()` with regression objects, the intercept is excluded by default. Since the plot is used to visualize coefficient estimates for predictor variables, the intercept isn't necessary. To include it, use `dwplot(m1, show_intercept = TRUE)`. Also, `dwplot()` shows 95% confidence interval by default. We rarely need to adjust this setting, but you can specify if you want to show 90% confidence interval for example: `dwplot(m2, ci=.90)`.

```
# Create the dot-and-whisker plot using dotwhisker
dwplot(m1)
```



The coefficients for damage and duration are hard to interpret due to their relative size. To make coefficients more comparable, consider scaling these variables.

Generate a dot-and-whisker plot for `m2`

Now, let's repeat this process to plot the results of different regressions.

First, run a second regression model and store it as an object, calling it `m2`. This second model will keep `truthvictim` as key predictor variable, but will use political variables — political constraints (`polcon`) and the polity score (`polity2`) — instead of conflict variables as controls.

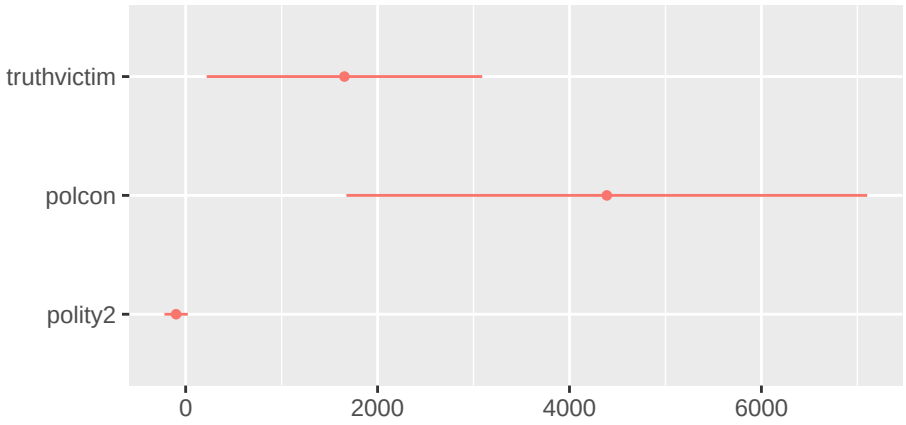
```
# Fit a linear regression model
## DV: foreign direct investment inflows
m2 <-lm(fdiflow ~ truthvictim + polcon + polity2, data = al2012)

# Display regression results
summary(m2)
```

```
##
## Call:
## lm(formula = fdiflow ~ truthvictim + polcon + polity2, data = al2012)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3578.5 -1015.7  -144.8   209.3 21085.1
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -632.59     459.17  -1.378  0.17168
## truthvictim  1654.68     722.89   2.289  0.02439 *
## polcon       4389.13    1366.29   3.212  0.00182 **
## polity2       -99.64      61.24  -1.627  0.10718
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2719 on 91 degrees of freedom
## Multiple R-squared:  0.1555, Adjusted R-squared:  0.1277
## F-statistic: 5.586 on 3 and 91 DF,  p-value: 0.00146
```

Then, use the object `m2` to generate a dot-and-whisker plot, showing coefficient estimates for the predictors in Model 2.

```
# Create the dot-and-whisker plot using dotwhisker
dwplot(m2)
```



The estimate for `truthvictim` in Model 2 differs from Model 1, but comparing separate plots is challenging.

Create dot-and-whisker plot for multiple regression models

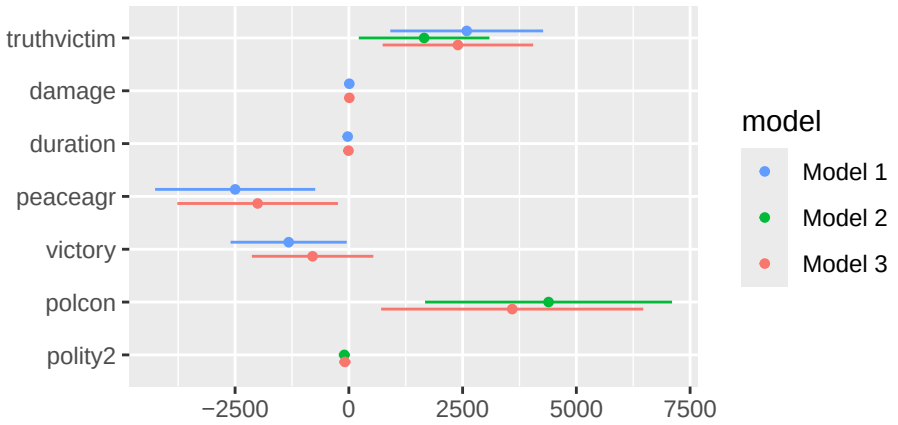
To compare the estimates for `truthvictim` in Models 1 and 2, we'll visualize results from multiple regression models in a single plot. This makes it easier to compare coefficient estimates across models.

We'll employ the `m1` and `m2` models from before, and introduce a third regression model, `m3`, that includes `truthvictim` as key predictor variable and both political and conflict variables as controls.

```
# Fit a linear regression model
## DV: foreign direct investment inflows
m3 <-lm(fdiflow ~ truthvictim + damage + duration + peaceagr + victory + polcon + polity2,
        data = al2012)
```

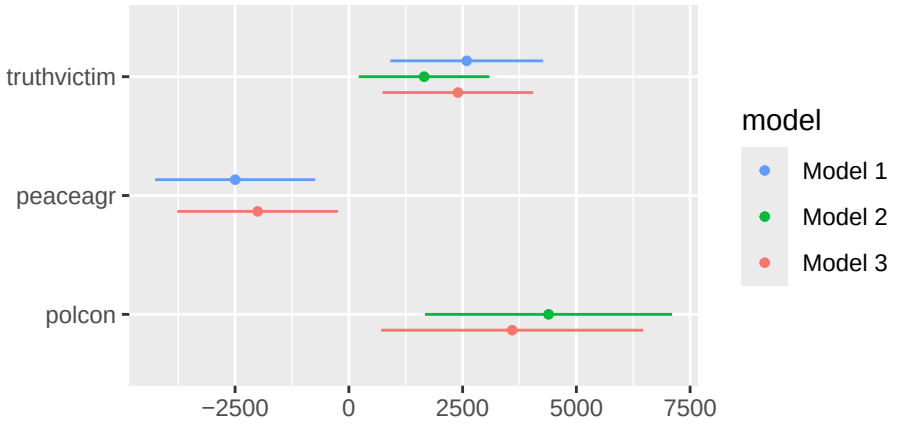
Next, we use `list()` within `dwplot()` to generate a dot-and-whisker plot that displays regression results from all models, similar to a multi-column regression table. This plot visualizes coefficient estimates and ilustrates variation of the `truthvictim` coefficient across three models

```
# Create a dot-and-whisker plot for m1, m2 and m3
dwplot(list(m1,m2,m3))
```



Note that the plot gets crowded as we add more predictors. For effective visualization, it is a good idea to focus on key variables. Let's focus on `truthvictim`, and the statistically significant conflict and political variables `peaceagr` and `polcon`.

```
# Create a dot-and-whisker plot for m1, m2 and m3 focusiing on truthvictim, peaceagr, polcon
dwplot(list(m1, m2, m3), vars_order = c("truthvictim", "peaceagr", "polcon"))
```

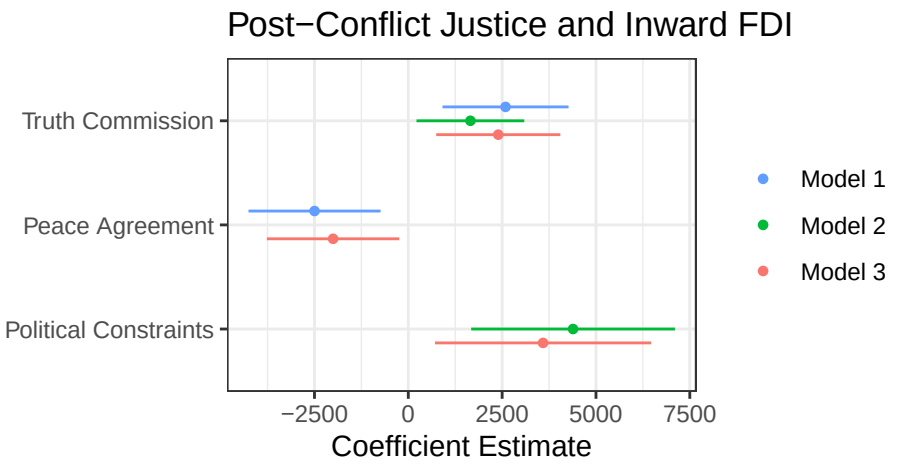


Customize the plot

As with `ggplot`, we can customize the dot-and-whisker plot for better clarity and visuals. At minimum, we want to select key variables, label our models and variables properly, and add details to clearly convey the plot's message (label the axis, add a title, simplify the background, etc.).

```
# Customize the plot
plot1 <- dwplot(list(m1,m2,m3),
         model_order = c("Model 1","Model 2","Model 3"),
         # Specifies the order of the models
         vars_order = c("truthvictim","peaceagr","polcon")) %>%
         # Select the order of the variables
  relabel_predictors(c(truthvictim = "Truth Commission",
                       peaceagr = "Peace Agreement",
                       polcon = "Political Constraints")) +
                       # Relabel the variables' names
  theme_bw() +
  # Applies a minimalistic black and white theme
  xlab("Coefficient Estimate") +
  ylab("") +
  # Label the axis
```

```
  ggtitle("Post-Conflict Justice and Inward FDI") +
  # Set the title of the plot
  theme(legend.title=element_blank())
  # Remove the legend title

plot1
```



Save the last version of your plot as a `.png` file using `ggsave()`, setting dimensions as needed.

Helpful Hint: To save your plot in R, don't forget to include the full file path and desired file name like this: `ggsave("your/filepath/your_filename.png")`

```
# Save the plot
ggsave("dwplot1.png", plot1, width = 6.5, height = 4.5, dpi = 400)
```

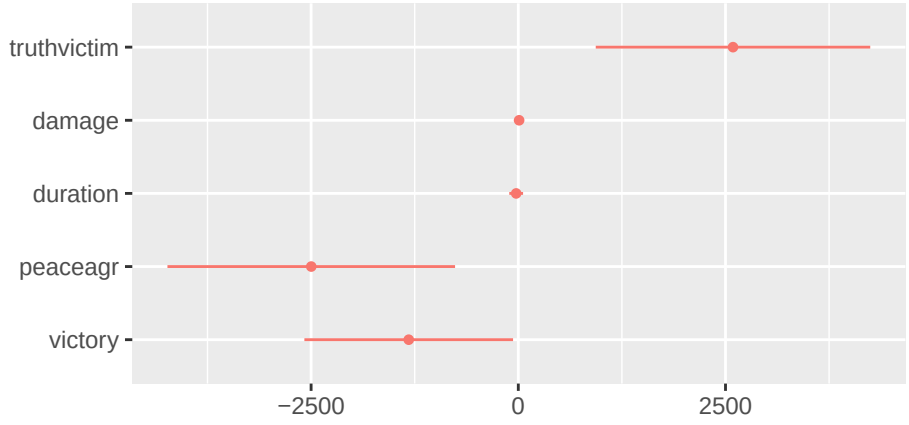
Approach 3: Using Tidied Results as Input

The `dwplot()` can also take inputs from regression results stored in tidy dataframes. We use the `tidy()` function from the `broom` package to tidy results and store them in dataframes.

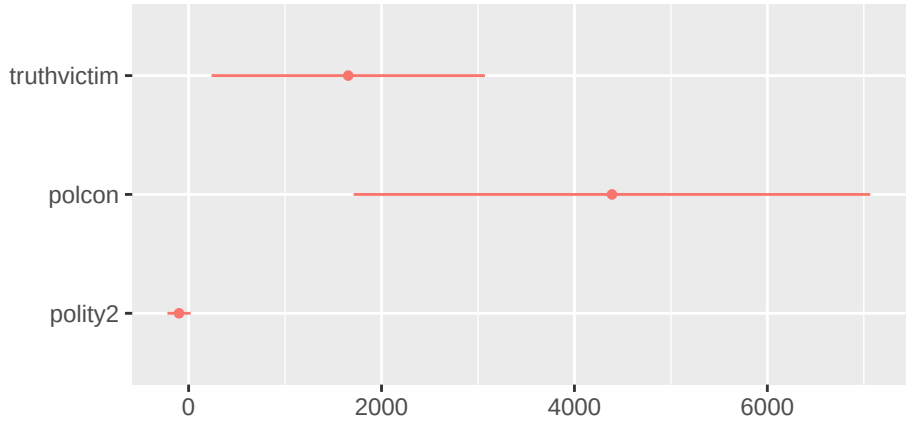
Generate dot-and-whisker plots from tidy dataframes

We use `tidy()` to tidy regression results. Tidy the regression models `m1`, `m2`, ans `m3` from Approach 2, and let's call our tidy dataframes `m1df`, `m2df`, and `m3df` respectively, to distinguish them from the regression objects.

```
# Create tidy dataframes from the models m1, m2 and m3
m1df <- tidy(m1)
m2df <- tidy(m2)
m3df <- tidy(m3)

# Create dot-and-whisker plots using dotwhisker
dwplot(m1df)
```

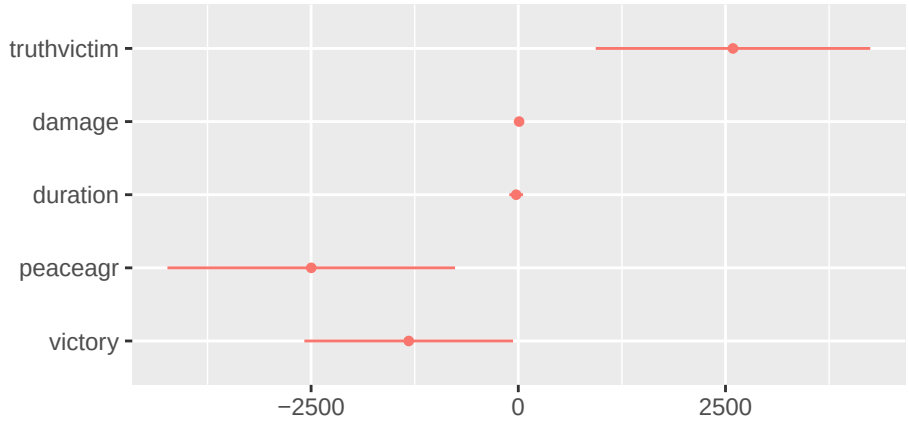



```
dwplot(m2df)
```



We see that `dwplot(m1df)` generates the same plot as `dwplot(m1)`, except that it does not omit the intercept by default. We can use the `filter()` function to drop the intercept.

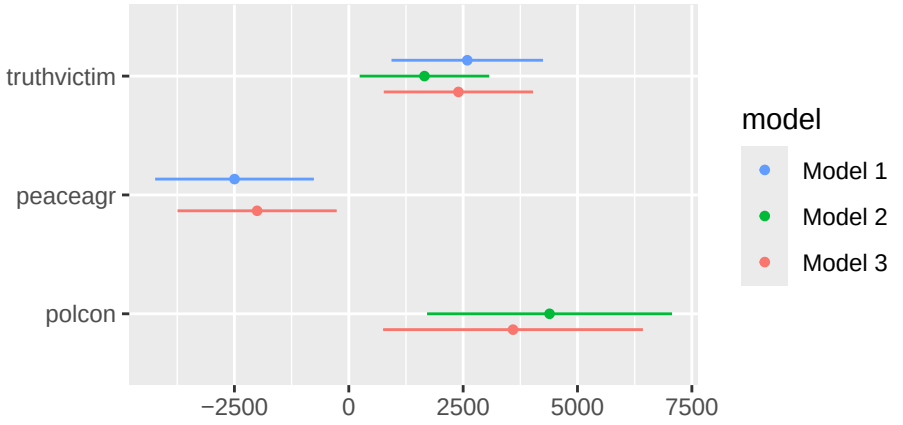
```
# Drop intercept using filter() function
m1df_no_intercept <- tidy(m1) %>%
  filter(term != "(Intercept)")

# Create dot-and-whisker plots using dotwhisker
dwplot(m1df_no_intercept)
```



Plot results from multiple regression models

We can also plot multiple regression models in one dot-and-whiskers plot with tidy regression results. To do so, let's filter the variables that we want to plot (`truthvictim`, `peaceagr` for `m1` and `m3`, and `polcon` for `m2` and `m3`) and add a column called `Model (model number)` to identify each model after we combine the tidy dataframes with `rbind()`. Lastly, generate a dot-and-whisker plot for the dataframe that has all the tidy dataframes merged.

```
# Tidy up the results from model m1, m2 and m3
m1df <- tidy(m1) %>%
  filter(term == "truthvictim" | term == "peaceagr") %>%
  # Filter to include only the terms 'truthvictim' and 'peaceagr'
  mutate(model = "Model 1")
  # Assigning the label 'Model 1' to these rows

m2df <- tidy(m2) %>%
  filter(term == "truthvictim" | term == "polcon") %>%
  # Filter to include only the terms 'truthvictim' and 'polcon'
  mutate(model = "Model 2")
  # Assigning the label 'Model 2' to these rows

m3df <- tidy(m3) %>%
  filter(term == "truthvictim" | term == "peaceagr" | term == "polcon") %>%
  # Filter to include only the terms 'truthvictim', 'peaceagr' and 'polcon'
  mutate(model = "Model 3")
  # Assigning the label 'Model 3' to these rows

# Combine the tidied dataframes into one
models <- rbind(m1df, m2df, m3df)

# Create a dot-and-whisker plot for 'models'
dwplot(models)
```

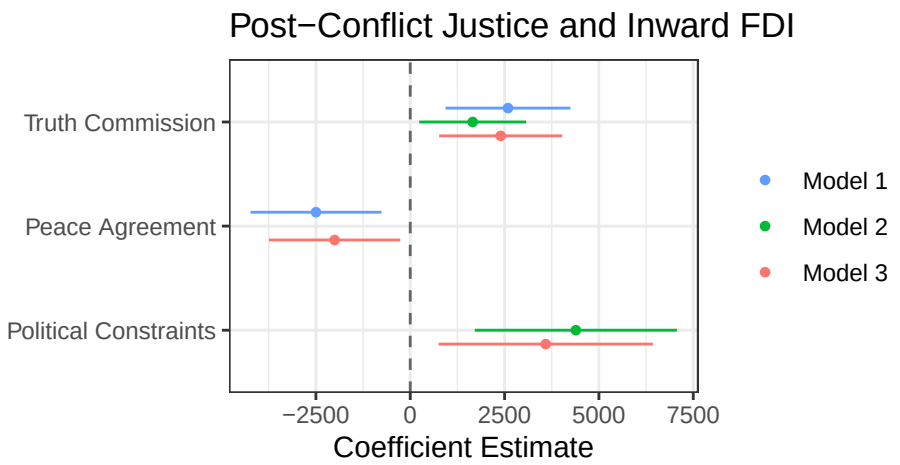


Customize the plot

Let's also customize the plot we just generated with our final tidy dataframe by replicating the plot you saved above (`dwplot1.png`) and storing it as `plot 2`. Save the plot as `dwplot2.png`using the `ggsave() function`.

Bonus: Add a dotted vertical line where x-intercept is 0. This is helpful for visualizing coefficient estimates with both positive and negative signs. The plot illustrates our finding that establishing a post-conflict justice institution correlates with increased FDI inflows, indicating a positive effect on attracting inward FDI.

```r
# Customize the plot
plot2 <- dwplot(models,
                model_order = c("Model 1","Model 2","Model 3"),
                # Specifies the order of the models
                vars_order = c("truthvictim","peaceagr","polcon")) %>%
                # Select the order of the variables
  relabel_predictors(c(truthvictim = "Truth Commission",
                       peaceagr = "Peace Agreement",
                       polcon = "Political Constraints")) +
                       # Relabel the variables' names
  theme_bw() +
  # Applies a minimalistic black and white theme
  xlab("Coefficient Estimate") +
  ylab("") +
  # Label the axis
  geom_vline(xintercept = 0,
             colour = "grey40",
             linetype = 2) +
  # Bonus: Add a vertical line at x-intercept = 0
  ggtitle("Post-Conflict Justice and Inward FDI") +
  # Set the title of the plot
  theme(legend.title=element_blank())
  # Remove the legend title

plot2
```




```r
# Save the plot
ggsave("dwplot2.png", plot2, width = 6.5, height = 4.5, dpi = 400)
```