Coefficent Plots using ggplot2

Rafael Campos-Gottardo*

2025-03-22

In addition to being a powerful tool for statistical analysis R it provides acess to `ggplot2` a package designed for data visualization.

One of the most important skills when learning regression is how to visualize the results from a regression model. Although you can present regression results in a table, these results are harder to interpret and more cumbersome for readers. Therefore, regression tables are usually presented in the appendix and various graphs are presented in the main body. In this lab guide we are going to focus on using `ggplot` to create a coefficent plot that you can use in your final paper.

Let's get started with data visualization by loading the `tidyverse` which contains `ggplot2`.

Load packages

```
# install.packages("tidyverse") # Install the package if you have not already done so.
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

Now that we have ggplot loaded let's create a simply regression model to use for our visualization. Today we will use data from the Canadian Election Study (CES). In this class we have been mostly working with country-year data, however, a lot of quantitative research is done with public opinion data like the Canadian election study. I have uploaded a cleaned version of the CES for this lab.

However, notice that the data is not a `.csv` file but is a `.dta` file. Most public opinion data is uploaded as `.dta` files (`Stata` files) because it preseves the labels. In order to open `.dta` files in R we need to use the `haven` package.

```
# install.packages("haven")
library(haven)
```

Load data

Now we can use the `read_dta()` function to open the data file.

*This lab guide builds on those created by Aaron Erlich and Colin Scott. I would also like to acknowledge Brendan Szendro, Nicholas Vaxelaire, Guila Cohen, and Natalie Delmonte for their help developing this lab guide.

```
CES <- read_dta("poli311_CES.dta")
```

Now we can use the head function to inspect the data. You'll notice that there are 5 variables including vote choice, gender, satsifaction with democracy, and province of residence.

```
head(CES)
```

```
## # A tibble: 6 x 5
##     Age Province         DemSat Gender  VoteChoice
##   <dbl> <chr>             <dbl> <chr>   <chr>
## 1    57 Quebec                3 A Man   ""
## 2    22 British Columbia      3 A Woman "NDP"
## 3    28 British Columbia      3 A Woman ""
## 4    29 Ontario               4 A Woman ""
## 5    41 Quebec                3 A Woman "NDP"
## 6    63 Quebec                3 A Woman "Bloc Quebecois"
```

Clean data

Before we fit a regression model we need to clean up the vote choice variable. Using the unique function we can see that there is a blank category. We want to remove the blank (" ") category and also make the Liberal Party the reference category.

```
unique(CES$VoteChoice)
```

```
## [1] ""                   "NDP"                "Bloc Quebecois"
## [4] "Conservative Party" "Liberal Party"      "Another Party"
## [7] "Green Party"
```

```
CES <- CES %>%
  mutate(VoteChoice = replace(VoteChoice, VoteChoice == "", NA), # Replace blank party with NA
         VoteChoice = factor(VoteChoice, levels = c(
           "Liberal Party", "Conservative Party", "NDP",
           "Bloc Quebecois", "Green Party", "Another Party")
           ) # Make the Liberal Party the reference category
         )
```

Fit a regression model

```
model_1 <- lm(DemSat ~ Age + Gender + VoteChoice, data = CES)
```

Now that we have fit a regression model regressing satisfaction with democracy on Age, Gender, and Vote Choice. After we fit the regression model we need to extract the coefficients from it to make a coefficient plot. If we look at the `class` of our model object we can see that it is an `lm` object and `ggplot` requires a data object as the first argument.

```
class(model_1)
```

```
## [1] "lm"
```

Create a data frame from our model

In order to create a data object from our model, we can use the `tidy()` function from the `broom` package.

```
# install.packages("broom")
library(broom)
```

```r
# Create a data frame (tibble) from the regression model

model_1_df <- tidy(model_1,
                   # We want to specify that we want a confidence interval
                   # for error bars
                   conf.int = TRUE)
```

Now that we have created a data set we can manipulate it to clean up the code for graphing and create a coefficient plot.

1. First, we need to use the `filter()` function to remove unwanted terms like the intercept. We would also filter out the coefficients for fixed effects (e.g. country, year, etc).
2. Second, we want to rename our terms to make the graph more presentable. Finally, we want to make the term variable a factor to make sure they display in the right order. The y axis renders from bottom to top so we have to specify the variables backwards in the `factor()` function.

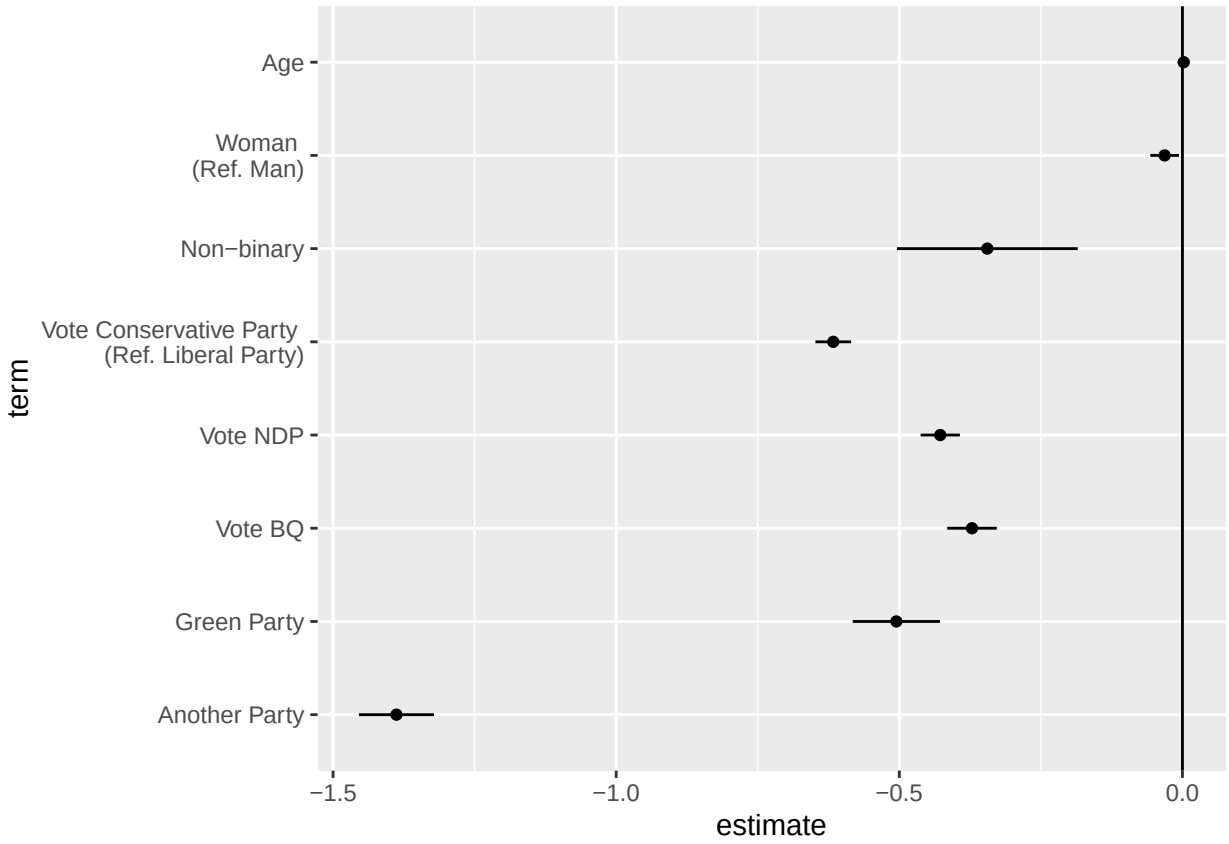
```r
model_1_df <- model_1_df %>%
  filter(term != "(Intercept)") %>% # remove the intercept
  mutate(term = case_match(term, # Rename the terms to make a cleaner plot
                           "Age" ~ "Age",
                           "GenderA Woman" ~ "Woman \n (Ref. Man)",
                           # \n puts the text after on the next line
                            "GenderNon-binary/Other Gender" ~ "Non-binary",
                            "VoteChoiceConservative Party" ~
                             "Vote Conservative Party \n (Ref. Liberal Party)",
                          "VoteChoiceNDP" ~ "Vote NDP",
                          "VoteChoiceBloc Quebecois" ~ "Vote BQ",
                          "VoteChoiceGreen Party" ~ "Green Party",
                          "VoteChoiceAnother Party" ~ "Another Party"
                           ),
         term = factor(term, levels = c("Another Party", "Green Party", "Vote BQ",
                                        "Vote NDP",
                                        "Vote Conservative Party \n (Ref. Liberal Party)",
                                        "Non-binary",
                                        "Woman \n (Ref. Man)",
                                        "Age"
                                        )))
```

Create a coefficent plot

Now that we have cleaned our dataset we can create a coefficient plot. Our x value is going to be our estimate, our x value is the variables (or term), and the xmin and xmax are our confidence intervals for our error bars.

The two geoms we use are `geom_point()` for the coefficient estimates and `geom_linerange()` for the confidence intervals. The x and y values are for `geom_point()` and the xmin and xmax are for `geom_linerange()`. However, we specify all of them in the aesthetics function (`aes()`) for `ggplot()`. Finally, we want to add a vertical line at 0 to see if the 95% confidence intervals for the regression coefficients touch 0. If the 95% confidence interval touches 0 then the coefficient on that variable is not statistically signficiant. To add this line we use `geom_vline()` and specify that the xintercept is at 0.

```r
model_1_df %>%
  ggplot(aes(x = estimate, y = term, xmin = conf.low, xmax = conf.high)) +
  geom_point() +
  geom_linerange() +
  geom_vline(xintercept = 0)
```



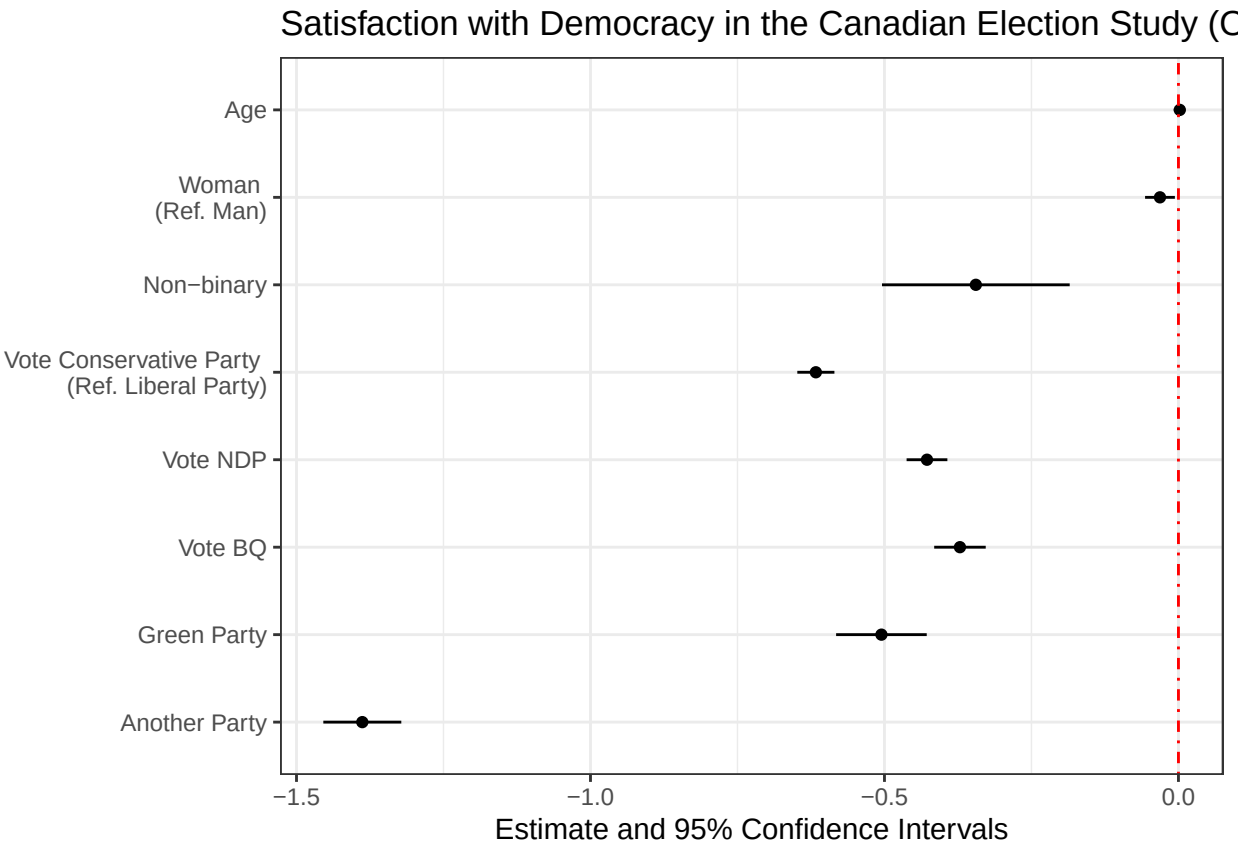
Now that we have a basic coefficient plot we can clean it up.

We want to colour our coefficients black if they are statistically significant and red if they are not, change our labels, change the line type for our x intercept, and change the theme.

Table 1: Most Common Line types ("lty")

Line type	Name
1	"solid"
2	"dashed"
3	"dotted"
4	"dotdash"

```
model_1_df %>%
  ggplot(aes(x = estimate, y = term, xmin = conf.low, xmax = conf.high)) +
  geom_point(colour = ifelse(model_1_df$p.value < 0.05, "black", "red")) +
  geom_linerange(colour = ifelse(model_1_df$p.value < 0.05, "black", "red")) +
  geom_vline(xintercept = 0,
             col = "red", # col is short for colour
             lty = 4) + # lty is short for linetype and lty 4 is a dashed and dotted line
  labs(title = "Satisfaction with Democracy in the Canadian Election Study (OLS Model)",
       x = "Estimate and 95% Confidence Intervals",
       y = NULL) + # We do not need a y axis label
  theme_bw()
```



Now we have a complete coefficient plot that we can present in a quantitative research paper.

Regression Tables

In addition to the coefficient graph, you should present your results in a regression table in the appendix of your paper. However, you should not just copy the regression output from R. To make R output a nice regression table we can use the `modelsummary()` function from the `modelsummary` package. First, we need to install and load the package.

```
# install.packages("modelsummary")
library(modelsummary)
```

```
## `modelsummary` 2.0.0 now uses `tinytable` as its default table-drawing
##   backend. Learn more at: https://vincentarelbundock.github.io/tinytable/
##
## Revert to `kableExtra` for one session:
##
##   options(modelsummary_factory_default = 'kableExtra')
##   options(modelsummary_factory_latex = 'kableExtra')
##   options(modelsummary_factory_html = 'kableExtra')
##
## Silence this message forever:
##
##   config_modelsummary(startup_message = FALSE)
```

Now we can use the `modelsummary()` function to display our regression model. We simply have to name our model in the function and then specify `stars = TRUE`. The stars argument adds the stars that indicate the significance level from the regression output produced by the `summary()` function.

	(1)
(Intercept)	3.116***
	(0.025)
Age	0.002***
	(0.000)
GenderA Woman	−0.031*
	(0.013)
GenderNon-binary/Other Gender	−0.345***
	(0.081)
VoteChoiceConservative Party	−0.617***
	(0.016)
VoteChoiceNDP	−0.428***
	(0.018)
VoteChoiceBloc Quebecois	−0.372***
	(0.022)
VoteChoiceGreen Party	−0.505***
	(0.039)
VoteChoiceAnother Party	−1.388***
	(0.034)
Num.Obs.	12 292
R2	0.180
R2 Adj.	0.179
AIC	25 824.2
BIC	25 898.4
Log.Lik.	−12 902.111
RMSE	0.69

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

```
modelsummary(model_1, stars = TRUE)
```

You will notice that we also need to clean up our variables in our model summary. We can use the map argument to change the variable names and order. We can use the same code that we used in the `case_match()` function for the map argument we just need to change the ~ to an equals sign.

```
modelsummary(model_1, stars = TRUE,
             coef_map = c("Age" = "Age",
                          "GenderA Woman" = "Woman (Ref. Man)",
                            "GenderNon-binary/Other Gender" = "Non-binary",
                            "VoteChoiceConservative Party" =
                             "Vote Conservative Party (Ref. Liberal Party)",
                          "VoteChoiceNDP" = "Vote NDP",
                          "VoteChoiceBloc Quebecois" = "Vote BQ",
                          "VoteChoiceGreen Party" = "Green Party",
                          "VoteChoiceAnother Party" = "Another Party"
```

	(1)
Age	0.002***
	(0.000)
Woman (Ref. Man)	−0.031*
	(0.013)
Non-binary	−0.345***
	(0.081)
Vote Conservative Party (Ref. Liberal Party)	−0.617***
	(0.016)
Vote NDP	−0.428***
	(0.018)
Vote BQ	−0.372***
	(0.022)
Green Party	−0.505***
	(0.039)
Another Party	−1.388***
	(0.034)
Num.Obs.	12 292
R2	0.180
R2 Adj.	0.179
AIC	25 824.2
BIC	25 898.4
Log.Lik.	−12 902.111
RMSE	0.69

+ p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

```
))
```

Finally, if you want to display multiple models in the same table you can feed in multiple models using the `list()` function: `modelsummary(list(model1, model2), stars = TRUE)`.