Econometrics in R

Luke DiMartino

December 18th, 2021

Contents

1 Introduction

1.1 Purpose

In this paper, I summarize methods for conducting statistical tests and fitting regressions from introductory econometrics (at Georgetown, the Introduction to Econometrics class) in the R programming language. Despite my experience with data import, wrangling, cleaning, and visualization in R, I know nearly nothing about modelling in R - the vast majority of my experience modelling is from economics classes taught in Stata.

This paper logs my own work learning the regression workflow in R. My goal is to learn the best methods for developing models in R that are simple but can extend beyond introductory econometrics. Since I suspect some people, particularly undegraduates at Georgetown, have experience similar to my own, I have dedicated a few hours to transforming my work into this open-source guide for anyone in a similar position to myself.[1] If you are in this position, this paper may be helpful to you. Beyond the mathematics, the only other pre-requisite is a basic understanding of R. I frequently refer to Stata's regression workflow, but it is not a necessary reference point.

The license for this is in the GitHub repository in which it is stored. In short, distribute this to whomever you please.

Since I am well-versed in data import, wrangling, and visualization in R, I will avoid these topics, except when they are a relevant to the workflow of developing models. There is ample high-quality, free instructional material on these topics: Data Science for R by Hadley Wickham and Garrett Grolemund is the standard recommendation for new R users. Modern R by Bruno Rodrigues, although unfinished, is a rigorous introduction to R's data structures and functional programming capabilities.

1.2 Workflow

R's workflow for analysis is a bit different than Stata's. The famous saying to remember is:

> "Everything in R is an object, and everything that happens in R is a function call."

Stata's workflow is:

1. regression command stores a model in memory
2. post-estimation command(s) extract values or visualize data

Stata holds the last regression in memory and applies relevant post-estimation commands to it. R is more thorough and explicit because data and regression models are both stored as objects in working memory.

R's workflow is:

1. regression function creates model object
2. analysis functions extract data (residuals, predictions, statistical summaries, coefficients, etc.) from model
3. cleaning functions prepare that data for visualization

Unfortunately, modelling in R is more verbose and less user-friendly than in Stata; nevertheless, many economists and data scientists use it as their primary tool for analysis. While the few lines of code for modelling are more annoying in R than in Stata, mastering them prepares you to use a more powerful, flexible, and general language for data analysis.[2]

[1] I suspect I am in a somewhat large group of people who know the math (or enough of it, at least) to do basic econometric analysis, but learned to conduct it in Stata. As an undergrad at Georgetown, at least, that is standard.

[2] A brief defense of R: Modelling is Stata's bread-and-butter. R's advantages are in every other part of the workflow: R

1.3 R and Stata

R makes one critical tradeoff: R is free and Stata is very much not. This is a win for R in my book, but it affects the development of new functionality in each language. Both languages have solid, built-in matrix algebra computation for regression. Most regression is possible with only a handful of matrix manipulations.

In layering advanced functionality, like single commands/functions for complex models and more complex standard error calculations, the language differ. The paid engineers at StataCorp developed functionality for all different types of regression modelling. Since they are at one company, the syntax, output, and general behavior is standardized.

R has the same functionality in open-source packages. It is difficult to overstate how great the R open-source community is and how easy programming is with well-built R packages that cover every general-purpose programming and data science topic.[3] The downside is that to learn new techniques, you must choose between multiple packages and become familiar with the syntax and output of one.[4]

has general programming language features that are well-optimized, like simultaneous storage for multiple objects/datasets, optimized looping and conditionals, operating system interaction functions, command line access, easy API integration, and custom function and package development, not to mention the `tidyverse`, the best combination of power and simplicity for data import, wrangling, and visualization.

R's large open-source community is constantly developing new packages that multiply R's power. Look at how many packages I use in this paper alone! The `wooldridge` package is a simple example. Instead of downloading, storing, and managing dozens of datasets, I can call them with `d <- wooldridge::dataset_name`.

R is substantially faster than Stata as well. Stata's speed with larger datasets is arguably the lynchpin factor in industry for abandoning Stata. If you are interested in data science, Stata simply lacks functionality for advanced models including machine learning regression and classification techniques, natural language processing, and the like. R also has a native presentation format, RMarkdown, which I used to compose this document. I did little more than type the text you see, prepend 15 lines of format settings, and mark off code chunks. `knitr` compiles this document using LaTeX, prints the code output, and voila! The total time to knit it is about 20 seconds, with integrated code and output, and support for LaTeX equations if we were to need it.

Few of these features matter with the small, curated datasets in this paper, but in a standard project involving data management, API calls, and data cleaning Stata will struggle while R hits its stride.

[3] As of December 2021, there are more than 18,000 certified packages on the Comprehensive R Archive Network (CRAN), where R looks when you call `install.packages()`. There are thousands more on GitHub and stored locally by R users everywhere.

[4] Packages may also have dependency issues or stop receiving updates. In my opinion, this disadvantage is heavily overstated. The community is large and strong enough that the requisite packages to do undergraduate/master's econometrics are constantly updated. Computer scientists get worried about dependencies because their code goes into production - it is executed on a server automatically and in quasi-constant operation. Therefore, if one function fails, the entire system probably will too. If a data analysis tool is going to stop receiving updates, users and dependent packages have months if not years from the initial announcement to find a different tool. An analysis may not be precisely reproducible, but stored results remain valid.

1.4 Data

I exclusively use data from the `wooldridge` package. For more information on any dataset, find the `wooldridge` documentation here.

2 Setup

This is information required to recreate my code. Ignore this section if you are reading as a reference.

2.1 Packages

These are the packages I use in this paper.

```r
library(wooldridge) # Necessary for datasets
library(knitr) # Required
library(tidyverse) # For the occasional data manipulation and visualization
# dplyr is sufficient in most cases
library(lmtest) # For statistical tests
library(sandwich) # For statistical tests
library(estimatr) # For heteroskedasticity-robust modelling
library(parameters) # For parameter statistic tables
library(modelsummary) # For model and data summaries
library(fastDummies) # For dummy variables
library(margins) # For estimating marginal effects
library(did) # For advanced dif-in-dif
library(tsibble) # For time-series data
library(lubridate) # For time variables
library(fable) # For time-series modelling
library(fixest) # For fixed effects modelling
library(ivreg) # For instrumental variables
library(AER) # For Tobit model
library(sampleSelection) # For Heckman model
```

2.2 Version

If you are having difficulty reproducing my code, here is the basic version information. My packages are essentially up-to-date as of November 1st, 2021.

```r
sessionInfo()
```

```
R version 4.1.1 (2021-08-10)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] sampleSelection_1.2-12 maxLik_1.5-2           miscTools_0.6-26
```

```
 [4] AER_1.2-9            survival_3.2-11      car_3.0-12
 [7] carData_3.0-4        ivreg_0.6-1          fixest_0.10.1
[10] fable_0.3.1          fabletools_0.3.2     lubridate_1.8.0
[13] tsibble_1.1.1        did_2.1.0            margins_0.3.26
[16] fastDummies_1.6.3    modelsummary_0.9.4   parameters_0.15.0
[19] estimatr_0.30.4      sandwich_3.0-1       lmtest_0.9-38
[22] zoo_1.8-9            forcats_0.5.1        stringr_1.4.0
[25] dplyr_1.0.7          purrr_0.3.4          readr_2.1.0
[28] tidyr_1.1.4          tibble_3.1.6         tidyverse_1.3.1
[31] knitr_1.33           wooldridge_1.4-2     ggplot2_3.3.5
[34] hrbrthemes_0.8.0

loaded via a namespace (and not attached):
 [1] VGAM_1.1-5          colorspace_2.0-2    ggsignif_0.6.3
 [4] ellipsis_0.3.2      fs_1.5.0            rstudioapi_0.13
 [7] ggpubr_0.4.0        farver_2.1.0        fansi_0.5.0
[10] mvtnorm_1.1-3       xml2_1.3.2          splines_4.1.1
[13] extrafont_0.17      Formula_1.2-4       jsonlite_1.7.2
[16] broom_0.7.9         Rttf2pt1_1.3.9      anytime_0.3.9
[19] dbplyr_2.1.1        compiler_4.1.1      httr_1.4.2
[22] backports_1.2.1     assertthat_0.2.1    Matrix_1.3-4
[25] fastmap_1.1.0       cli_3.1.0           htmltools_0.5.2
[28] tools_4.1.1         gtable_0.3.0        glue_1.4.2
[31] dreamerr_1.2.3      tables_0.9.6        Rcpp_1.0.7
[34] cellranger_1.1.0    vctrs_0.3.8         nlme_3.1-152
[37] extrafontdb_1.0     insight_0.14.4      xfun_0.25
[40] rvest_1.0.1         lifecycle_1.0.0     rstatix_0.7.0
[43] MASS_7.3-54         scales_1.1.1        hms_1.1.0
[46] yaml_2.2.1          gdtools_0.2.3       stringi_1.7.3
[49] bayestestR_0.11.0   rlang_0.4.11        pkgconfig_2.0.3
[52] systemfonts_1.0.2   distributional_0.2.2 systemfit_1.1-24
[55] evaluate_0.14       lattice_0.20-44     prediction_0.3.14
[58] tidyselect_1.1.1    magrittr_2.0.1      R6_2.5.1
[61] generics_0.1.0      DBI_1.1.1           pillar_1.6.2
[64] haven_2.4.3         withr_2.4.2         datawizard_0.2.1
[67] abind_1.4-5         modelr_0.1.8        crayon_1.4.1
[70] utf8_1.2.2          tzdb_0.1.2          rmarkdown_2.10
[73] grid_4.1.1          readxl_1.3.1        data.table_1.14.0
[76] reprex_2.0.1        digest_0.6.27       numDeriv_2016.8-1.1
[79] stats4_4.1.1        munsell_0.5.0
```

3 Pre-Regression Analysis

3.1 Correlation and Univariate Analysis

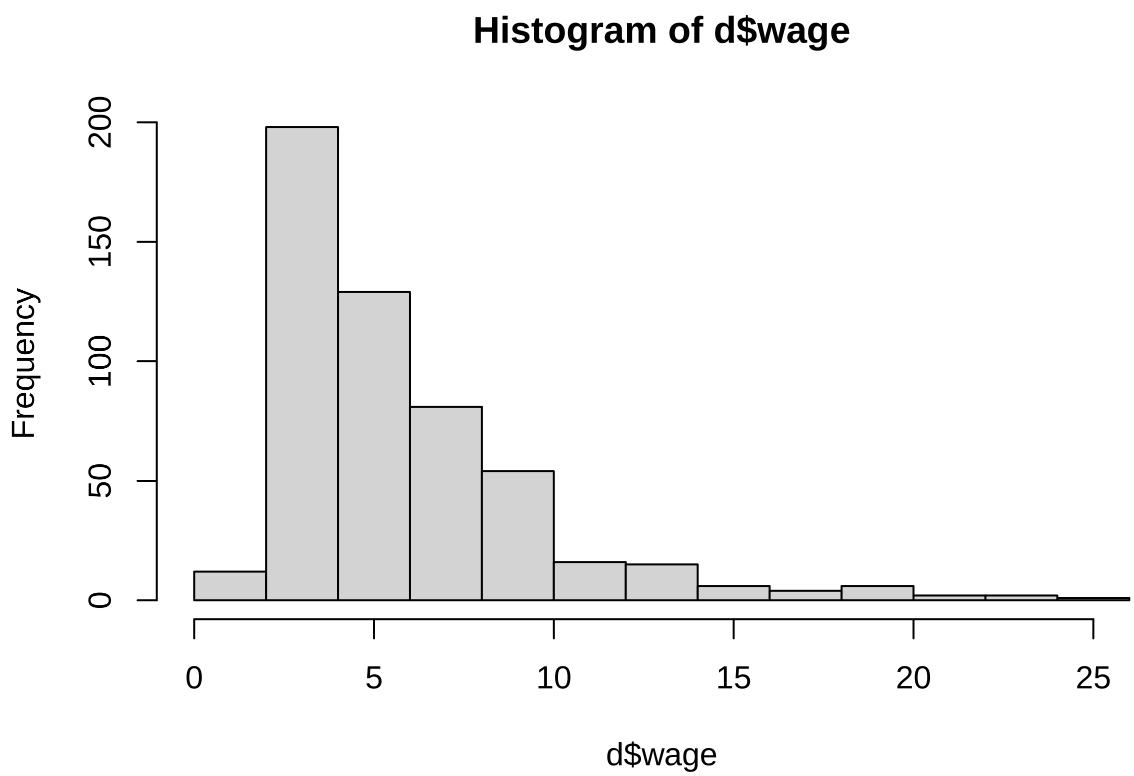
First, load data and find the correlation between the x and y variables.[5] These data are yearly US consumption growth and disposable income growth data from the Bureau of Labor Statistics.

`hist()`, `barplot()`, `boxplot()`, `plot()`, or more sophisticated `ggplot2` methods visualize distributions of one variable, the first step in the regression workflow.

```
d <- wooldridge::wage1
cor(d$wage, d$educ, use = "complete.obs")
```

```
[1] 0.4059033
```

```
hist(d$wage)
```



That graph is pretty ugly, but it's useful to have such concise syntax for workflow visualizations that help analysis. `ggplot2`'s power will be useful for graphs that are for others.

3.2 Dataset Overview with `modelsummary`

With unfamiliar data, a fundamental grasp of each variable is essential. The `modelsummary` package provides a family of `datasummary_*()` functions that do a ridiculous amount of work for you.

This data is wage data - each observation is a worker.

[5]As a matter of syntax, I always store the relevant data as `d`, as it is the sole data of interest. The rest of my syntax follows standard R conventions.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
wage	241	0	5.9	3.7	0.5	4.7	25.0	
educ	18	0	12.6	2.8	0.0	12.0	18.0	
exper	51	0	17.0	13.6	1.0	13.5	51.0	
tenure	34	0	5.1	7.2	0.0	2.0	44.0	
nonwhite	2	0	0.1	0.3	0.0	0.0	1.0	
female	2	0	0.5	0.5	0.0	0.0	1.0	
married	2	0	0.6	0.5	0.0	1.0	1.0	
numdep	7	0	1.0	1.3	0.0	1.0	6.0	
smsa	2	0	0.7	0.4	0.0	1.0	1.0	
northcen	2	0	0.3	0.4	0.0	0.0	1.0	
south	2	0	0.4	0.5	0.0	0.0	1.0	
west	2	0	0.2	0.4	0.0	0.0	1.0	
construc	2	0	0.0	0.2	0.0	0.0	1.0	
ndurman	2	0	0.1	0.3	0.0	0.0	1.0	
trcommpu	2	0	0.0	0.2	0.0	0.0	1.0	
trade	2	0	0.3	0.5	0.0	0.0	1.0	
services	2	0	0.1	0.3	0.0	0.0	1.0	
profserv	2	0	0.3	0.4	0.0	0.0	1.0	
profocc	2	0	0.4	0.5	0.0	0.0	1.0	
clerocc	2	0	0.2	0.4	0.0	0.0	1.0	
servocc	2	0	0.1	0.3	0.0	0.0	1.0	
lwage	241	0	1.6	0.5	-0.6	1.5	3.2	
expersq	51	0	473.4	616.0	1.0	182.5	2601.0	
tenursq	34	0	78.2	199.4	0.0	4.0	1936.0	

```
d <- wooldridge::wage1
datasummary_skim(d, output = "kableExtra")
```

Other functions in the family draw correlation tables, examine individual variables, and much more. This is an incredibly useful start to data analysis.

4 Ordinary Least Squares Regression

4.1 Simple Linear Regression Model Fitting

The standard workflow uses `lm()`, `summary()`, and `plot()` to fit and analyze regression models.[6] These functions replicate the process of regression analysis in Stata. They provide easy-to-analyze outputs, but analysis is tricky to extend (i.e. it is annoying to try to store any value, like an r-squared from a regression, as a variable for later analysis).

```
d <- wooldridge::consump
lm_base <- lm(gc ~ gy, data = d)
```

Nothing appears! Remember that R is built to manipulate data structures. The model has been fit and sits as an object in memory. The easiest way to see the data (although not to manipulate it further) is with the `summary()` command. This workflow is almost as close to Stata as it gets.

```
summary(lm_base)
```

```

Call:
lm(formula = gc ~ gy, data = d)

Residuals:
       Min         1Q     Median         3Q        Max 
-0.0140496 -0.0035407 -0.0005813  0.0044080  0.0116890 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 0.008079   0.001899   4.254 0.000155 ***
gy          0.570781   0.067354   8.474 6.75e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.007268 on 34 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.6787,	Adjusted R-squared:  0.6692 
F-statistic: 71.81 on 1 and 34 DF,  p-value: 6.754e-10
```

There is one problem - there is one `NA` in the data, a missing value. While R understands this and fits the model just fine, predictions and residuals will not fit into our data frame because each vector is too short.

```
lm_base <- lm(gc ~ gy, data = d, na.action = na.exclude)
summary(lm_base)
```

```

Call:
lm(formula = gc ~ gy, data = d, na.action = na.exclude)

Residuals:
       Min         1Q     Median         3Q        Max 
```

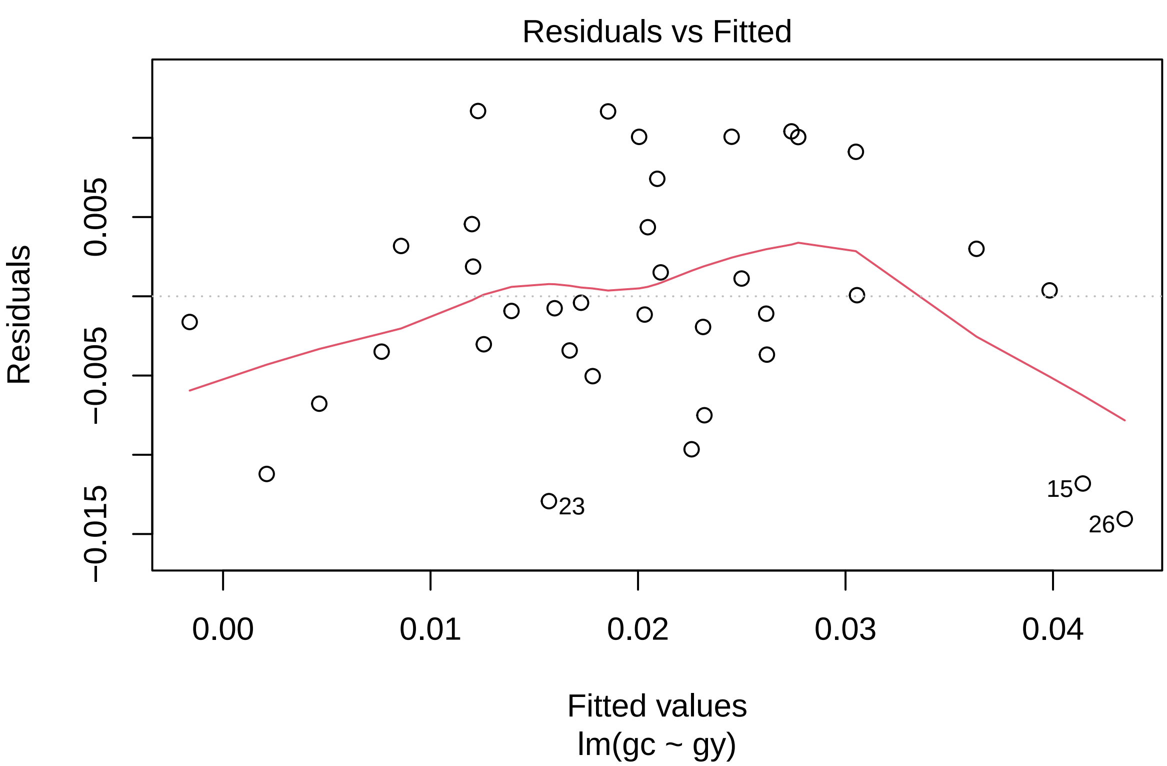
[6] An alternative workflow is available via the `tidymodels` package. `tidymodels` is not particularly popular and if you think `base` is verbose, be ready for a lot of typing if you choose to learn it. For the purposes of this paper, I ignore it. However, if you are interested in data science applications, machine learning, or any modelling that involves substantial preprocessing and model testing, `tidymodels` is one of many good packages to learn.

```
-0.0140496 -0.0035407 -0.0005813  0.0044080  0.0116890

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.008079   0.001899   4.254 0.000155 ***
gy          0.570781   0.067354   8.474 6.75e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.007268 on 34 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.6787,    Adjusted R-squared:  0.6692
F-statistic: 71.81 on 1 and 34 DF,  p-value: 6.754e-10
```

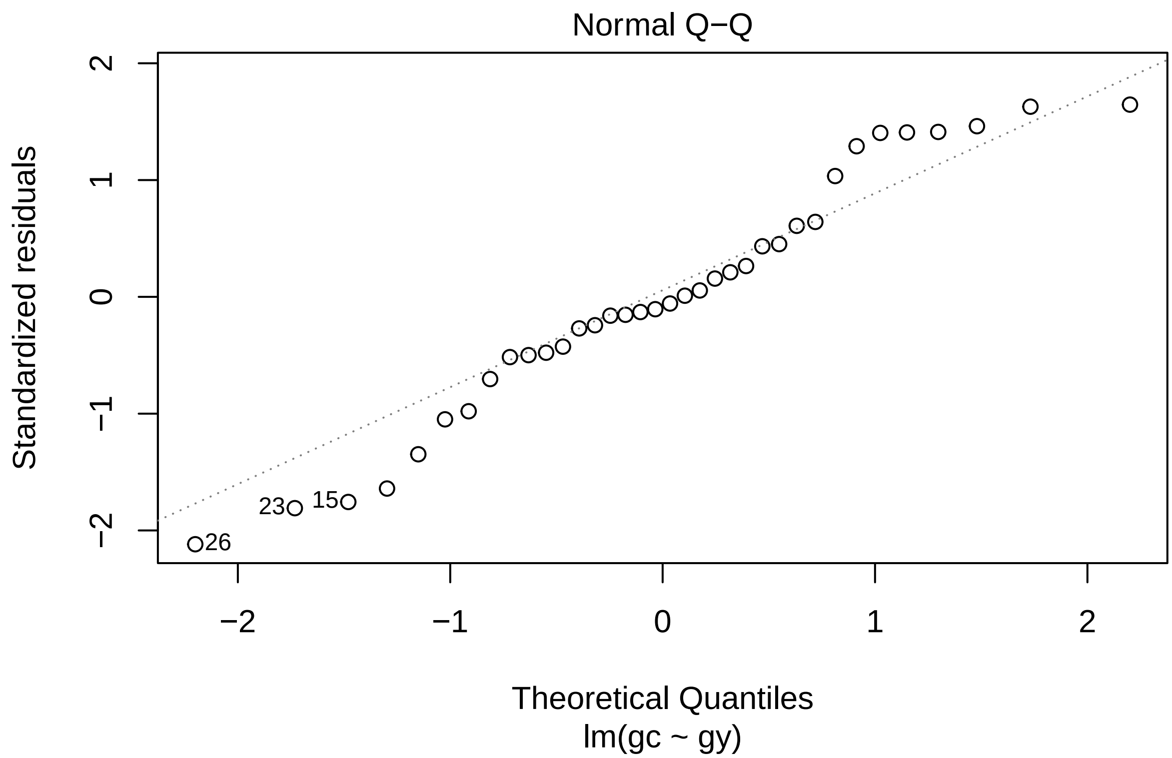
The `na.exclude` option fixes this for now.[7]

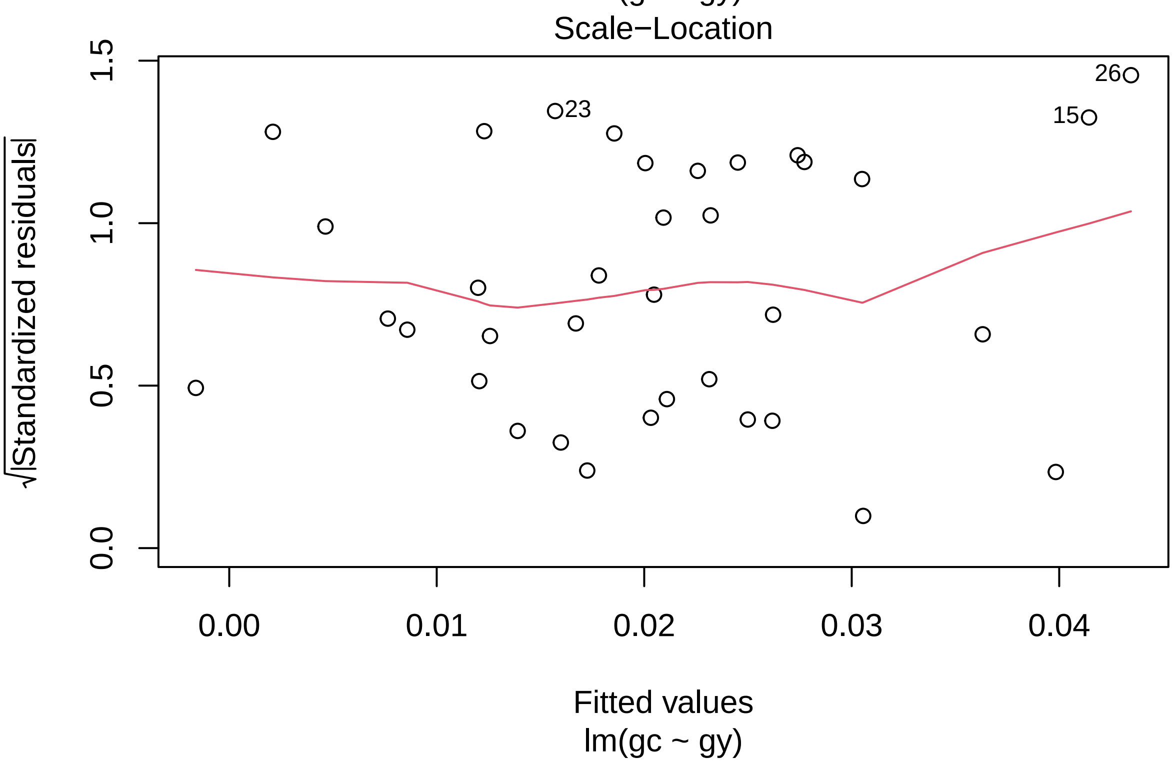
The `plot()` function prints regression summary plots.

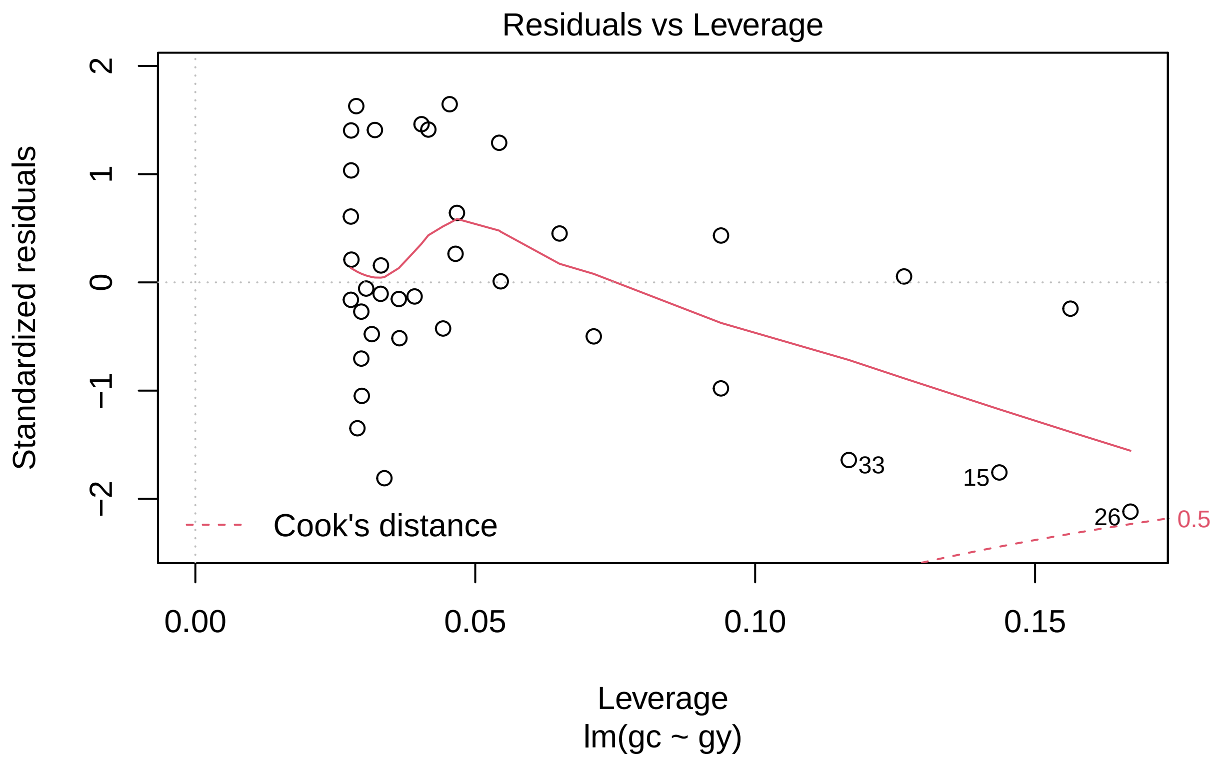
```
plot(lm_base)
```



[7] Sorry, Stata lovers. Get used to passing arguments for edge cases like `NA`s in the data. R tries to keep you going (admittedly, not as much as Stata), but is designed around flexibility and control. A part of the design philosophy is that these decisions matter, so they should be declared explicitly.


Normal Q−Q
Standardized residuals
2
1
0
−1
−2
26
23
15
−2
−1
0
1
2
Theoretical Quantiles
lm(gc ~ gy)


Scale−Location
√|Standardized residuals|
1.5
1.0
0.5
0.0
23
26
15
0.00
0.01
0.02
0.03
0.04
Fitted values
lm(gc ~ gy)



Let's extract predictions and residuals and store them as new variables in our data frame for further analysis.

```
d <- mutate(d,
                        yhat = predict(lm_base),
                        u = residuals(lm_base)
)
```

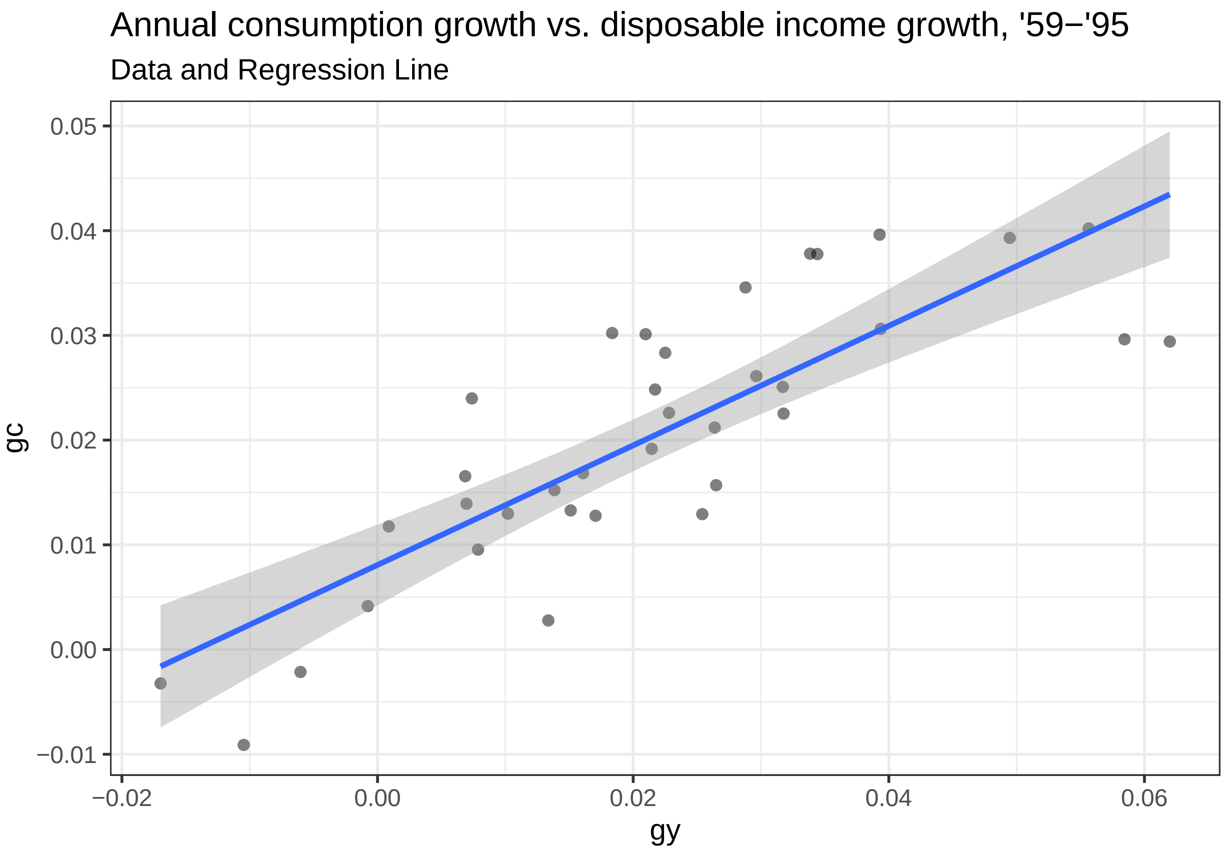
That looks pretty good! This is the bare bones of the regression workflow in R.

4.2 Custom Visualizations with ggplot2

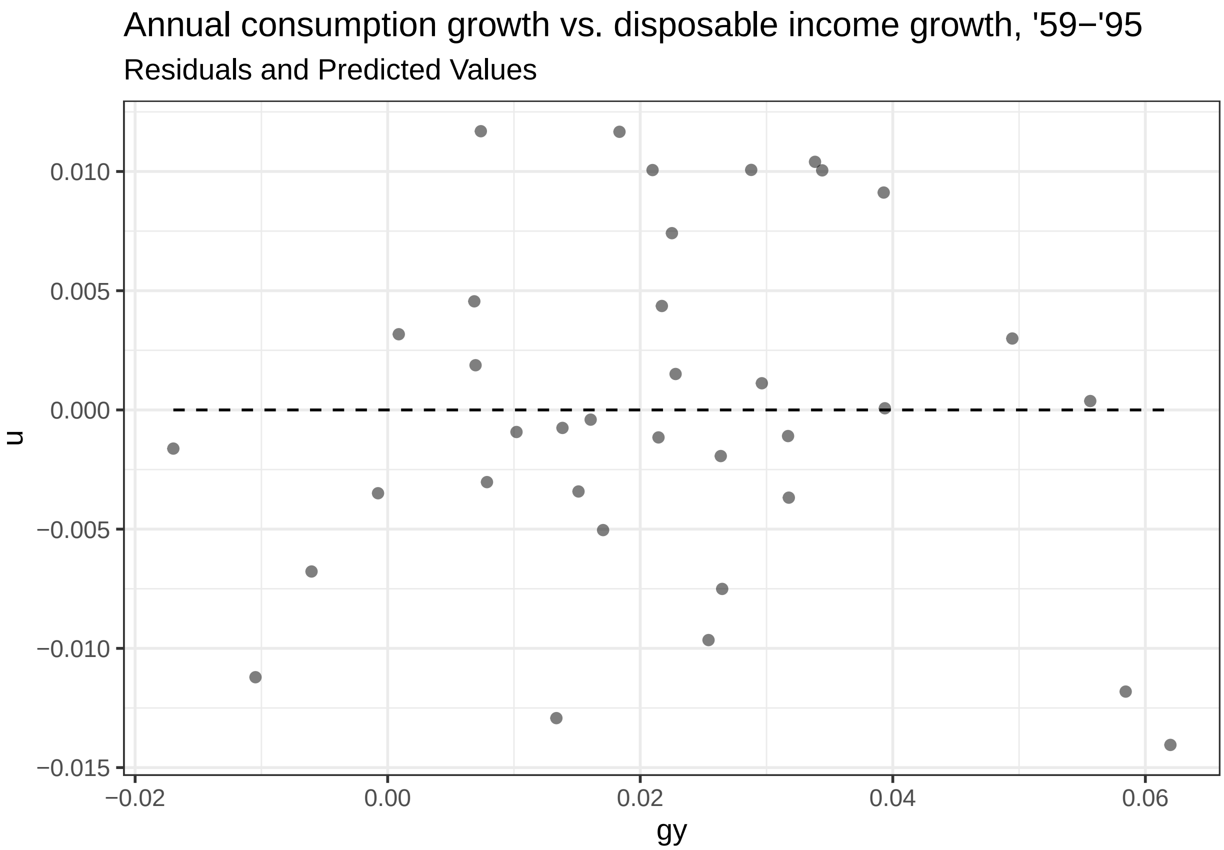
Custom visualizations are great for examining regressions and displaying results for others. Of course, visualizing multiple regression is more difficult than simple linear regression, but these tools should apply to all regression models. [8]

```
ggplot(d) +
    geom_point(aes(x = gy, y = gc), alpha = .5) +
    geom_smooth(aes(x = gy, y = gc), method = 'lm', formula = y ~ x) +
    labs(title = "Annual consumption growth vs. disposable income growth, '59-'95",
         subtitle = "Data and Regression Line")
```

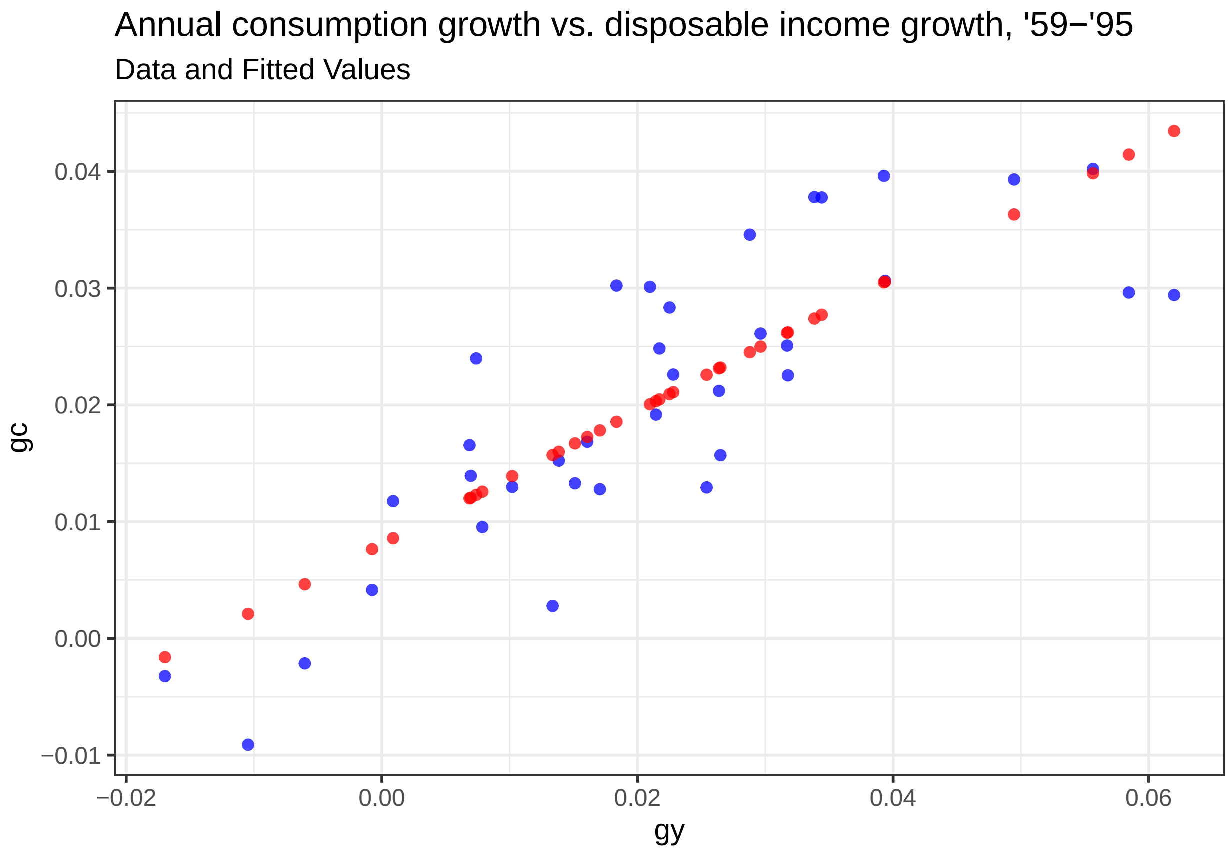
[8] `ggplot2` is built for general data visualization, so it is substantially more powerful, but also more verbose, than Stata's plotting. If you're plotting frequently, you can package these into custom functions, as many packages already do!




```
ggplot(d) +
    geom_point(aes(x = gy, y = u), alpha = .5) +
    geom_line(aes(x = gy, y = 0), linetype = "dashed" ) +
    labs(title = "Annual consumption growth vs. disposable income growth, '59-'95",
         subtitle = "Residuals and Predicted Values")
```




```r
ggplot(d) +
    geom_point(aes(x = gy, y = gc), alpha = .75, color = "blue") +
    geom_point(aes(x = gy, y = yhat), alpha = .75, color = "red") +
    labs(title = "Annual consumption growth vs. disposable income growth, '59-'95",
         subtitle = "Data and Fitted Values")
```



Let's extract the standard errors and confidence intervals from our model. This is one of the major shortcomings of the `summary()` function - we don't see them right away.

```
coef(lm_base)
```

```
(Intercept)          gy
0.008079147 0.570780965
```

```
confint(lm_base, level = .95)
```

```
                  2.5 %     97.5 %
(Intercept) 0.004219672 0.01193862
gy          0.433900187 0.70766174
```

Great work! This exactly matches Stata's output.

4.3 Heteroskedasticity Robustness with `sandwich`, `lmtest`, and `estimatr`

Economists often contend with heteroskedastic standard errors.[9] Standard errors are a somewhat complex topic in R and the workflow for managing them is very different than Stata. I highly recommend reading this blog post by Grant McDermott, I will summarize it below.

A few facts about standard errors:

[9] Technically, calling a model robust means nothing. However, since this seems somewhat standard in economics, and perhaps because of the option in Stata, I use robust to denote models that are calculated with standard errors robust to heteroskedasticity.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
wage	241	0	5.9	3.7	0.5	4.7	25.0	
educ	18	0	12.6	2.8	0.0	12.0	18.0	

1. The computationally-intensive part of modelling is fitting a model, especially with generalized linear models that require the computer to solve (get arbitrarily close to the answer to) complex optimization problems. Standard error calculations are trivial in comparison - they only require a few steps of simple arithmetic for each observation.
2. Fitting finds predicted values. Standard error specifications do not change the fitted values or the regression coefficients.
3. Standard errors affect some, but not all regression evaluation statistics.

In Stata, the syntax breaks down these truths about standard errors. `reg y x` and `reg y x, robust` seem to be two different models. Stata requires the declaration of a certain standard error specification "at estimation time," meaning when the model is fit (the coefficients derived and predicted values found).

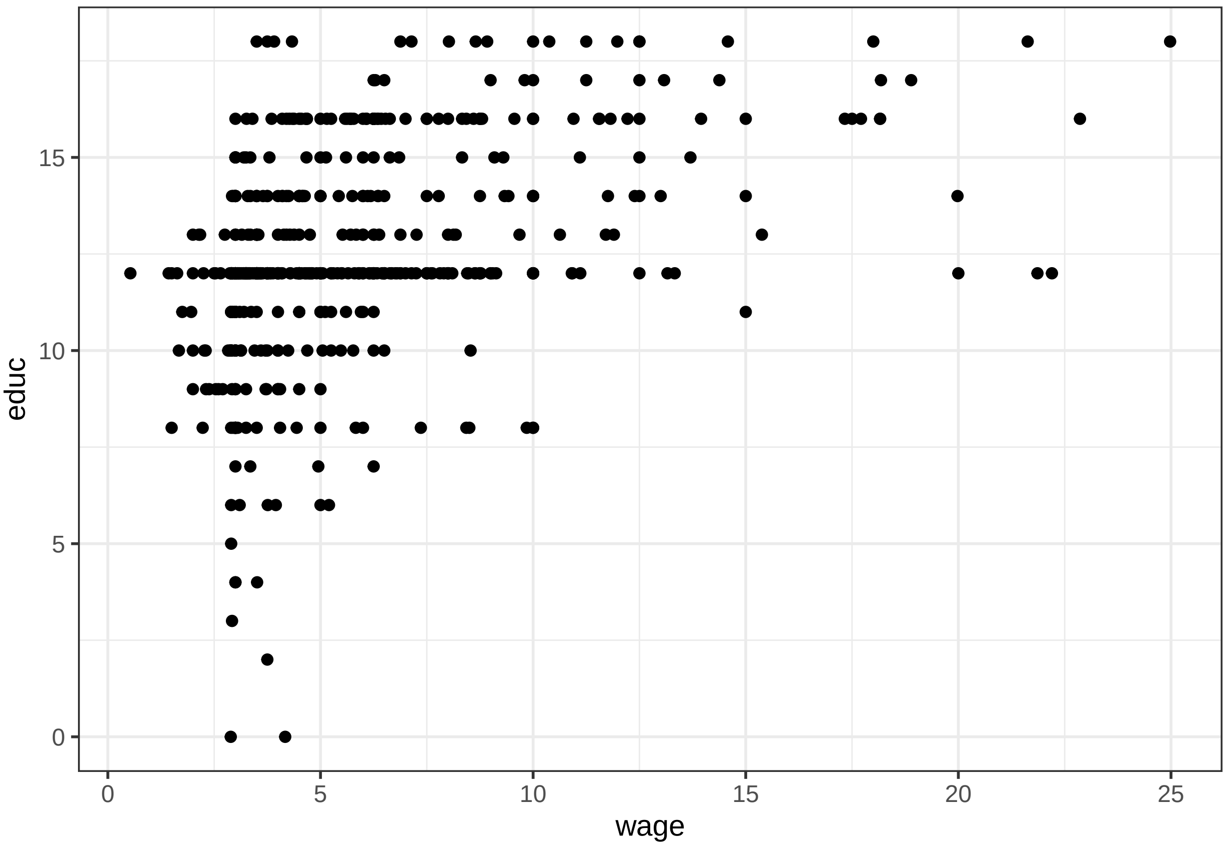
R gives the user the option of modifying standard errors in "post-estimation time," meaning after the model is fit. While the `lm()` model does contain standard errors, they can be easily modified. Why prefer post-estimation? Well, it's complicated. For high-dimensional models or large data sets, computational power is an constraint. It is better to fit the model once, then extract different standard error calculations. It also makes model comparison more intuitive and abides by the coding principle DRY (Don't Repeat Yourself), because the regression function is only called once.

Here is how this all works in practice. These data are worker wage data, each observation is a worker.

```r
d <- wooldridge::wage1
d <- select(d, wage, educ)
datasummary_skim(d, output = "kableExtra")
```

The heteroskedasticity is clear from a simple scatterplot. The data appear in a cone shape, with error variance presumably decreasing as education increases.

```r
ggplot(d) +
    geom_point(aes(x = wage, y = educ))
```



To calculate "HC" standard errors, use the `sandwich` package, which computes robust covariance matrix estimators `vcovHC()` supports common heteroskedasticity-robust standard error calculations, while some other functions support other standard errors. `coeftest()` from the `lmtest` package provides an analysis of coefficients.

The syntax here is critical to remember: `vcov`, in many model analysis functions, takes a standard error calculation method. In newer packages, it may have shortcuts like `iid`, `robust`, `HC`, and `stata`. In any model analysis that is affected by standard errors, post-estimation functions must take a `vcov` specification for heteroskedasticity robustness.

The mathematics of different types are in the documentation.

```
lin_mod <- lm(wage ~ educ, data = d)
coeftest(lin_mod)
```

```

t test of coefficients:

             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.904852   0.684968  -1.321   0.1871
educ         0.541359   0.053248  10.167   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coeftest(lin_mod, vcov = vcovHC(lin_mod, type = "HC1"))
```

```

t test of coefficients:
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.90485    0.72548 -1.2472   0.2129
educ         0.54136    0.06126  8.8371   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

While it does not affect the statistical significance in this case (this model is laughably simplistic), the difference between the t values should indicate the importance of standard error specification.

There is also the option to include heteroskedasticity-robust standard errors from the start, using the package `estimatr`. Unfortunately, the support for this package is sorely lacking, so it does not work in nearly every use case. I use post-estimation specification because it is really the only option for time-series and panel data. Conveniently, the `summary()` output of a `lm_robust()` model is more detailed, clearer, and even more similar to Stata output.

```
lm_rob_mod <- lm_robust(wage ~ educ, data = d)
summary(lm_rob_mod)
```

```
Call:
lm_robust(formula = wage ~ educ, data = d)

Standard error type:  HC2

Coefficients:
            Estimate Std. Error t value  Pr(>|t|) CI Lower CI Upper  DF
(Intercept)  -0.9049     0.7287  -1.242 2.149e-01  -2.3364   0.5267 524
educ          0.5414     0.0615   8.803 1.947e-17   0.4205   0.6622 524

Multiple R-squared:  0.1648 ,   Adjusted R-squared:  0.1632
F-statistic: 77.49 on 1 and 524 DF,  p-value: < 2.2e-16
```

There is one major issue with this package - it automatically applies the `na.action = na.pass` to the internal `lm()` call. Therefore, when there are `NA` values in the relevant variables in the data frame, `predict()` will not work. There are three options to resolve this.

1. If the problem requires predicted values, use some version of `na.omit()` to clean the data frame beforehand. [10]
2. Use the `commarobust()` function to transform an `lm(..., na.action = na.exclude)` variable into a `lm_robust()` object. In most of my test cases, these transformations worked.[11]
3. Build both models, extracting predictions out of the `lm()` model and then analyzing the `lm_robust()` model.

Since `estimatr` plays so nice with `summary()`, a great function for analyzing regressions in development, it may seem easier from a workflow perspective - why use a regression function that requires two different functions for analysis. Hold on for one short subsection on multiple regression before I outline some post-estimation-standard-error-friendly methods for analyzing regressions that are even better than `summary()`!

4.4 Multiple Regression

Like Stata, R computes regression models with matrix algebra, so the computation is already generalized to multiple regression. In other words, the same functions work, they just require partiuclar syntax. The `lm()`

[10] Ensure you aren't omitting all rows with `NA`'s, just rows with `NA`'s in the regressors.
[11] Credit to the authors of `estimatr` for a great Stata pun.

function `formula` argument is in R's formula syntax, which (when an equation is called for) replaces `=` with `~`. Hence, `y ~ x`. For multiple regression, the syntax is: `y ~ x1 + x2 + x3`...

```
d <- wooldridge::big9salary

d <- mutate(d, salaryk = salary/1000)

lm_mul_mod <- lm(salaryk ~ female + prof, data = d)
summary(lm_mul_mod)
```

```
Call:
lm(formula = salaryk ~ female + prof, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-55.167 -14.025  -3.367  10.834  97.633

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   61.261      1.890  32.419   <2e-16 ***
female        -2.513      3.608  -0.696    0.486
prof          32.107      2.172  14.785   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.01 on 569 degrees of freedom
  (214 observations deleted due to missingness)
Multiple R-squared:  0.2945,    Adjusted R-squared:  0.292
F-statistic: 118.8 on 2 and 569 DF,  p-value: < 2.2e-16
```

That is all there is to it. Of course, visualizations and interpretation are more challenging with multiple regressors, but the code is similar. `lm_robust()` syntax is identical to `lm()` syntax.

4.5 Model Analysis with `parameters` and `modelsummary`

Interpreting models is an essential skill of econometrics but attempting to do so with `summary()` is quite the challenge. Its main faults are:

1. No post-estimation adjustments
2. No full parameter statistics
3. No model comparison
4. No extensions, like built-in graphing functions
5. No presentation-ready output, only text in the console

`parameters` and `modelsummary` solve all of these issues. `parameters` will excel in workflow for model development, expressing the differences clearly in text, while `modelsummary` is optimized for presentation.

First, some models, with data of 141 undergraduates at Michigan State University including GPA and demographic information.

```
d <- wooldridge::gpa1
d <- select(d, colGPA, hsGPA, ACT, greek, skipped, age)
datasummary_skim(data = d, output = "kableExtra")
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
colGPA	19	0	3.1	0.4	2.2	3.0	4.0	
hsGPA	16	0	3.4	0.3	2.4	3.4	4.0	
ACT	15	0	24.2	2.8	16.0	24.0	33.0	
greek	2	0	0.3	0.5	0.0	0.0	1.0	
skipped	8	0	1.1	1.1	0.0	1.0	5.0	
age	8	0	20.9	1.3	19.0	21.0	30.0	

```r
lm_fit_1 <- lm(colGPA ~ hsGPA, data = d)
lm_fit_2 <- lm(colGPA ~ ACT, data = d)
lm_fit_m1 <- lm(colGPA ~ hsGPA + ACT + greek, data = d)
lm_fit_m2 <- lm(colGPA ~ hsGPA + ACT + greek + skipped, data = d)
```

`parameters` is part of a developing data analysis workflow called `easystats`. It provides excellent functionality for analysis of parameters of a model, `summary()`'s biggest weakness.

```r
parameters(lm_fit_1)
```

```
Parameter   | Coefficient |   SE |       95% CI | t(139) |      p
-----------------------------------------------------------------
(Intercept) |        1.42 | 0.31 | [0.81, 2.02] |   4.61 | < .001
hsGPA       |        0.48 | 0.09 | [0.30, 0.66] |   5.37 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

That's a nice table, particularly for development. `modelsummary()` displays data for an entire model.[12]

```r
msummary(lm_fit_1,
                 statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                 output = "latex_tabular")
```

	Model 1
(Intercept)	1.415
	SE: 0.307 CI: [0.809, 2.022] t-value: 4.611
hsGPA	0.482
	SE: 0.090 CI: [0.305, 0.660] t-value: 5.371
Num.Obs.	141
R2	0.172
R2 Adj.	0.166
AIC	99.9
BIC	108.8
Log.Lik.	-46.963
F	28.845

[12] The syntax for the `statistic` argument is a bit complicated. It's `glue` syntax, in which brackets surround variable names to pull them into the string. The variables refer to the `broom::tidy()` output. This will make sense after the next chapter. If you're concerned with its verbosity, you'll be pleased to know that it can be set as a global option to avoid repetition - see `modelsummary` documentation for more.

The vertical presentation is not my favorite, but this is still an excellent summary graphic.

These packages truly shine with post-estimation adjustments and model comparison. First, let's compare the original model standard error calculations.

```
parameters(lm_fit_1)
```

```
Parameter   | Coefficient |   SE |       95% CI | t(139) |      p
-----------------------------------------------------------------
(Intercept) |        1.42 | 0.31 | [0.81, 2.02] |   4.61 | < .001
hsGPA       |        0.48 | 0.09 | [0.30, 0.66] |   5.37 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

```
parameters(lm_fit_1, robust = T)
```

```
Parameter   | Coefficient |   SE |       95% CI | t(139) |      p
-----------------------------------------------------------------
(Intercept) |        1.42 | 0.33 | [0.76, 2.07] |   4.29 | < .001
hsGPA       |        0.48 | 0.10 | [0.29, 0.67] |   4.96 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

That is about as clear as R could be about the difference between standard and HC robust standard errors, but all that text might be annoying for a reader. `parameters(lm_fit_1, vcov = vcovHC())` also works, but this syntax is clear and concise.

To present the same model with different standard error calculations, use a vector longer than one for the `vcov` argument of `msummary()`. `msummary()` conveniently notes the different calculations at the bottom.

```
msummary(lm_fit_1, # one model
                 vcov = c("iid", "robust", "stata"), # three standard error specs
                   statistic = "SE: {std.error} t: {statistic}",
                   output = "latex_tabular")
```

	Model 1	Model 2	Model 3
(Intercept)	1.415	1.415	1.415
	SE: 0.307 t: 4.611	SE: 0.330 t: 4.293	SE: 0.324 t: 4.365
hsGPA	0.482	0.482	0.482
	SE: 0.090 t: 5.371	SE: 0.097 t: 4.957	SE: 0.096 t: 5.040
Num.Obs.	141	141	141
R2	0.172	0.172	0.172
R2 Adj.	0.166	0.166	0.166
AIC	99.9	99.9	99.9
BIC	108.8	108.8	108.8
Log.Lik.	-46.963	-46.963	-46.963
F	28.845	24.569	25.405
Std.Errors	IID	Robust	Stata

The next comparison between models is in regressor specification - which variables are used to predict the outcome variable?

For comparison, `modelsummary` accepts a list (of course, possible to declare inline). `parameters` requires two calls. I will not demonstrate because total summaries and visualizations are better with large models. Naming the list helps with clarity in the summary.

```
models <- list(
    GPA_mod = lm_fit_1,
    ACT_mod = lm_fit_2,
    big_mod = lm_fit_m2)
```

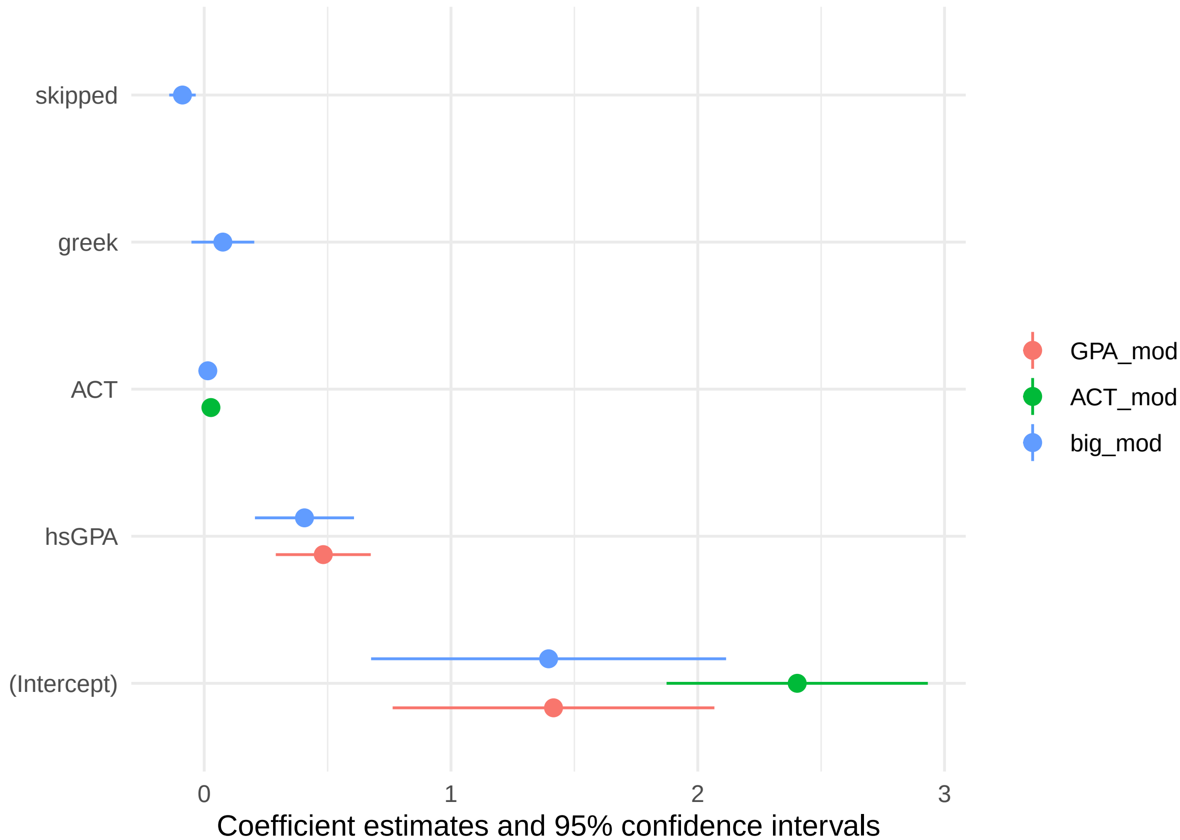
With either `robust = TRUE` or the more specific `vcov` argument, standard error specifications are made in `modelsummary()`.

```
msummary(models,
                    vcov = "robust",
                    statistic = "SE: {std.error} t: {statistic}",
                    output = "latex_tabular")
```

	GPA_mod	ACT_mod	big_mod
(Intercept)	1.415	2.403	1.395
	SE: 0.330 t: 4.293	SE: 0.268 t: 8.975	SE: 0.364 t: 3.836
hsGPA	0.482		0.406
	SE: 0.097 t: 4.957		SE: 0.102 t: 4.001
ACT		0.027	0.015
		SE: 0.011 t: 2.422	SE: 0.011 t: 1.298
greek			0.075
			SE: 0.064 t: 1.172
skipped			-0.088
			SE: 0.027 t: -3.248
Num.Obs.	141	141	141
R2	0.172	0.043	0.242
R2 Adj.	0.166	0.036	0.220
AIC	99.9	120.4	93.4
BIC	108.8	129.2	111.1
Log.Lik.	-46.963	-57.177	-40.692
F	24.569	5.864	11.071
Std.Errors	Robust	Robust	Robust

Numeric values are always ripe for visualization, and model parameters are no different. `modelplot()` compares parameters visually in one line of code.

```
modelplot(models, vcov = "robust") # That is the list of models (could also be written inline)
```



These settings are getting tedious to re-type every time. Setting global options avoids that problem. Unfortunately, `modelsummary` does not yet support a global option for `statistic` or `vcov` parameters.

```
options(modelsummary_factory_default = "latex_tabular")
```

A few final notes on `msummary`. The `ouput` argument is only necessary for my LaTeX compiler; the default works in general. `msummary()` has built-in functionality to add different table features. In addition, it supports most of the common table-editing frameworks in R, including my favorite `gt`. That means that the table is totally customizable, so adding a title, subtitle, caption, highlighting the variable of interest, etc. is all straightforward for final presentation.

Likewise, `modelplot` creates a `ggplot2` object, so it supports total customization.

4.6 Extracting Tidy Data with `broom`

R excels in data manipulation. While the `summary()` output is great for understanding the model of interest, it does not make the data easy to manipulate.[13] `broom` underlies `parameters` and `modelsummary` by doing the extraction for each package's presentation of data from the model.

The package `broom` provides the solution. It has three major functions. These functions take a model (and sometimes the data used to build the model) and return summary data, like `summary()`, except in tidy data

[13] Why might you want to manipulate regression results? Here's a standard case. Consider the Gapminder dataset with four variables, `country`, `year`, `gdp`, and `life_expectancy`. Each observation is a country in a given year, from 1900-2020, for example, and has the country's GDP and life expectancy in that year. R makes it very easy to nest this larger data frame by `country` with `nest()` so we now have a data frame for each country. Now, after mapping `lm()` over each country with a model `life_expectancy ~ GDP`, we want to compare the r-squareds of these regressions. In other words, this exercise was to compare how well gross domestic product predicts life expectancy in different countries. Now, we want to graph this r-squared, maybe against `gdp` or `life_expectancy` or on a map. To do that, we need a function that extracts the r-squared value from a model.

frames as opposed to text. That way, that data can be extracted for analysis and computation, or printed much more neatly.

- `broom::tidy()` returns the coefficients and relevant statistics
- `broom::glance()` returns the anova regression statistics
- `broom::augment()` returns statistics for each observation, including fitted and residual values[14]

Let's see them in action.

```
d <- wooldridge::consump
lm_mod <- lm(gc ~ gy, data = d, na.action = na.exclude)
tidy(lm_mod)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)  0.00808   0.00190      4.25 1.55e- 4
2 gy           0.571     0.0674       8.47 6.75e-10
```

```
glance(lm_mod)
```

```
# A tibble: 1 x 12
  r.squared adj.r.squared   sigma statistic  p.value    df logLik   AIC   BIC
      <dbl>         <dbl>   <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
1     0.679         0.669 0.00727      71.8 6.75e-10     1   127. -248. -244.
# ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
augment(lm_mod, data = d)
```

```
# A tibble: 37 x 30
    year    i3   inf rdisp rnondc rserv    pop      y rcons     c    r3    lc
   <int> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
 1  1959  3.41 0.700 1530.   606.  687. 177830  8604. 1294. 7275.  2.71  8.89
 2  1960  2.93 1.70  1565.   615.  717. 180671  8664. 1333. 7377.  1.23  8.91
 3  1961  2.38 1     1616.   627.  746. 183691  8796. 1373. 7476.  1.38  8.92
 4  1962  2.78 1     1694.   646.  783. 186538  9080. 1430. 7665.  1.78  8.94
 5  1963  3.16 1.30  1756.   660   819. 189242  9276. 1479. 7814.  1.86  8.96
 6  1964  3.55 1.30  1882.   692.  868. 191889  9807. 1561. 8134.  2.25  9.00
 7  1965  3.95 1.60  2002.   729.  915. 194303 10305. 1644. 8460.  2.35  9.04
 8  1966  4.88 2.90  2107.   769.  961  196560 10717. 1730. 8802.  1.98  9.08
 9  1967  4.32 3.10  2198.   781. 1008. 198712 11063. 1789  9003.  1.22  9.11
10  1968  5.34 4.20  2298.   817. 1060. 200706 11451. 1876. 9349.  1.14  9.14
# ... with 27 more rows, and 18 more variables: ly <dbl>, gc <dbl>, gy <dbl>,
#   gc_1 <dbl>, gy_1 <dbl>, r3_1 <dbl>, lc_ly <dbl>, lc_ly_1 <dbl>, gc_2 <dbl>,
#   gy_2 <dbl>, r3_2 <dbl>, lc_ly_2 <dbl>, .fitted <dbl>, .resid <dbl>,
#   .hat <dbl>, .sigma <dbl>, .cooksd <dbl>, .std.resid <dbl>
```

Now, this data can be extracted as values easily.

$

[14] Note that `augment()` has some weird behavior with `NA` functions. With `na.action = na.exclude`, you must also pass the original data. With the default `na.action = na.omit`, `augment()` returns a data frame of different length.

```r
r_sq <- glance(lm_mod)$r.squared
# tricky syntax: glance() is a data frame, so $r.squared extracts a variable, as df$var
```

$

```r
print(r_sq)
```

```
[1] 0.6786799
```

`broom` is so popular that it is replicated by packages with regression functions that create objects `broom`'s main functions do not understand. In short, call `broom` functions to extract values from any model object.

5 Regression Extras

This chapter encompasses everything that supports regression analysis including statistical tests, some standard variable manipulations, the extraction of marginal effects, and the creation of interaction effects. When statistically appropriate, these methods should generalize to multiple regression, time series and panel data-derived models, generalized linear models, and more advanced model objects in R.

5.1 F Test

Before we conduct an F-test, let's fit robust models.

```
d <- wooldridge::wage2

d <- filter(d, !(is.na(meduc) | is.na(feduc)))

lm_mod <- lm(lwage ~ educ, data = d)
mr_mod <- lm(lwage ~ educ + IQ + KWW, data = d)
```

F-tests in R are slightly more complicated than in Stata. The F-test is either performed as a joint test on an entire model, or performed as a comparison between models, so the first-level abstraction is different than in Stata.[15]

The overall/joint F-test is performed in `lm()` and `lm_robust()` so the use cases of this function are rather limited.

```
waldtest(mr_mod, vcov = vcovHC, test = "F")
```

```
Wald test

Model 1: lwage ~ educ + IQ + KWW
Model 2: lwage ~ 1
  Res.Df Df      F    Pr(>F)
1    718
2    721 -3 46.199 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

F-tests are often required to evaluate the addition of new variables to the model. Unfortunately, R's syntax is verbose because it requires declaring models with and without the variables of interest.

Here, let's examine whether `meduc` and `feduc` should be added to the model.

```
mr_mod_2 <- lm(lwage ~ educ + IQ + KWW + meduc + feduc, data = d)
waldtest(mr_mod, mr_mod_2, vcov = vcovHC, test = "F")
```

```
Wald test

Model 1: lwage ~ educ + IQ + KWW
Model 2: lwage ~ educ + IQ + KWW + meduc + feduc
  Res.Df Df      F Pr(>F)
```

[15] In Stata, the abstraction, as evidenced by the syntax, is the null hypothesis that the variables are equal to zero. In R, the primary abstraction is the comparison of models with and without the relevant variables.

```
1    718
2    716  2 3.2135 0.0408 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Of course, you could always do the same thing in-line.

```
waldtest(
    lm(lwage ~ educ + IQ + KWW, data = d),
    lm(lwage ~ educ + IQ + KWW + meduc + feduc, data = d),
    vcov = vcovHC,
    test = "F")
```

```
Wald test

Model 1: lwage ~ educ + IQ + KWW
Model 2: lwage ~ educ + IQ + KWW + meduc + feduc
  Res.Df Df      F Pr(>F)
1    718
2    716  2 3.2135 0.0408 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This is returns the same calculations as Stata.[16]

5.2 Dummy Variables with `dplyr` and `fastDummies`

Dummy variables, formally Bernoulli variables, take values 1 or 0. They are used to include categorical variables in regression or to simplify regression interpretation.[17]

`dplyr` is necessary to generate dummies from continuous variables, like the `gen ... replace ... if` workflow in Stata. `if_else()` with `mutate()` creates the new dummy.[18]

The data are from Botswana's 1988 Demographic and Health Survey. There are 4361 women with 27 variables of information about each.

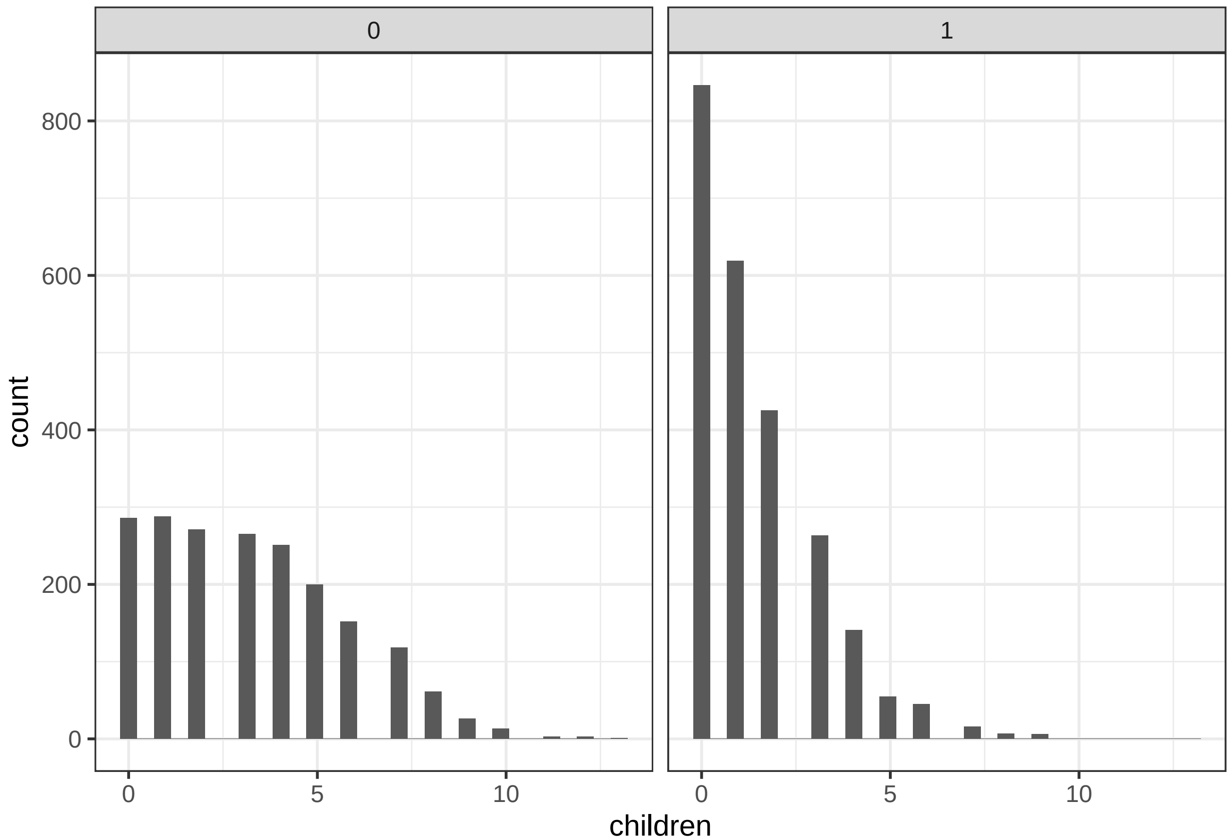
```
d <- wooldridge::fertil2
d <- mutate(d, grade7 = if_else(educ > 6, 1, 0))

ggplot(d) +
    geom_histogram(aes(x = children)) +
    facet_wrap(~grade7)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

[16] Here are the equivalent commands in Stata. First, `reg lwage educ IQ KWW meduc feduc, robust` generates the model. Then, either `testparm meduc feduc` or the more general `test (meduc = 0) (feduc = 0)` conducts the F-test.

[17] Instead of regression on `yrs_educ`, we could create a dummy `hs` that takes 1 if the observation graduated high-school and 0 if not, and the same for college. While less precise, the interpretations of the coefficients on each might be more clear than the coefficient of `yrs_educ`.

[18] `if_else()` is a vectorized `if {} else{}` logic that conducts a test on each value of `educ`. In the rows in which the test `educ > 6` returns `TRUE`, `grade7` gets `1`. In the rows in which the test returns `FALSE`, `grade7` gets `0`. The `base::ifelse()` is equivalent, but can return different types, which `if_else()` will not.



The more common use case for dummy variables is with categorical variables that take a small number of unique values. In this case, I examine seasonality with dummy variables for months. R has a `factor` data type, in which integers are associated with each unique value of the variable. This data type is complex and sometimes difficult to work with, but transforming our character vector categorical variables into factors makes modelling straightforward.

This data is 108 observations of 25 variables on number of employment claims over time in or out of the Anderson enterprise zone.

```
d <- wooldridge::ezanders
d <- select(d, -c(jan, feb, mar, may, apr, jun, jul, aug, sep, oct, nov, dec))
# removed existing dummies
lm_mod <- lm(uclms ~ factor(month), data = d)
parameters(lm_mod)
```

```
Parameter     | Coefficient |      SE |               95% CI | t(95) |      p
----------------------------------------------------------------------------
(Intercept)   |     7085.67 | 1656.83 | [ 3796.44, 10374.89] |  4.28 | < .001
month [AUG]   |     -658.89 | 2343.11 | [-5310.56,  3992.78] | -0.28 | 0.779
month [DEC]   |      346.96 | 2415.23 | [-4447.87,  5141.79] |  0.14 | 0.886
month [FEB]   |     2243.78 | 2343.11 | [-2407.89,  6895.45] |  0.96 | 0.341
month [JAN]   |     2409.22 | 2343.11 | [-2242.45,  7060.89] |  1.03 | 0.306
month [JULY]  |    -1415.22 | 2343.11 | [-6066.89,  3236.45] | -0.60 | 0.547
month [JUNE]  |    -1154.78 | 2343.11 | [-5806.45,  3496.89] | -0.49 | 0.623
month [MAR]   |     2152.67 | 2343.11 | [-2499.00,  6804.34] |  0.92 | 0.361
month [MAY]   |     -826.89 | 2343.11 | [-5478.56,  3824.78] | -0.35 | 0.725
month [NOV]   |    -2075.44 | 2343.11 | [-6727.11,  2576.22] | -0.89 | 0.378
```

```
month [OCT]  |    -2778.44 | 2343.11 | [-7430.11,  1873.22] | -1.19 | 0.239
month [SEPT] |    -2606.22 | 2343.11 | [-7257.89,  2045.45] | -1.11 | 0.269

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

R has quietly eliminated April because of collinearity. This solution works, but we may prefer to eliminate the intercept. The syntax for a no-intercept formula is `y ~ -1 + x1 + x2 + factor(x3) ....` `y ~ 0 + x1 + x2 + factor(x3)` also works.

```r
lm_mod_no_int <- lm(uclms ~ -1 + factor(month), data = d)
parameters(lm_mod_no_int)
```

```
Parameter    | Coefficient |      SE |              95% CI | t(95) |      p
---------------------------------------------------------------------------
month [APR]  |     7085.67 | 1656.83 | [3796.44, 10374.89] |  4.28 | < .001
month [AUG]  |     6426.78 | 1656.83 | [3137.55,  9716.00] |  3.88 | < .001
month [DEC]  |     7432.62 | 1757.34 | [3943.87, 10921.38] |  4.23 | < .001
month [FEB]  |     9329.44 | 1656.83 | [6040.22, 12618.67] |  5.63 | < .001
month [JAN]  |     9494.89 | 1656.83 | [6205.66, 12784.12] |  5.73 | < .001
month [JULY] |     5670.44 | 1656.83 | [2381.22,  8959.67] |  3.42 | < .001
month [JUNE] |     5930.89 | 1656.83 | [2641.66,  9220.12] |  3.58 | < .001
month [MAR]  |     9238.33 | 1656.83 | [5949.11, 12527.56] |  5.58 | < .001
month [MAY]  |     6258.78 | 1656.83 | [2969.55,  9548.00] |  3.78 | < .001
month [NOV]  |     5010.22 | 1656.83 | [1721.00,  8299.45] |  3.02 | 0.003
month [OCT]  |     4307.22 | 1656.83 | [1018.00,  7596.45] |  2.60 | 0.011
month [SEPT] |     4479.44 | 1656.83 | [1190.22,  7768.67] |  2.70 | 0.008

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

To create variables in the data object, use `fastDummies::dummy_cols()`. Writing them for regression, though, is horribly tedious, so instantiate them in the data frame only when absolutely necessary.

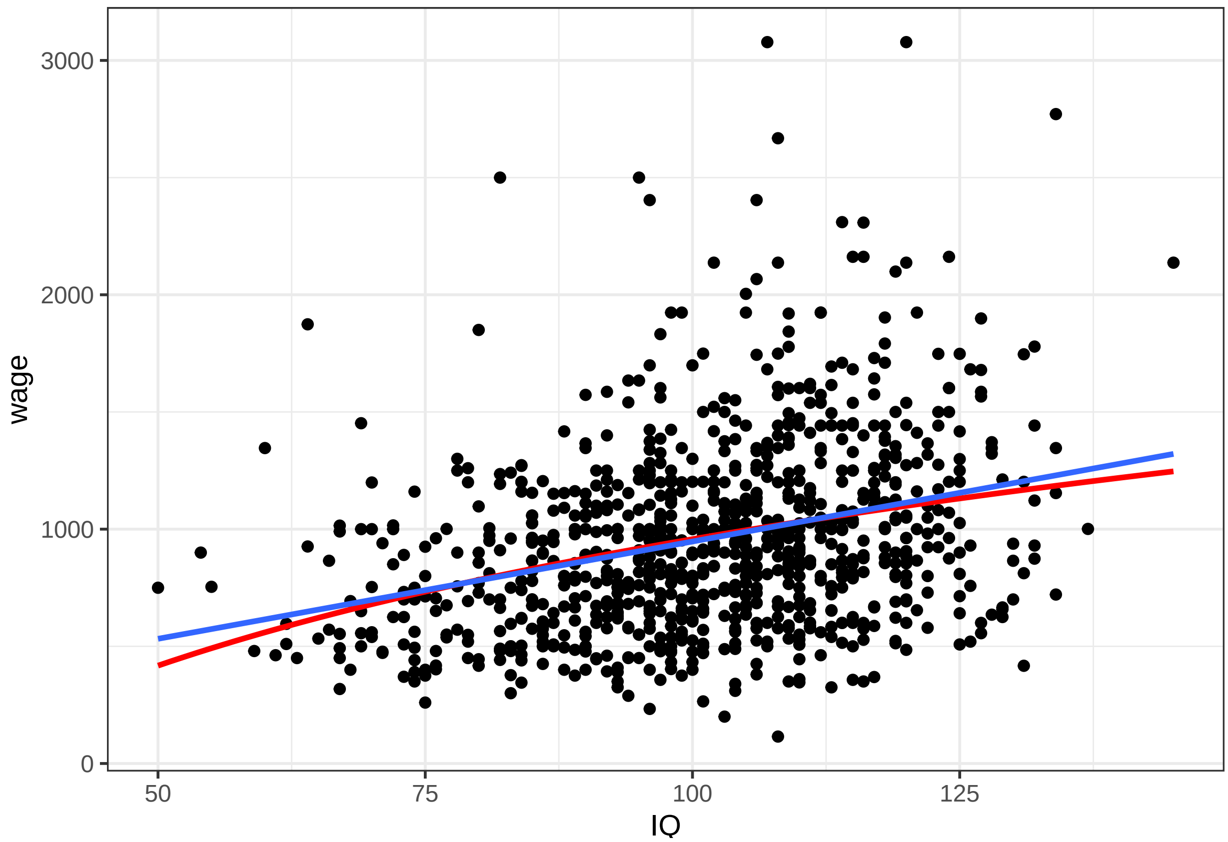
```r
d <- dummy_cols(d, select_columns = "month")
print(colnames(d))
```

```
 [1] "month"      "uclms"      "ez"         "year"       "y81"
 [6] "y82"        "y83"        "y84"        "y85"        "y86"
[11] "y87"        "y88"        "luclms"     "month_APR"  "month_AUG"
[16] "month_DEC"  "month_FEB"  "month_JAN"  "month_JULY" "month_JUNE"
[21] "month_MAR"  "month_MAY"  "month_NOV"  "month_OCT"  "month_SEPT"
```

5.3 Linear Transformations

R has many many functions to transform variables. The `log()` function will be of use here - use it with `mutate(..., log_x = log(x)` or in a formula `y ~ log(x)`.

```
d <- wooldridge::wage2

ggplot(d) +
    geom_point(aes(x = IQ, y = wage)) +
    geom_smooth(aes(x = IQ, y = wage), method = 'lm', formula = y ~ log(x),
                        color = "red", se = F) +
    geom_smooth(aes(x = IQ, y = wage), method = 'lm', formula = y ~ x, se = F)
```



Another transformation is polynomial regression. You may manually create square and cube variables with `dplyr`, and then generate a model:

```
d <- wooldridge::hprice3

d <- mutate(d, agesq = age^2)

lm_poly_manual <- lm(lprice ~ age + agesq, data = d)
parameters(lm_poly_manual)
```

```
Parameter   | Coefficient |       SE |         95% CI | t(318) |      p
-----------------------------------------------------------------------
(Intercept) |       11.58 |     0.02 | [11.53, 11.63] | 468.14 | < .001
age         |       -0.02 | 1.50e-03 | [-0.02, -0.02] | -12.90 | < .001
agesq       |    1.03e-04 | 1.02e-05 | [ 0.00,  0.00] |  10.15 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

But, graphing this will be a bit of a pain. Instead, use R's built in `poly()` function, which makes this syntactically easier and generalizes to the `k`th power.

Because of some complicated math, `raw = TRUE` is required when using poly in a model.[19]

```
lm_poly <- lm(lprice ~ poly(age, 2, raw = TRUE), data = d)
parameters(lm_poly)
```

```
Parameter         | Coefficient |       SE |         95% CI | t(318) |      p
----------------------------------------------------------------------------
(Intercept)       |       11.58 |     0.02 | [11.53, 11.63] | 468.14 | < .001
age [1st degree] |       -0.02 | 1.50e-03 | [-0.02, -0.02] | -12.90 | < .001
age [2nd degree] |    1.03e-04 | 1.02e-05 | [ 0.00,  0.00] |  10.15 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

The following code produces an identical model. `^` is a formula operator, so you must wrap the polynomial term with `I()`; using `^` alone in a formula will always fail. For polynomials with a small degree this is a reasonable combination of legibility and verbosity:

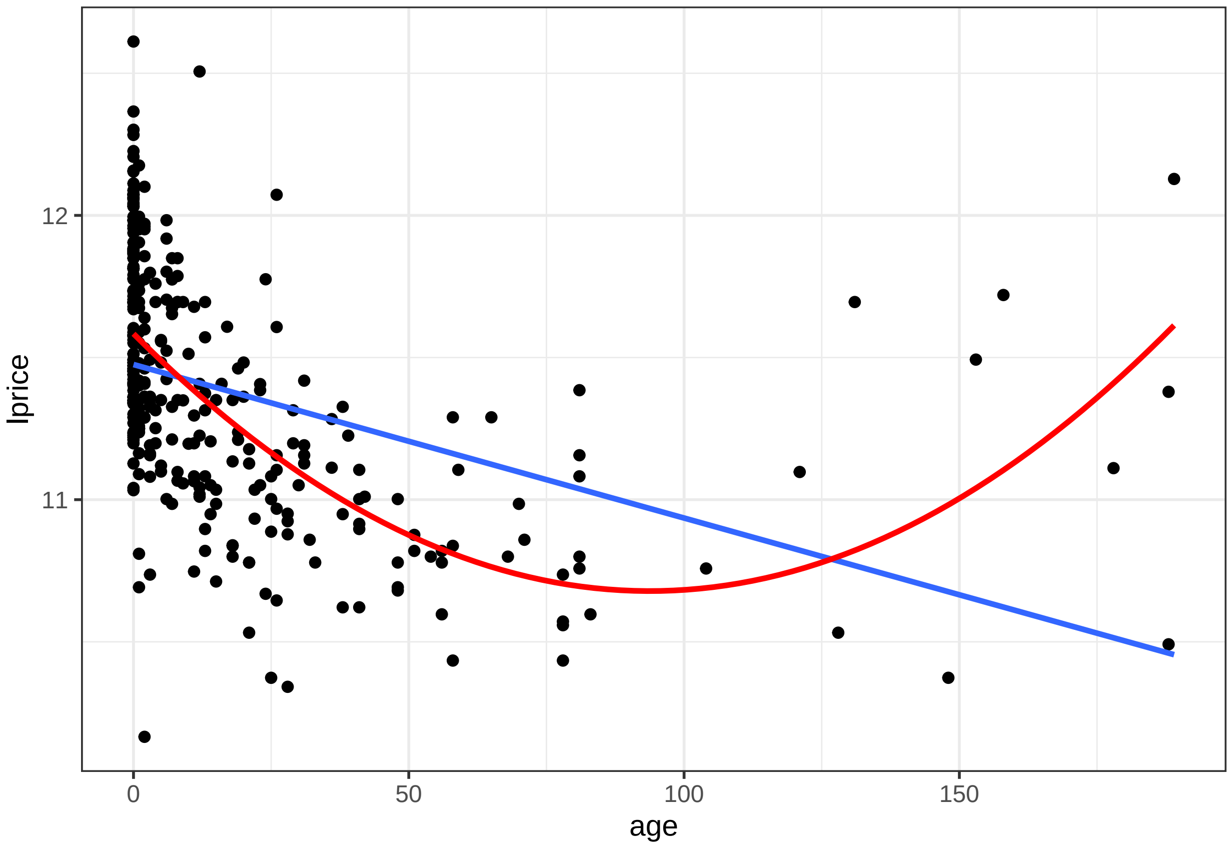
```
lm_poly_verbose <- lm(lprice ~ age + I(age^2), data = d)
parameters(lm_poly_verbose)
```

```
Parameter    | Coefficient |       SE |         95% CI | t(318) |      p
-----------------------------------------------------------------------
(Intercept) |       11.58 |     0.02 | [11.53, 11.63] | 468.14 | < .001
age         |       -0.02 | 1.50e-03 | [-0.02, -0.02] | -12.90 | < .001
age^2       |    1.03e-04 | 1.02e-05 | [ 0.00,  0.00] |  10.15 | < .001

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

Likewise, this makes graphing much more straightfoward.

```
ggplot(d) +
    geom_point(aes(x = age, y = lprice)) +
    geom_smooth(aes(x = age, y = lprice), method = 'lm', formula = y ~ x, se = F) +
    geom_smooth(aes(x = age, y = lprice), method = 'lm', formula = y ~ poly(x, 2),
                color = "red", se = F)
```

[19] Read this StackOverflow thread for more information.



5.4 Marginal Effects with `margins`

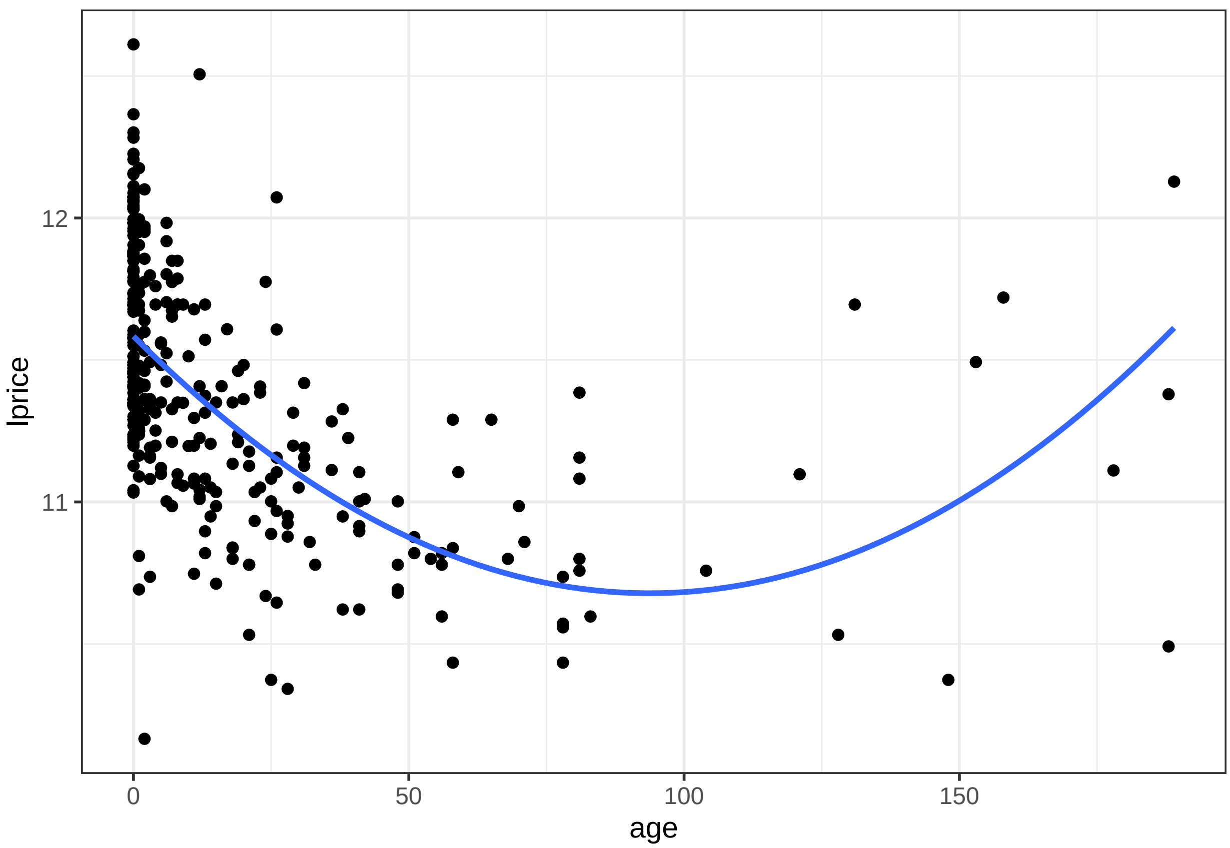
With a simple linear model the marginal effect is the coefficient. When `x` appears in multiple regression terms, however, the derivative is no longer just the coefficient of `x`. The `margins` package is a workhorse.

```
d <- wooldridge::hprice3

lm_mod <- lm(lprice ~ age + I(age^2), data = d)
summary(lm_mod)
```

```

Call:
lm(formula = lprice ~ age + I(age^2), data = d)

Residuals:
     Min       1Q   Median       3Q      Max
-1.37951 -0.23434 -0.03549  0.24868  1.13948

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.158e+01  2.474e-02  468.14   <2e-16 ***
age         -1.931e-02  1.497e-03  -12.90   <2e-16 ***
I(age^2)     1.030e-04  1.015e-05   10.15   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.3499 on 318 degrees of freedom
Multiple R-squared:  0.3662,	Adjusted R-squared:  0.3622
F-statistic: 91.87 on 2 and 318 DF,  p-value: < 2.2e-16
```

```
ggplot(d) +
    geom_point(aes(x = age, y = lprice)) +
    geom_smooth(aes(x = age, y = lprice), method = 'lm', formula = y ~ poly(x, 2),
                               se = F)
```



Intuitively, the derivative of the curve is not constant over `x`; in truth, it is a function of `x`, but the computer cannot do algebra. `margins()` calculates it at any value of any regressor.[20] This function returns a data frame that can be saved or printed.

```
margins(lm_mod, at = list(age = c(20, 120)))
```

```
Average marginal effects at specified values


lm(formula = lprice ~ age + I(age^2), data = d)


 at(age)       age
      20 -0.015189
     120  0.005409
```

[20]Note this function's strange syntax. The `at` parameter requires a named list. The syntax for multiple regression would be `margins(mr_mod, at = list(x1 = c(x1_val1, x1_val2, ...), x2 = c(x2_val1, x2_val2, ...))`. Without an `at` parameter, the function returns the average marginal effect, which is not of much use.

5.5 Interaction Effects

Interaction effects between regressors are intuitive. The syntax is as expected, but there are tricks to have in your toolkit.

```
d <- wooldridge::wage2

lm_mod <- lm(lwage ~ educ + age + educ*age, data = d)
parameters(lm_mod)
```

```
Parameter   | Coefficient |       SE |         95% CI | t(931) |      p
----------------------------------------------------------------------
(Intercept) |        6.49 |     0.91 | [ 4.71, 8.27]  |   7.16 | < .001
educ        |       -0.03 |     0.07 | [-0.17, 0.10]  |  -0.52 | 0.606
age         |       -0.02 |     0.03 | [-0.07, 0.04]  |  -0.56 | 0.574
educ * age  |    2.82e-03 | 1.99e-03 | [ 0.00, 0.01]  |   1.42 | 0.157

Uncertainty intervals (equal-tailed) and p values (two-tailed) computed using a
  Wald t-distribution approximation.
```

R displays the interaction effect between `x1` and `x2` as `x1:x2`.

So, adding the `x1*x2` term creates the interaction effect between the two. You can also use the following syntactical shortcuts:

```
d <- wooldridge::wage2

lm_full <- lm(lwage ~ educ*age, data = d)
lm_interaction_only <- lm(lwage ~ educ:age, data = d)
lm_nested_interaction <- lm(lwage ~ educ/age, data = d)
```

Here are the coefficients of each model:

```
coef(lm_full)
```

```
 (Intercept)         educ          age     educ:age
 6.489349204 -0.034771048 -0.015103689  0.002823177
```

```
coef(lm_interaction_only)
```

```
(Intercept)    educ:age
5.989650012 0.001772014
```

```
coef(lm_nested_interaction)
```

```
(Intercept)        educ    educ:age
5.981036268 0.002527894 0.001714920
```

Without `x1` or `x2` independently in the model:

- `x1*x2` includes `x1`, `x2`, and their interaction `x1:x2`
- `x1:x2`includes only their interaction `x1:x2`
- `x1/x2` includes `x1` and the interaction `x1:x2`

6 Developments on Least Squares Regression

Important models in economics are developments on least squares regression built to counter endogeneity and leverage additionally information about the data, like the natural ordering of time series data. I cover differences-in-differences, autoregression, and fixed effects models. In their simplest form, these models are based on the math of linear regression, so `lm()` calls are theoretically sufficient (but not optimal) to fit them.

This presents a challenge: there are often multiple competing implementations. In most cases, I show multiple methods, a simple method for a simpler model, and the gold-standard, workhorse package with the power for much more advanced regression models.[21] I support learning the more advanced implementation in general because syntax and verbosity are small prices to pay for power. In addition, the design choices made by the more advanced packages are often better.[22]

6.1 Difference-in-Differences

Difference in differences is a statistical technique used to mimic an experiment using observational data, by studying the differential effect of a treatment on a 'treatment group' versus a 'control group' in a natural experiment. It is increasingly popular in economics and worth learning thoroughly. Unfortunately, it is split in R - simple dif-in-dif models can be built with `lm()` calls and standard tools, but more complex models require `did`, a package built for dif-in-dif analysis.

6.1.1 Structural Break in `base`

Of course, since dif-in-dif is built on linear regression, in a simple case of a 'structural break' for all observations, we can use an interaction effect.

This data is from a Massachusetts community in 1978 and 1981. In 1979, the construction of a garbage incinerator was announced. Concern was immediately voiced over the impact on housing prices.

I create a dummy variable representing houses within 3 miles of the incinerator. I also use `d81`, a dummy variable for 1981 observations, after the incinerator has been announced. The idea is that housing prices should only be affected by the incinerator after it was announced in 1979.

```
d <- wooldridge::hprice3
d <- select(d, year, age, agesq, price, lprice, y81, dist)
datasummary_skim(d)
```

[21] This section is the most difficult and consequently the most worthwhile. Learning these tools is challenging. This is generally the case in R: to develop a new skill for one case, like basic AR(1), you may have to learn an entire package that can do so much more. The benefit, obviously, is that your skills cover swaths of new territory as you learn the best method for a general problem. An obvious, simple `tidyverse` example: working with date and time data. Using `base` functions, POSIX data structures, and string manipulation might solve your specific date and time data manipulation problem in 15 minutes. But, spend 20 minutes learning `lubridate` and nearly all date and time manipulation problems are trivial.

[22] Of course, sometimes it is worse. By far the most important aspect of design is support for integration into tidy and other frameworks.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
year	2	0	1979.3	1.5	1978.0	1978.0	1981.0	
age	61	0	18.0	32.6	0.0	4.0	189.0	
agesq	61	0	1381.6	4801.8	0.0	16.0	35721.0	
price	200	0	96100.7	43223.7	26000.0	85900.0	300000.0	
lprice	200	0	11.4	0.4	10.2	11.4	12.6	
y81	2	0	0.4	0.5	0.0	0.0	1.0	
dist	176	0	20715.6	8508.2	5000.0	19900.0	40000.0	

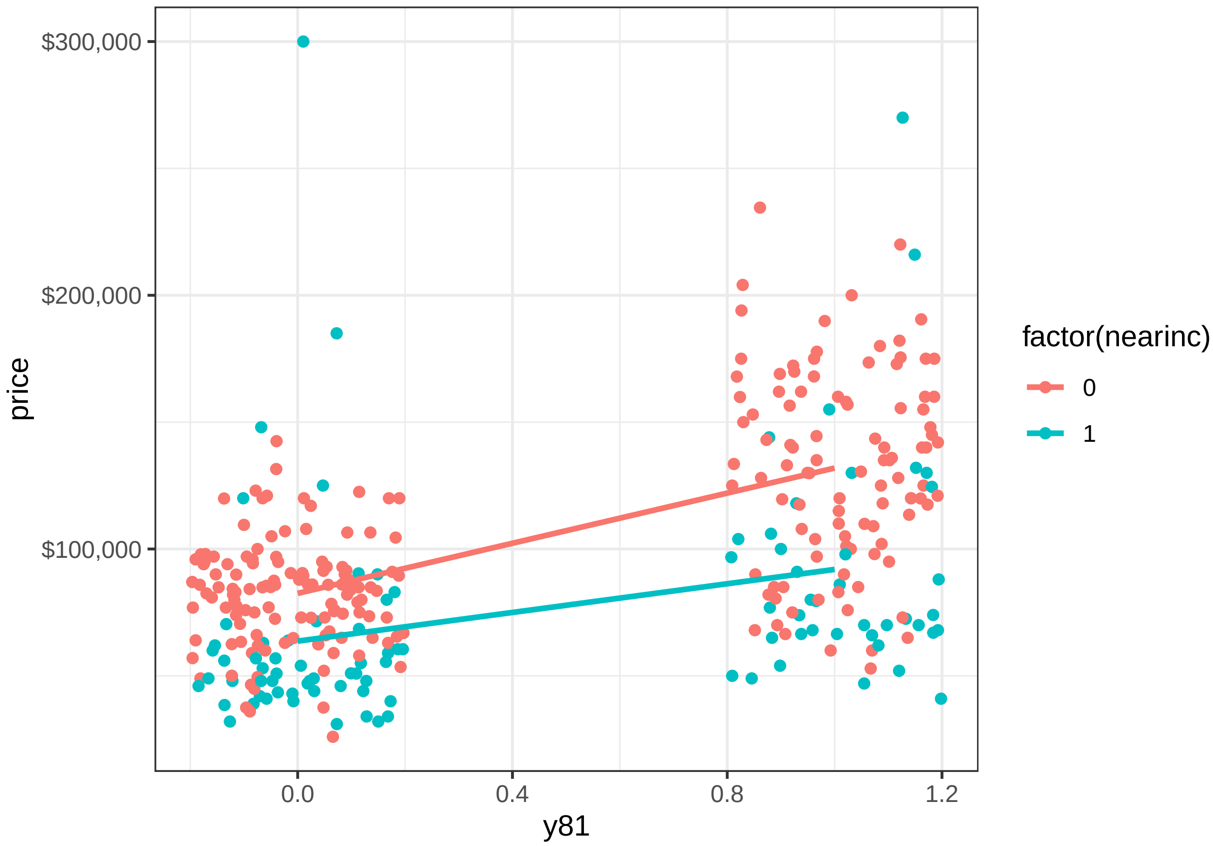
```
d <- mutate(d, nearinc = if_else(dist < (5280*3), 1, 0))
did_lm <- lm(price ~ nearinc*y81, data = d)
summary(did_lm)
```

```

Call:
lm(formula = price ~ nearinc * y81, data = d)

Residuals:
   Min     1Q Median     3Q    Max 
-79002 -19693  -3517  13383 236307 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)    82517       3142  26.263  < 2e-16 ***
nearinc       -18824       5617  -3.351 0.000902 ***
y81            49385       4666  10.583  < 2e-16 ***
nearinc:y81   -21132       8592  -2.460 0.014443 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 34850 on 317 degrees of freedom
Multiple R-squared:  0.3562,	Adjusted R-squared:  0.3501 
F-statistic: 58.46 on 3 and 317 DF,  p-value: < 2.2e-16
```

Since dif-in-dif is such an important technique, it is worth creating a standard dif-in-dif graph in ggplot.[23]

```
ggplot(d) +
    geom_jitter(aes(x = y81, y = price, color = factor(nearinc)), width = .2, height = 0) +
    geom_smooth(aes(x = y81, y = price, color = factor(nearinc)), method = 'lm', formula = y ~ x, se = 
    scale_y_continuous(labels = scales::dollar_format())
```

[23]There is one non-standard aspect of this graph. I used `geom_jitter()` instead of `geom_point()` - since the x variable is a dummy, the points overplot (plot over one another) when they are all directly on the 1978 or 1981 vertical lines. `geom_jitter()` with width but not height jittering creates random variation in x, but none in y, so you can see the points more clearly.



6.1.2 Complex Difference-in-Differences with `did`

For more complex dif-in-dif modelling, explore the `did` package, which has more functionality for:

- More than two time periods
- Variation in treatment timing (i.e., units can become treated at different points in time)
- Treatment effect heterogeneity (i.e, the effect of participating in the treatment can vary across units and exhibit potentially complex dynamics, selection into treatment, or time effects)
- The parallel trends assumption holds only after conditioning on covariates

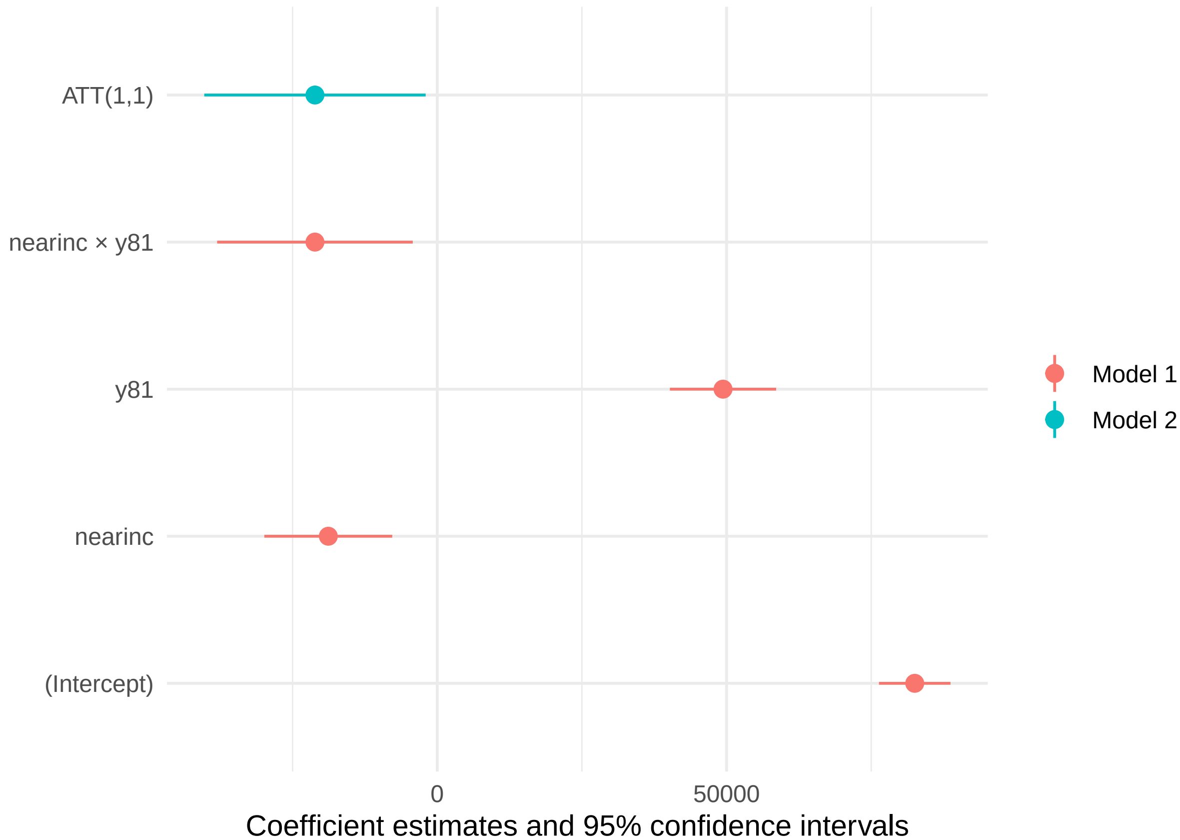
This is an example of non-standard package design.[24] Instead of using a formula to isolate which variables serve which purpose in the model, they are taken as arguments.

```
did_fit <- did::att_gt(yname = "price", tname = "y81", gname = "nearinc", panel = FALSE, data = d)
```

```
No pre-treatment periods to test
```

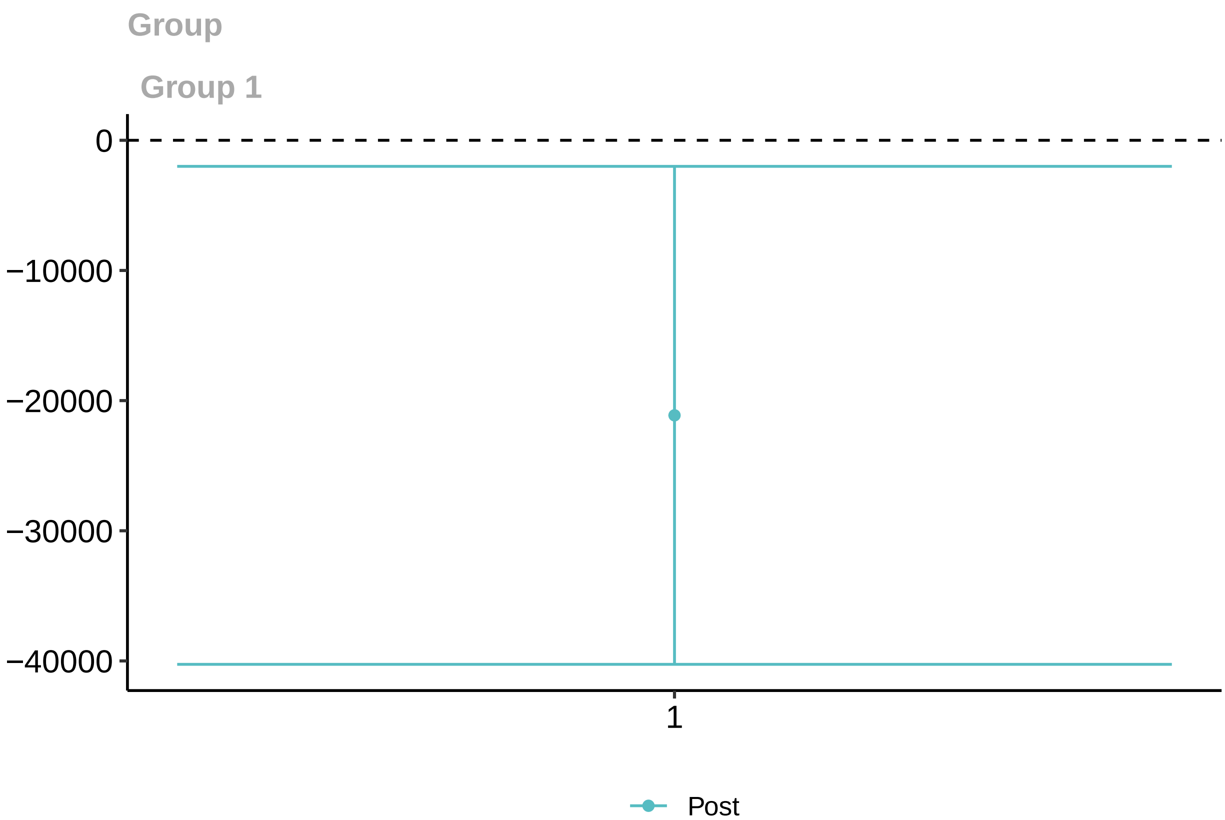
```
modelplot(list(did_lm, did_fit))
```

[24] Every dif-in-dif package is designed this way, for some peculiar reason.



This is a perfect case study of when using a more complex package is counter-productive. With multiple groups or differing treatment times, `base` will be difficult and `did`'s extra features will be useful. `did` also provides a nice built-in plotting function that works well for multiple groups. For this application, it seems a bit silly.

```
ggdid(did_fit)
```



6.2 Autoregression Models (Time Series Data)

Time series data is characterized by naturally ordered observations occurring over time.

There are different methods for working with time-series data in R and little community agreement over which are the best.[25] There are some extraneous functions you should be aware of, like `dplyr::lag()` and `dplyr::lead()` that are useful for working with time-series data by hand. Of course, mutating a lagged variable and then regressing on it can be done with only `dplyr` and `lm()`.

6.2.1 Simple AR(1) with `base`

```
d <- wooldridge::traffic2

d <- mutate(d,
                        yrmnth = seq(
                            make_yearmonth(year = 1981, month = 1),
                            length.out = nrow(d),
                            by = 1),
                            month = month(yrmnth, label = TRUE, abbr = TRUE))

d <- select(d, yrmnth, month, totacc, fatacc, spdlaw, beltlaw)
datasummary_skim(d, output = "kableExtra")
```

[25] The `base` functions for time-series are built to work on vectors, not dataframes, which is syntactically tedious and feels hacked together, nowhere near as elegant as `tsset`. Moreover, models are `ts()` objects with limited graphing options only available through packages. I genuinely prefer Stata to the `base` functionality, I think. `tsibble` and `fable` seem to me to be the best options, but the community is split.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
totacc	108	0	42831.3	4608.3	32699.0	42863.5	52971.0	
fatacc	79	0	377.9	48.5	266.0	370.0	500.0	
spdlaw	2	0	0.3	0.5	0.0	0.0	1.0	
beltlaw	2	0	0.4	0.5	0.0	0.0	1.0	

There are a few steps here. R is not as smart as Stata when it comes to creating time variables. `tsibble` provides a family of `year*()` functions like `yearmonth()`, which create special date vectors so that R "knows" the interval and timeframe in the dataset. This dataset does not have a great date variable to convert, so instead I use `make_yearmonth()` combined with `seq()` to manually construct a vector that corresponds to the correct number of months because it is the same length as the data (which isn't missing rows!) and starts in January 1980, same as the data. This is not an ideal workflow.

Then, I use `lubridate::month()` to extract the month from that data.

```
lm_fit <- lm(totacc ~ lag(totacc) + month + beltlaw, data = d)
lm_fit_noconstant <- lm(totacc ~ 0 + lag(totacc) + month + beltlaw, data = d)

msummary(list("AR(1)" = lm_fit, "AR(1) w/o Constant" = lm_fit_noconstant),
                 vcov = vcovHC,
                 statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                 output = "latex_tabular")
```

	AR(1)	AR(1) w/o Constant
(Intercept)	15891.574	
	SE: 4086.029 CI: [7777.531, 24005.617] t-value: 3.889	
lag(totacc)	0.596	0.596
	SE: 0.103 CI: [0.391, 0.802] t-value: 5.772	SE: 0.103 CI: [0.391, 0.802] t-value: 5.772
month.L	4144.367	
	SE: 710.232 CI: [2733.987, 5554.748] t-value: 5.835	
month.Q	-1466.321	
	SE: 719.486 CI: [-2895.078, -37.565] t-value: -2.038	
month.C	2350.128	
	SE: 668.790 CI: [1022.043, 3678.212] t-value: 3.514	
month^4	-1889.868	
	SE: 671.133 CI: [-3222.605, -557.132] t-value: -2.816	
month^5	1394.838	
	SE: 622.206 CI: [159.260, 2630.417] t-value: 2.242	
month^6	773.938	
	SE: 577.224 CI: [-372.315, 1920.190] t-value: 1.341	
month^7	-1339.458	
	SE: 546.124 CI: [-2423.952, -254.963] t-value: -2.453	
month^8	3180.244	
	SE: 534.725 CI: [2118.387, 4242.101] t-value: 5.947	
month^9	-1241.441	
	SE: 485.137 CI: [-2204.828, -278.054] t-value: -2.559	
month^10	2497.165	
	SE: 481.533 CI: [1540.936, 3453.395] t-value: 5.186	
month^11	-872.757	
	SE: 397.615 CI: [-1662.340, -83.173] t-value: -2.195	
beltlaw	3122.739	3122.739
	SE: 832.169 CI: [1470.216, 4775.263] t-value: 3.753	SE: 832.169 CI: [1470.216, 4775.263] t-value: 3.753
monthJan		11528.175
		SE: 4537.144 CI: [2518.306, 20538.044] t-value: 2.541
monthFeb		13395.161
		SE: 3826.593 CI: [5796.305, 20994.017] t-value: 3.501
monthMar		19419.352
		SE: 3743.751 CI: [11985.006, 26853.699] t-value: 5.18
monthApr		13905.610
		SE: 4258.343 CI: [5449.384, 22361.835] t-value: 3.265
monthMay		16084.464
		SE: 3952.946 CI: [8234.696, 23934.232] t-value: 4.069
monthJun		15291.222
		SE: 4016.306 CI: [7315.635, 23266.809] t-value: 3.807
monthJul		16382.777
		SE: 3995.841 CI: [8447.829, 24317.725] t-value: 4.100
monthAug		16669.494
		SE: 4121.074 CI: [8485.857, 24853.131] t-value: 4.045
monthSep		15796.399
		SE: 4174.465 CI: [7506.739, 24086.060] t-value: 3.784
monthOct		17902.830
		SE: 4182.460 CI: [9597.294, 26208.366] t-value: 4.280
monthNov		16378.557
		SE: 4233.711 CI: [7971.246, 24785.867] t-value: 3.869
monthDec		17944.843
		SE: 4389.717 CI: [9227.736, 26661.951] t-value: 4.088
Num.Obs.	107	107
R2	0.907	0.999
R2 Adj.	0.894	0.999
AIC	1884.9	1884.9
BIC	1925.0	1925.0

As you can tell, I fitted two models, with and without a constant - for some reason, `lm_robust()` gets annoying when it must automatically omit a factor. That's the basic AR(1) model, using `lm_robust()` and `dplyr::lag()` to input data, similar to using `reg` in Stata.[26]

6.2.2 ARIMA with tsibble and fable

While the `base` method is convenient, it does not unleash the full power of time-series econometrics. To do that, we'll need `tsibble` to create special data frames and `fable` for tidy `ARIMA` modelling.

Like Stata's workflow, the first step is to convert our raw data frame into a `tsibble`, the tidy data object that stores information about the time-series data.

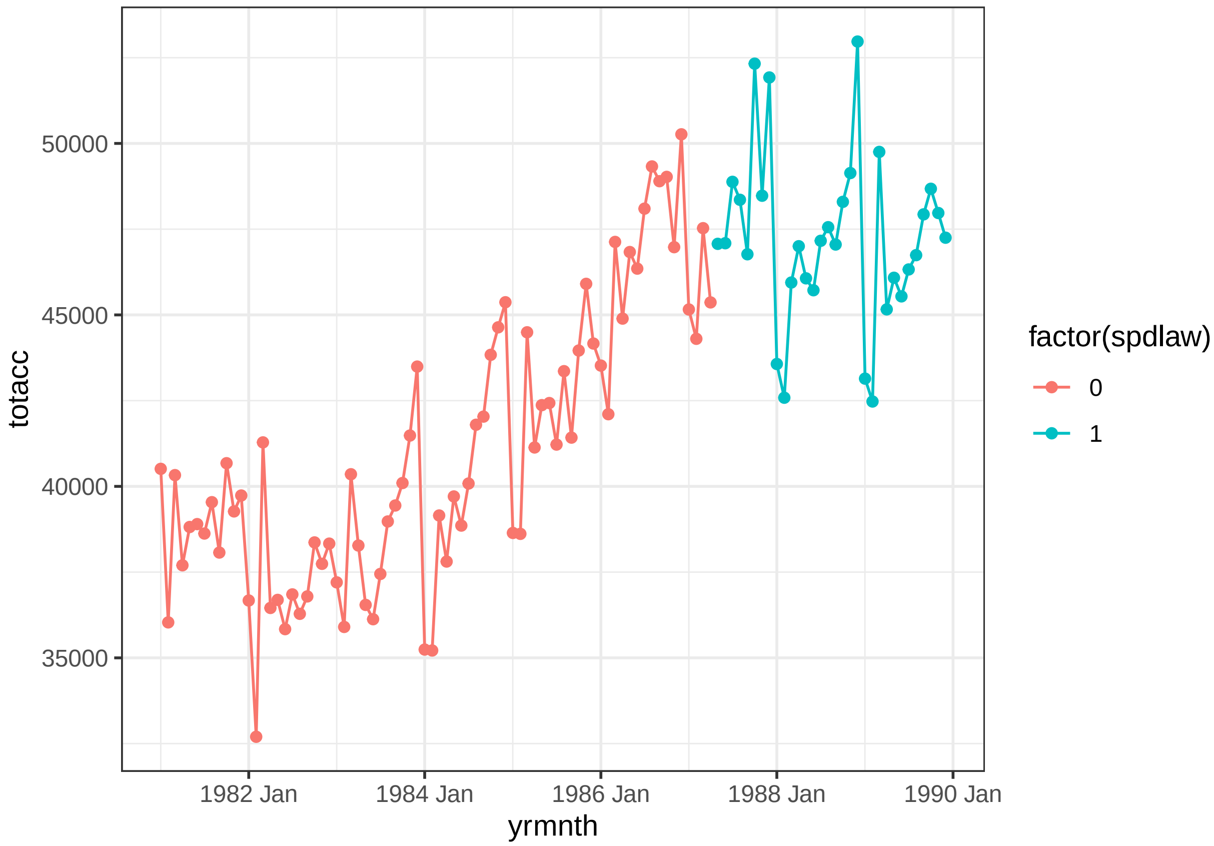
```
d <- wooldridge::traffic2

d <- mutate(d,
                        yrmnth = seq(
                            make_yearmonth(year = 1981, month = 1),
                            length.out = nrow(d),
                            by = 1))

d <- select(d, yrmnth, totacc, fatacc, spdlaw, beltlaw)
d <- na.omit(d)
d <- as_tsibble(d, index = yrmnth)
```

Next, I use `as_tsibble()` with the `index` parameter set to `yrmnth`, the new date variable, to construct the `tsibble` data frame.

Here's a basic time series graph.

```
ggplot(d, aes(x = yrmnth)) +
    geom_point(aes(y = totacc, color = factor(spdlaw))) +
    geom_line(aes(y = totacc, color = factor(spdlaw)))
```

[26]The exact command in Stata would be `reg totacc l.totacc i.month beltlaw, robust`.



The syntax of `fable` is tidy, reflecting the `tidyverse`, rather than `base`. In addition to using the pipe to model, `summary()` returns an error - you must use `report()`.

```
ts_fit2 <- d %>% model(ARIMA(totacc ~ pdq(1, 0, 0) + PDQ(0, 0, 1) + beltlaw))
report(ts_fit2)
```

```
Series: totacc
Model: LM w/ ARIMA(1,0,0)(0,0,1)[12] errors

Coefficients:
        ar1    sma1    beltlaw   intercept
      0.433  0.5407  5877.5370  40264.1623
s.e.  0.091  0.0822   948.0243    665.9052

sigma^2 estimated as 4249222:  log likelihood=-977.55
AIC=1965.09   AICc=1965.68   BIC=1978.51
```

```
ts_fit3 <- d %>% model(TSLM(totacc ~ lag(totacc) + season() + beltlaw))

msummary(ts_fit3,
                 statistic = "SE: {std.error}, t-value: {statistic}", # had issues with confidence inte
                 gof_omit = "_", # printing issues with LaTeX
                 output = "kableExtra")
```

	Model 1
(Intercept)	11528.175
	SE: 3283.510, t-value: 3.511
lag(totacc)	0.596
	SE: 0.077, t-value: 7.745
seasonyear2	1866.986
	SE: 826.733, t-value: 2.258
seasonyear3	7891.177
	SE: 887.115, t-value: 8.895
seasonyear4	2377.435
	SE: 740.822, t-value: 3.209
seasonyear5	4556.289
	SE: 790.441, t-value: 5.764
seasonyear6	3763.047
	SE: 771.705, t-value: 4.876
seasonyear7	4854.602
	SE: 781.052, t-value: 6.215
seasonyear8	5141.319
	SE: 760.569, t-value: 6.760
seasonyear9	4268.225
	SE: 746.494, t-value: 5.718
seasonyear10	6374.655
	SE: 752.738, t-value: 8.469
seasonyear11	4850.382
	SE: 733.739, t-value: 6.611
seasonyear12	6416.668
	SE: 735.513, t-value: 8.724
beltlaw	3122.739
	SE: 648.093, t-value: 4.818
AIC	1581.3
BIC	1621.4
.model	TSLM(totacc ~lag(totacc) + season + beltlaw)
sigma2	2276215.218
AICc	1586.539
CV	2656512.389
rank	14.000

6.3 Fixed Effects Models with `fixest`

Fixed effects models are an absolute workhorse in economics and also happen to be computationally intensive. Fortunately, recent developments in a relatively new package, `fixest`, have cut runtimes significantly.

```
d <- wooldridge::wagepan
d <- select(d, nr, lwage, educ, black, married, union, hisp, year)
datasummary_skim(d, output = "gt")
```

```
Warning in datasummary_skim_numeric(data, output = output, fmt = fmt, histogram
= histogram, : The histogram argument is only supported for (a) output types
"default", "html", or "kableExtra"; (b) writing to file paths with extensions
".html", ".jpg", or ".png"; and (c) Rmarkdown or knitr documents compiled to PDF
or HTML. Use `histogram=FALSE` to silence this warning.
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max
nr	545	0	5262.1	3496.1	13.0	4569.0	12548.0
lwage	3631	0	1.6	0.5	-3.6	1.7	4.1
educ	13	0	11.8	1.7	3.0	12.0	16.0
black	2	0	0.1	0.3	0.0	0.0	1.0
married	2	0	0.4	0.5	0.0	0.0	1.0
union	2	0	0.2	0.4	0.0	0.0	1.0
hisp	2	0	0.2	0.4	0.0	0.0	1.0
year	8	0	1983.5	2.3	1980.0	1983.5	1987.0

The data must be defined as a panel.

```
d <- panel(data = d, panel.id = c("nr", "year"))
```

`panel.id` has some strange syntax - it can be a formula or a vector, but in short, it takes the cross-sectional variable then the time variable.

I fit two models for comparison, one pooled OLS model with clustered standard errors, another with fixed effects.

```
lm_fit <- feols(lwage ~ married + union + educ + black + factor(year), cluster = "nr", data = d)
felm_fit <- feols(lwage ~ married + union + factor(year) | nr, data = d)
```

`feols()` uses `lm()` under the hood, so it is functionally equivalent. However, displaying clustered standard errors is a bit of a challenge with `lm()`, so I use `feols()` for the pooled OLS regression first, with clustered standard errors. Note that `nr` is the id variable for each worker. `feols()` conveniently automatically clusters the standard errors by the first fixed effect.

There are a few viewing options. Of course, `msummary()` works. `fixest` also includes `etable()`, which you may prefer.

```
etable(lm_fit)
```

```
                              lm_fit
Dependent Var.:                lwage
```

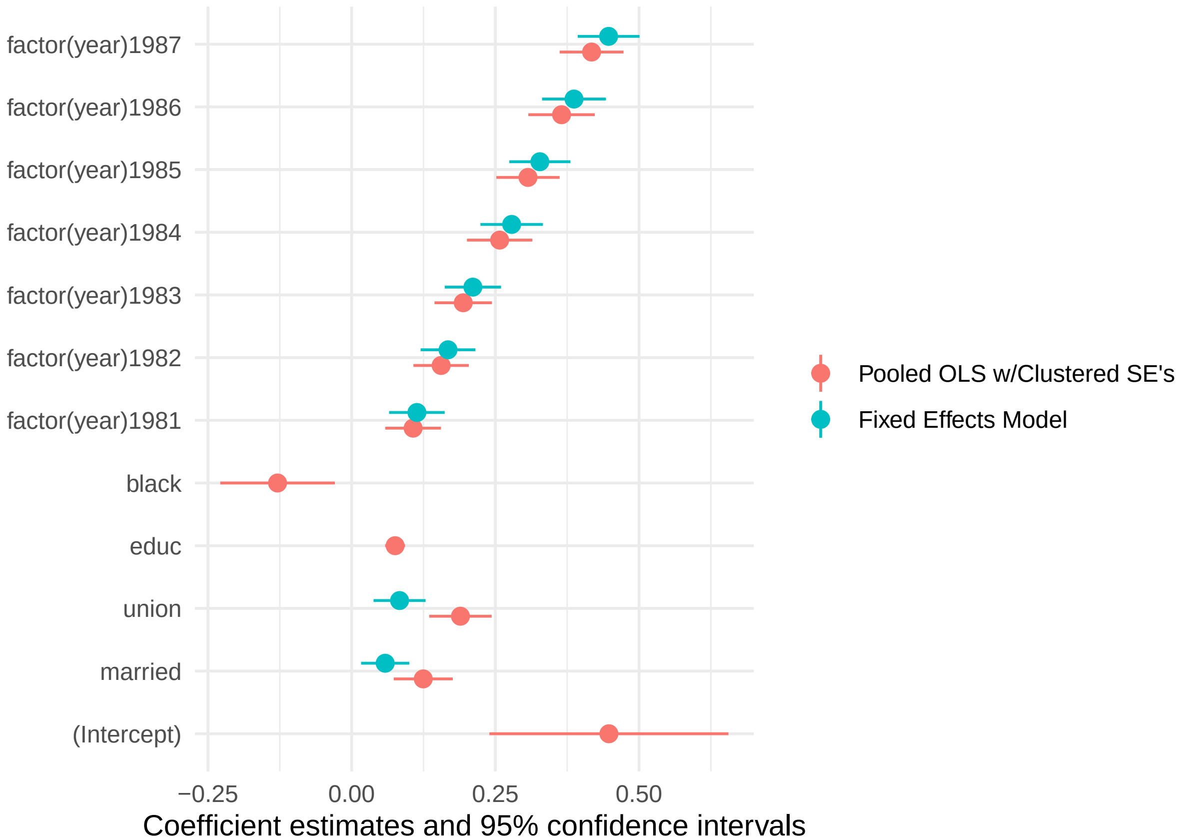
```
(Intercept)       0.4476*** (0.1059)
married           0.1246*** (0.0262)
union             0.1893*** (0.0276)
educ              0.0756*** (0.0088)
black              -0.1289* (0.0507)
factor(year)1981 0.1069*** (0.0247)
factor(year)1982 0.1557*** (0.0245)
factor(year)1983 0.1941*** (0.0255)
factor(year)1984 0.2575*** (0.0290)
factor(year)1985 0.3068*** (0.0280)
factor(year)1986 0.3652*** (0.0294)
factor(year)1987 0.4174*** (0.0283)
________________ __________________
S.E.: Clustered              by: nr
Observations                  4,360
R2                          0.18186
Adj. R2                     0.17979
```

```
etable(felm_fit)
```

```
                           felm_fit
Dependent Var.:               lwage

married            0.0583** (0.0213)
union             0.0834*** (0.0231)
factor(year)1981 0.1135*** (0.0246)
factor(year)1982 0.1677*** (0.0243)
factor(year)1983 0.2109*** (0.0250)
factor(year)1984 0.2784*** (0.0277)
factor(year)1985 0.3275*** (0.0271)
factor(year)1986 0.3868*** (0.0283)
factor(year)1987 0.4470*** (0.0274)
Fixed-Effects:   ------------------
nr                              Yes
________________ __________________
S.E.: Clustered              by: nr
Observations                  4,360
R2                          0.61551
Within R2                   0.16891
```

```
modelplot(list(`Pooled OLS w/Clustered SE's` = lm_fit, `Fixed Effects Model` = felm_fit))
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
mrdrte	90	0	6.9	3.7	0.8	6.2	20.3	
exec	13	0	1.3	3.8	0.0	0.0	34.0	
unem	62	0	5.9	1.7	2.2	5.8	12.0	



Extracting and analyzing the fixed effects coefficients is sometimes valuable. `fixest` makes this easy with `fixef()`. These data are state-wide crime data.

```
d <- wooldridge::murder
d <- filter(d, state != "DC")
d <- select(d, state, mrdrte, exec, unem)
datasummary_skim(d, output = "kableExtra")
```

```
felm_fit <- feols(mrdrte ~ exec + unem | state, data = d)
etable(felm_fit)
```

```
                          felm_fit
Dependent Var.:             mrdrte

exec              -0.0961 (0.0705)
unem              -0.1202 (0.1410)
Fixed-Effects:    ----------------
state                          Yes
```

```
---------------- ----------------
S.E.: Clustered        by: state
Observations                 150
R2                       0.93585
Within R2                0.05052
```

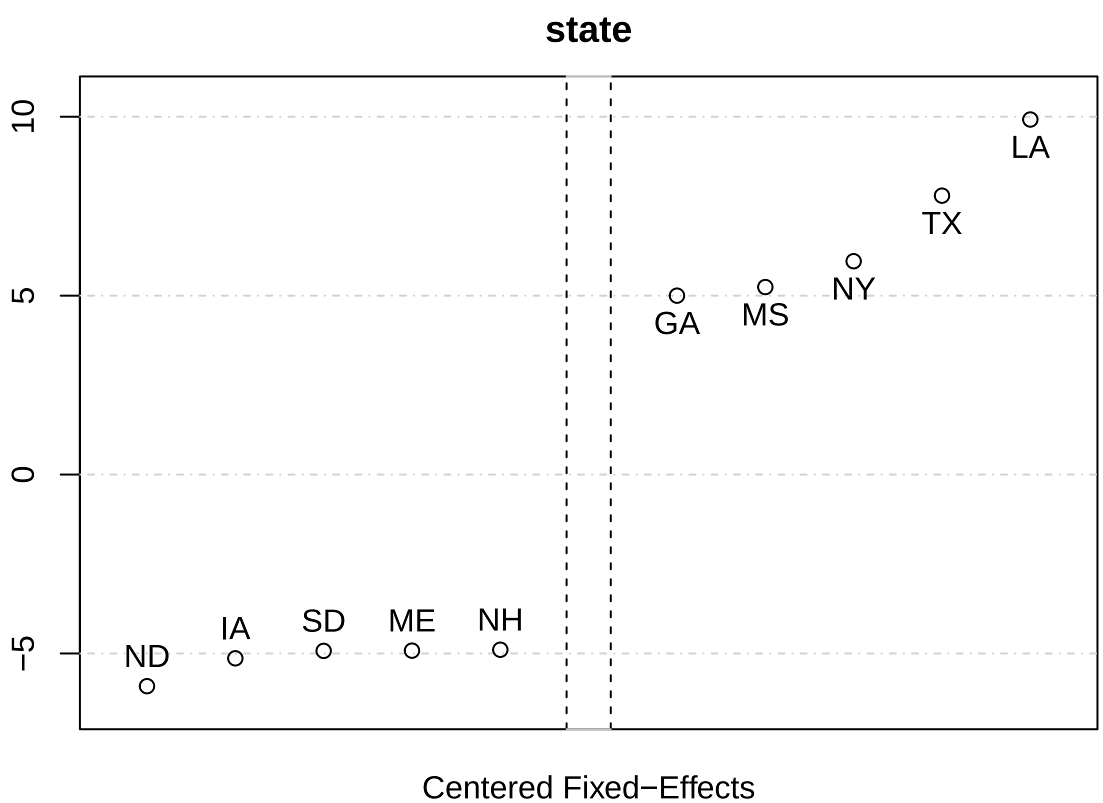
```r
state_fixed_effects <- fixef(felm_fit)
```

The output object of `fixef()` has great functionality.

```r
summary(state_fixed_effects)
```

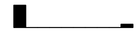
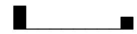
```
Fixed_effects coefficients
Number of fixed-effects for variable state is 50.
    Mean = 7.78 Variance = 14.7

COEFFICIENTS:
  state:   AK    AL    AR    AZ    CA
         9.88 12.01 10.34 8.739 12.76 ... 45 remaining
```

```r
plot(state_fixed_effects)
```



6.4 Instrumental Variable Regression with `ivreg`

Instrumentation is another technique to counter endogeneity, particularly when multiple regression cannot account for omitted variables because they cannot be measured or when there is simultaneous causality between the regressor and outcome variables.

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
educ	10	0	13.5	2.2	9.0	12.0	18.0	
age	11	0	33.1	3.1	28.0	33.0	38.0	
feduc	19	21	10.2	3.3	0.0	10.0	18.0	
meduc	20	8	10.7	2.8	0.0	12.0	18.0	
black	2	0	0.1	0.3	0.0	0.0	1.0	
south	2	0	0.3	0.5	0.0	0.0	1.0	
wage	449	0	957.9	404.4	115.0	905.0	3078.0	
KWW	42	0	35.7	7.6	12.0	37.0	56.0	
IQ	80	0	101.3	15.1	50.0	102.0	145.0	
married	2	0	0.9	0.3	0.0	1.0	1.0	
lwage	449	0	6.8	0.4	4.7	6.8	8.0	

Instrumental variable regression has great implementation in R in a number of settings.[27]

This data is from paper written by David Card. It is a standard wage dataset.

```
d <- wooldridge::wage2
d <- select(d, educ, age, feduc, meduc, black, south, wage, KWW, IQ, married, lwage)
datasummary_skim(d, output = "kableExtra")
```

In this example, `nearc2` is an instrument for `educ`.

There are two syntax options in `ivreg` for the formula.

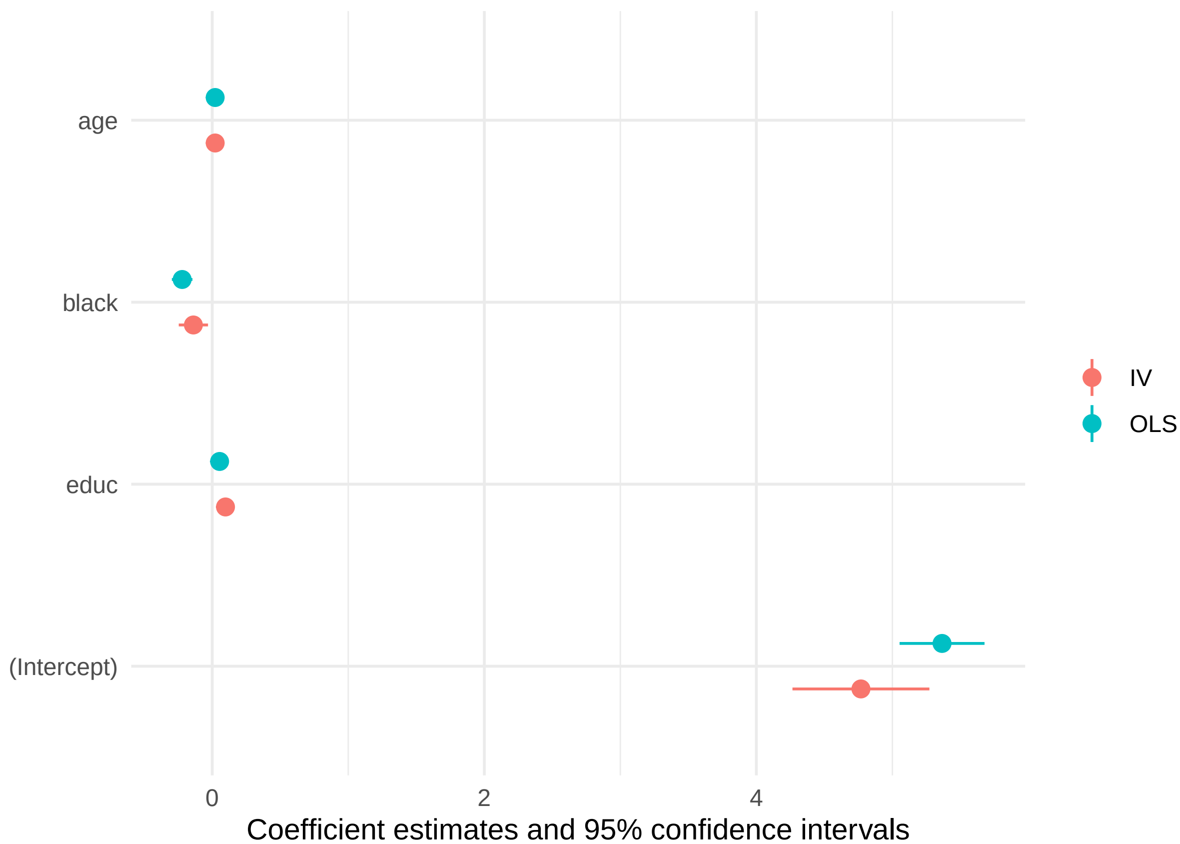
1. `y ~ exogenous | endogenous | instruments` separates the variables clearly but muddles two-stage least squares
2. `y ~ exogenous + endegenous | exogenous + instruments` requires repetition but expresses two-stage least squares clearly.

```
iv_fit <- ivreg::ivreg(lwage ~ black + age | educ | feduc + meduc, data = d)
# beware namespace conflicts - AER has an old ivreg()
lm_fit <- lm(lwage ~ educ + black + age, data = d)

msummary(list(IV = iv_fit, OLS = lm_fit),
                 vcov = vcovHC,
                 statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                 output = "latex_tabular")
```

[27] `ivreg` is my default choice because of its syntactical similarity to Stata. However, `fixest` also has excellent support for IV regression with both panel and non-panel data.

	IV	OLS
(Intercept)	4.769	5.365
	SE: 0.252 CI: [4.274, 5.264] t-value: 18.907	SE: 0.166 CI: [5.038, 5.691] t-value: 32.246
educ	0.097	0.054
	SE: 0.015 CI: [0.068, 0.126] t-value: 6.598	SE: 0.006 CI: [0.042, 0.066] t-value: 9.021
black	-0.139	-0.221
	SE: 0.056 CI: [-0.248, -0.029] t-value: -2.481	SE: 0.040 CI: [-0.299, -0.143] t-value: -5.590
age	0.022	0.022
	SE: 0.005 CI: [0.012, 0.031] t-value: 4.512	SE: 0.004 CI: [0.013, 0.030] t-value: 5.174
Num.Obs.	722	935
R2	0.091	0.155
R2 Adj.	0.088	0.152
AIC		888.1
BIC		912.3
Log.Lik.		-439.072

```
modelplot(list(IV = iv_fit, OLS = lm_fit))
```



7 Generalized Linear Models

Generalized linear models are generalizations of ordinary least squares linear regression. GLM generalizes linear regression by relating the model to the outcome variable by a link function.[28] R's syntax matches this distinction.

7.1 Logit and Probit

Logistic regression is an essential tool for binary response variables.

An important caution: functions that take models as arguments (`summary()`, any `broom` function, etc.) are not one, super flexible function. Instead, like many complex functions, these functions merely determine the type of the model that has been passed to the function then call a method (a type of function) built specifically for that object.[29] That was not an important piece of information until now because most functions are built for `lm()` and other linear models. With `glm()` models, though, you will need to refer to the function's `.glm()` method for documentation, because issues will arise - see an example with `broom` in the next code chunk.

```
set.seed(48)
n <- 200
x <- rnorm(n)
ystar <- 1 + x + rnorm(n, sd = exp(x))
y <- as.numeric(ystar > 0)

d <- data.frame(x, ystar, y)
lm_mod <- lm(y ~ x, data = d)
log_mod <- glm(y ~ x, family = binomial(link = "logit"), data = d)
prob_mod <- glm(y ~ x, family = binomial(link = "probit"), data = d)

models <- list("OLS" = lm_mod, "Logit" = log_mod, "Probit" = prob_mod)
msummary(models,
                vcov = vcovHC,
                statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                output = "latex_tabular")
```

	OLS	Logit	P
(Intercept)	0.696	0.893	0
	SE: 0.032 CI: [0.634, 0.759] t-value: 22.009	SE: 0.171 CI: [0.558, 1.229] t-value: 5.216	SE: 0.099 CI: [0.338
x	0.118	0.599	0
	SE: 0.036 CI: [0.046, 0.190] t-value: 3.242	SE: 0.210 CI: [0.188, 1.010] t-value: 2.858	SE: 0.123 CI: [0.084
Num.Obs.	200	200	
R2	0.066		
R2 Adj.	0.061		
AIC	247.9	234.8	2
BIC	257.8	241.4	2
Log.Lik.	-120.932	-115.407	-11

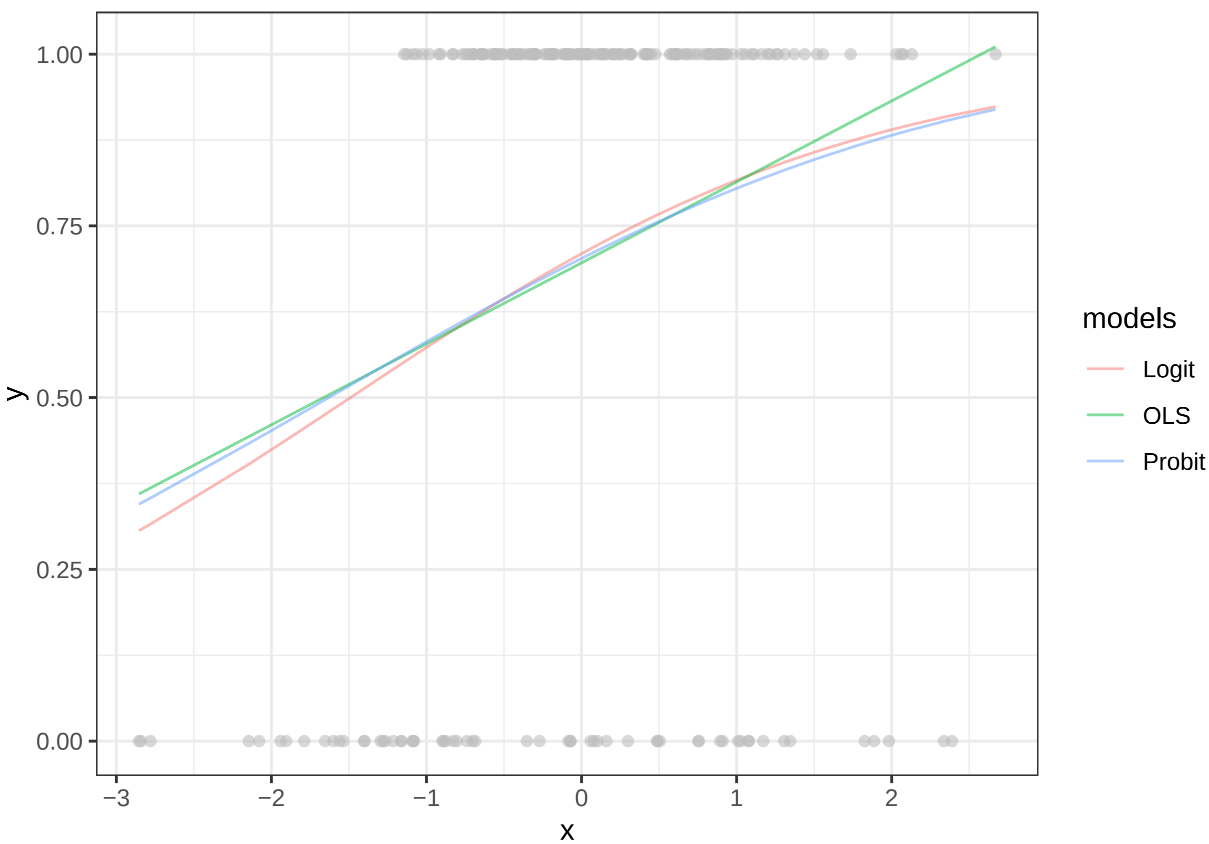
[28] This terminology might be new to some, but you are most likely already familiar with the underlying models, like logistic regression. I cover this generalization because this is how the matrix algebra works in R.

[29] Try calling `summary.lm()` on an `lm` object. It is also worth noting that model fitting functions rely on methods, like `lm.fit()`. In some big data/simulation applications, this matters - `lm.fit()` and it's even slimmer and faster parner, `.lm.fit()`, are up to 2x as fast. Regardless, when experiencing problems with more complex functions, often times the solution is in the documentation for the method, especially with the less common use cases.

This code wrangles the list of models into an `augment()` list of data with predictions and residuals. Be aware that `augment.glm()` takes an argument `type.predict` that puts predictions on the scale of the linear predictors, not the response variable. So, if you want to be seeing the predictions as probabilities between 0 and 1, this is an argument to `~augment` (`purrr::map()` takes a formula) that must be supplied when `augment()` calls `augment.glm()`. For more advanced and different types of models, expect similar behavior and remember that you must find the model-specific function to debug.

A bit of data wrangling with `purrr:map()` extracts the fitted values for plotting.

```
models <- list("OLS" = lm_mod, "Logit" = log_mod, "Probit" = prob_mod) # model list
models <- map(models, ~augment(.x, type.predict = "response")) # apply augment to each model, returning
df <- bind_rows(models, .id = "models") # bind 3 df's together

ggplot(df) +
    geom_point(aes(x = x, y = y), alpha = .25, color = "gray") +
    geom_line(aes(x = x, y = .fitted, group = models, color = models), alpha = .5) +
    scale_fill_brewer(palette = "Spectral") +
    theme_bw()
```



Now, recreating that analysis with real data.

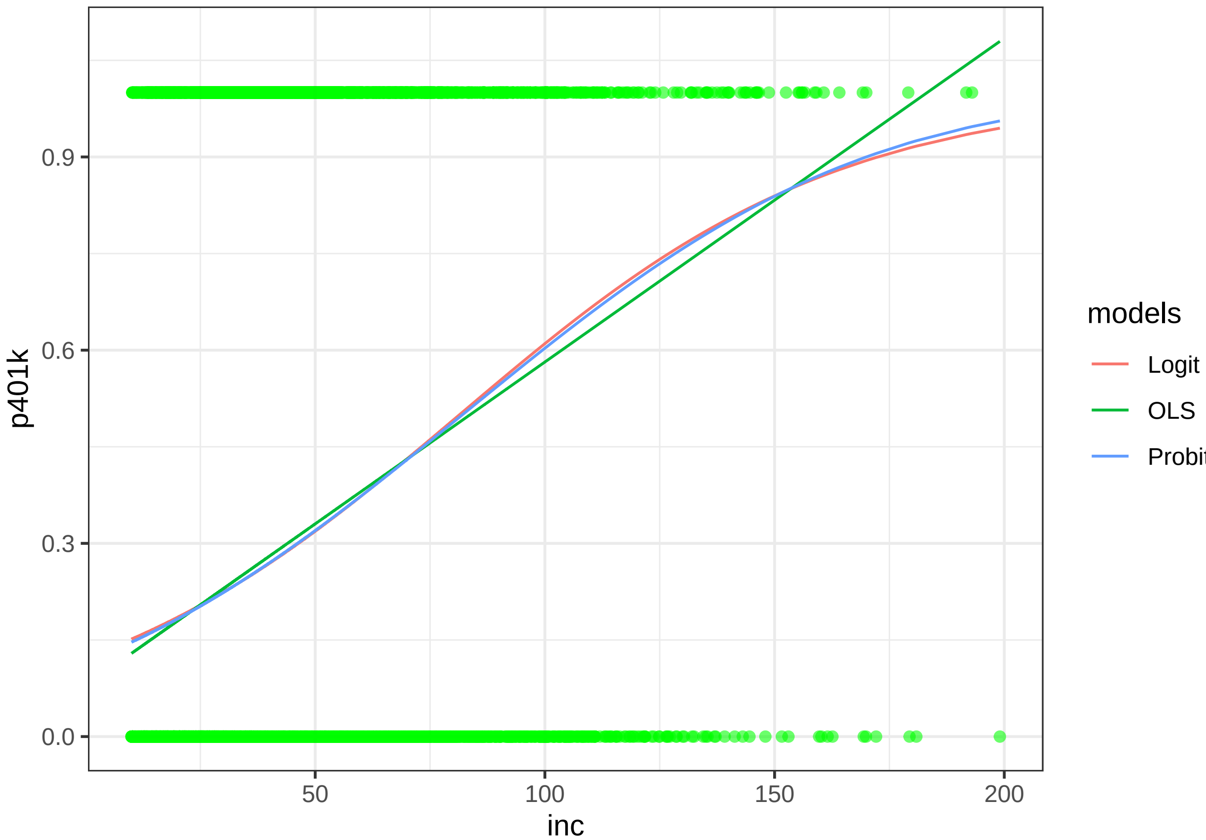
```
d <- wooldridge::k401ksubs
d <- select(d, p401k, inc)

lm_mod <- lm(p401k ~ inc, data = d)
log_mod <- glm(p401k ~ inc, family = binomial(link = "logit"), data = d)
prob_mod <- glm(p401k ~ inc, family = binomial(link = "probit"), data = d)
```

```r
models <- list("OLS" = lm_mod, "Logit" = log_mod, "Probit" = prob_mod)
msummary(models,
                 vcov = vcovHC,
                 statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                 output = "latex_tabular")
```

	OLS	Logit	
(Intercept)	0.079	-1.966	
	SE: 0.008 CI: [0.063, 0.095] t-value: 9.601	SE: 0.050 CI: [-2.063, -1.868] t-value: -39.539	SE: 0.029 CI: [-1
inc	0.005	0.024	
	SE: 0.000 CI: [0.005, 0.005] t-value: 24.523	SE: 0.001 CI: [0.022, 0.026] t-value: 22.495	SE: 0.001 CI: [
Num.Obs.	9275	9275	
R2	0.073		
R2 Adj.	0.073		
AIC	10689.7	10292.5	
BIC	10711.1	10306.8	
Log.Lik.	-5341.871	-5144.242	

```r
models <- list("OLS" = lm_mod, "Logit" = log_mod, "Probit" = prob_mod) # model list
models <- map(models, ~augment(.x, type.predict = "response")) # apply augment to each model, returning
df <- bind_rows(models, .id = "models") # bind 3 df's together

ggplot(df) +
    geom_point(aes(x = inc, y = p401k), alpha = .25, color = "green") +
    geom_line(aes(x = inc, y = .fitted, group = models, color = models)) +
    scale_fill_brewer(palette = "Spectral") +
    theme_bw()
```

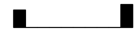
	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
cigs	30	0	8.7	13.7	0.0	0.0	80.0	
age	69	0	41.2	17.0	17.0	38.0	88.0	
income	11	0	19304.8	9143.0	500.0	20000.0	30000.0	



7.2 Tobit with survival and AER

```
d <- wooldridge::smoke
d <- select(d, cigs, age, income)
datasummary_skim(d, output = "kableExtra")
```

```
tob_fit <- tobit(cigs ~ age + income, left = 0, right = Inf, data = d)
msummary(tob_fit,
                 vcov = vcov, # strange standard error calculation issues with Tobit
                 statistic = "SE: {std.error} CI: [{conf.low}, {conf.high}] t-value: {statistic}",
                 output = "latex_tabular")
```

	Unique (#)	Missing (%)	Mean	SD	Min	Median	Max	
inlf	2	0	0.6	0.5	0.0	1.0	1.0	
lwage	374	43	1.2	0.7	-2.1	1.2	3.2	
educ	13	0	12.3	2.3	5.0	12.0	17.0	
nwifeinc	706	0	20.1	11.6	-0.0	17.7	96.0	
age	31	0	42.5	8.1	30.0	43.0	60.0	
kidslt6	4	0	0.2	0.5	0.0	0.0	3.0	

	Model 1
(Intercept)	-3.037
	SE: 4.165 CI: [-11.201, 5.127] t-value: -0.729
age	-0.152
	SE: 0.073 CI: [-0.295, -0.009] t-value: -2.084
income	0.000
	SE: 0.000 CI: [0.000, 0.000] t-value: 0.693
Num.Obs.	807
AIC	3563.4
BIC	3582.2
Log.Lik.	-1777.713
iter	3.000

7.3 Heckman with `ssmrob`

The Heckman model, also known as the Type II Tobit or, coloquially, the Heckit, is a variant of the Tobit model used to correct bias from non-randomly selected sample or an incidentally truncated dependent variable.

Unfortunately, because the Heckman model is less common, it is not particularly well-supported in R. `sampleSelection`, my package of choice, does not support heteroskedasticity robust standard error calculation. It also does not support `broom`, `sandwich`, or `lmtest` analysis. The `ssmrob` package only offers robust `rlm()` modelling for Heckman models.[30]

```
d <- wooldridge::mroz
d <- select(d, inlf, lwage, educ, nwifeinc, age, kidslt6)
datasummary_skim(d, output = "kableExtra")
```

```
heck_fit <- heckit(selection = inlf ~ educ + nwifeinc + age + kidslt6, outcome = lwage ~ educ + nwifein
summary(heck_fit)
```

```
--------------------------------------------
Tobit 2 model (sample selection model)
Maximum Likelihood estimation
Newton-Raphson maximisation, 6 iterations
Return code 8: successive function values within relative tolerance limit (reltol)
```

[30]In theory, one might be able to hack together a least-squares with heteroskedasticity robust standard errors model from this package, but I could not do it after an hour or two of work.

```
Log-Likelihood: -891.6522
753 observations (325 censored and 428 observed)
11 free parameters (df = 742)
Probit selection equation:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.107443   0.410439   0.262    0.794
educ         0.145925   0.022907   6.370 3.31e-10 ***
nwifeinc    -0.020110   0.004462  -4.507 7.63e-06 ***
age         -0.026906   0.006840  -3.934 9.16e-05 ***
kidslt6     -0.704553   0.114954  -6.129 1.44e-09 ***
Outcome equation:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.138695   0.304383   0.456 0.648768
educ        0.060485   0.017882   3.382 0.000756 ***
nwifeinc    0.009695   0.003657   2.651 0.008189 **
age         0.010888   0.004631   2.351 0.018978 *
   Error terms:
      Estimate Std. Error t value Pr(>|t|)
sigma  0.78970    0.04419  17.869   <2e-16 ***
rho   -0.73451    0.07657  -9.593   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Be aware that the default method, `method = "2-step"`, is the equivalent of `heckprob` in Stata, while `method = "ml"` is the equivalent of `heckman` in Stata.

8 Conclusion and Additional Resources

Thank you for reading, I hope this was as helpful for you as it was for me to write! One of the great things about well-documented open-source packages is that learning them is fairly straightforward - the documentation was written for the most general, inexperienced audience. But, finding them can be a challenge. Here is a list of resources that might help you learn something new in R.

8.1 Functions

R's verbosity can be annoying at times. Defining some of your own functions, even if they are just wrappers for one function, can save time and make code substantially more readable.

```
msum_rob <- function(list_models,
                                         se = vcovHC,
                                         stat_line = c(
                                               "se: {std.error}",
                                               "Conf. Int.: [{conf.low}, {conf.high}]",
                                               "t-stat: {statistic}",
                                               "p-value: {p.value}"),
                                         star_tf = TRUE,
                                         output_format = "gt") {
    return(modelsummary::msummary(
        models = list_models,
        vcov = se,
        statistic = stat_line,
        stars = star_tf,
        output = output_format))
}
msum_rob(lm_fit, output_format = "latex_tabular")
```

```
Warning: In version 0.8.0 of the `modelsummary` package, the default significance markers produced by tl
This warning is displayed once per session.
```

	Model 1
(Intercept)	5.365***
	se: 0.166
	Conf. Int.: [5.038, 5.691]
	t-stat: 32.246
	p-value: 0.000
educ	0.054***
	se: 0.006
	Conf. Int.: [0.042, 0.066]
	t-stat: 9.021
	p-value: 0.000
black	-0.221***
	se: 0.040
	Conf. Int.: [-0.299, -0.143]
	t-stat: -5.590
	p-value: 0.000
age	0.022***
	se: 0.004
	Conf. Int.: [0.013, 0.030]
	t-stat: 5.174
	p-value: 0.000
Num.Obs.	935
R2	0.155
R2 Adj.	0.152
AIC	888.1
BIC	912.3
Log.Lik.	-439.072

```
single_F <- function(model) {
    return(waldtest(model, vcov = vcovHC, test = "F"))
}

joint_F <- function(model1, model2) {
    return(waldtest(model1, model2, vcov = vcovHC, test = "F"))
}
```

8.2 Review and Validation in Econometrics

If you want simpler review materials (or external validation), check out RStudio's Stata to R cheatsheet.[31] I also took a lot of recommendations from University of Oregon Professor Grant McDermott's Data Science for Economists open-sourced lecture notes.

8.3 Package Exploration

There are more packages that are worth mentioning.

[31] When I found this halfway through working through this paper, my initial reaction was to give up and just refer to it when necessary. However, it's missing a lot of critical techniques, like time series, which is one of the most crowded spaces in R statistical analysis - I sorted through `base` and half a dozen other packages before finding `fable` and `tsibble`, a combination which I think is really straightforward and covers the ground not covered well by `base`.

- `MASS` does some back-end statistical computation - it's worth exploring for more niche statistical tests and matrix algebra work
- `tigerstats` is a great package for simulating data and statistical tests[32]
- `easystats` other packages provide additional tools for modelling analysis
- `DataExplorer` has excellent tools for preliminary analysis of a dataset - very little that could go in a final paper, but some functions that help you understand your data (like where missing values are, for example)
- `janitor` provides good tools for cleaning data en masse: `clean_names()` in particular is the fastest way to get unified style variable names
- `data.table` is a package worth knowing if you work larger data sets
- `rio` makes importing and exporting data a breeze and is substantially faster than the `tidyverse` `readr` package and the `base` `read.*()` family of functions
- `AER`, which I only used for Tobit, is a legendary econometrics package that has slowly lost almost all of its functions, but retains iconic, quality data sets[33]

If you want more depth, work through the packages used above. I estimate that I covered about 20% of the overall utility of each. Read documentation, test example code, and follow dependencies to understand what a package offers. To broaden your skillset, learn the tools of one package deeply is a great return on investment. Of course, it also may be worth exploring different packages.[34]

If you want new packages to solve different problems, use CRAN's task views. Find the general "Statistics for the Social Sciences" here. These are essentially maps of the available packages for different statistical problems organized by topic. The Econometrics, Time Series, and Finance task views (linked from the general social science sheet) list nearly every relevant package for each topic. Cheatsheets, vignettes, and online documentation are great resources to find other packages that may solve your problem or work better in your use case.

If you want information on data science topics, explore one of the many machine-learning packages available in R, as well as learning a pre-processing framework like `tidymodels` with `parsnip` and `recipes`. `caret`, `mlr3`, `e1071`, and `keras` seem to be the most popular, in addition to package for more specific problems.

8.4 Online Resources for R

The R community online is incredibly strong. That means, first and foremost, that there are many high-quality, free sources of information. It also means your questions can get answered on Twitter or StackOverflow quickly. The only important caution: know the author and their field. Every subfield (particularly the triangle of computer science, data science, and economics) has its own quirks.[35]

If you want more R, the community is mostly on Twitter, GitHub, and the blogosphere. Twitter (follow #rstats, it is incredibly lively) is a great place to start. You will quickly come across rbloggers.com, a website that compiles posts from more than 300 blogs. A compilation of the best R stuff is available on R Weekly.

If you want more fundamental data science skills, Hadley Wickham, Chief Data Scientist at RStudio, is a staple of the R community and the mastermind behind `tidyverse`. A YouTube search for his lectures reveals a treasure-trove of examples of data science workflow and tidy tools.

[32] `base` has much of the same functionality, but `tigerstats` improves dramatically on the implementation and makes code much more readable.

[33] Many of the best econometrics packages, including `ivreg`, emerged from `AER`. Notably, while `AER` has more than 100 datasets, it retainsonly two functions, `ivreg()` and `tobit()`, an interface to `survival`, a package that also computes a Tobit model.

[34] One chain worth following is the *fit functions. Lots of packages have a private package that underlies their analysis. For example, `fable` is essentially a wrapper around `fabletools`, which does a lot of the computation. Understanding what packages are being used to do the computation and what those depend on can be really beneficial when trying to find the right tool for the problem.

[35] Data science and economics tools overlap, but are not identical. Heteroskedasticity-robust standard errors are not the norm in data science and much more of data science is focused on prediction, not inference. On the other hand, economists often write awful code and rarely use the best tools for the task. Don't be surprised when you find an economics in R textbook doing all of their cleaning and visualization in `base`.

If you want more econometrics, some of my favorite resources are the GitHub's of two University of Oregon economics professors, who have dumped a ton of resources. Edward Rubin and Grant McDermott open-source entire undergrad, master's, and doctorate-level courses. They include lots of big data, data science, and computer science tools as well, all things that are useful to them in their research. Principles of Econometrics with R covers a lot of the same ground this paper does, and some more, but is five years old and not exactly written with smooth implementation in mind. If you need some review of the math, Econometrics With R review the math and demonstrates R code, but again, not exactly for the modern data scientist.

If you want more data science, R + Data Science has a good list of resources and covers some other data science territory. Perhaps the best resource it mentions is Introduction to Statistical Learning, a canonical data science textbook (with lots of math). A relatively new companion was just released by Emil Hvitfeldt, one of the masterminds of `tidymodels` at RStudio, that uses the `tidymodels` framework to do the labs in Introduction to Statistical Learning.

All of these resources are available online for free.[36]

I hope some of these are useful!

[36] It's disappointing that this is unimaginable in economics and most old, stuffy fields. The fact that Hadley Wickham and a couple of the folks at RStudio created the premier framework for data wrangling and visualization in `tidyverse`, wrote a few books about it, and then made it all free online while N. Gregory Mankiw still sells introductory economics textbooks for $86 a pop never ceases to amaze me.