Regression with R

EC 425/525, Lab 4

Edward Rubin
03 May 2019

Prologue

Schedule

Last time

1. `RStudio` basics
2. Getting data in and out of `R`.

Today

Regression!

Review

Data i/o

`readr` and `haven` have you covered for most of your i/o needs.

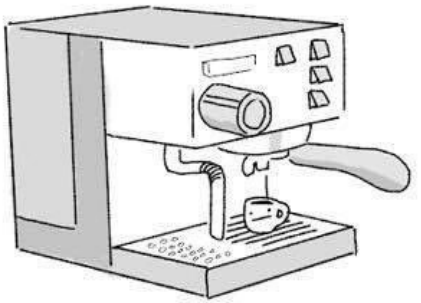
Best practices

1. Write code in `R` scripts. Troubleshoot in `RStudio`. Then run the scripts.
2. Comment your code. (`# This is a comment`)
3. Name objects and variables with intelligible, standardized names.
   - **BAD** `ALLCARS`, `Vl123a8`, `a.fun`, `cens.12931`, `cens.12933`
   - **GOOD** `unique_cars`, `health_df`, `sim_fun`, `is_female`, `age`
4. Set seeds when generating randomness, *e.g.*, `set.seed(123)`.
5. Parallelize when possible. (Packages: `parallel`, `purrr`, `foreach`, *etc.*)
6. Use projects in `RStudio` (next). And organize your projects.

Regression in `R`

Favored empirical technique by choice of coffee maker

Massive administrative data



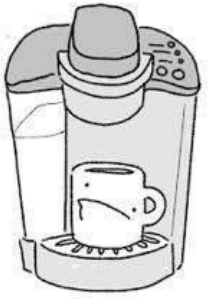
RCT

Regression discontinuity

Difference-in -difference

Analytic narrative

Matching

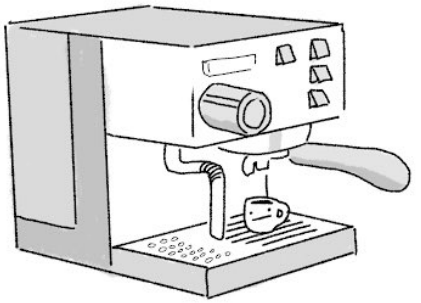


Kitchen sink regression

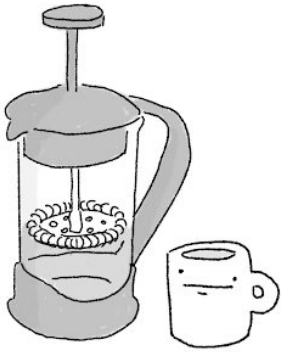
@Tommy Siegel

WHAT YOUR COFFEE PREPARATION METHOD SAYS ABOUT YOU

NOT REALLY INTO ABSTRACT ART



HAS A NEW YORKER SUBSCRIPTION BUT HAS NEVER READ IT

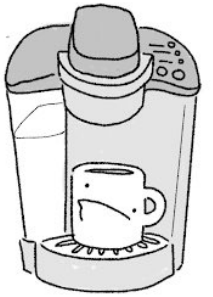


FACE THE FACTS: YOU'RE FAKING FANCY

HAS A NEW YORKER SUBSCRIPTION AND SOMEHOW READS IT

BELIEVES VINYL IS ALWAYS HIGHER QUALITY DESPITE CONFLICTING EVIDENCE

UNBEARABLE YOUNG SNOB OR GERIATRIC ITALIAN



WHO HURT YOU

@Tommy Siegel

Original credit Tommy Siegel @TommySiegel

Econ update David Clingingsmith @dclingi

Regression in R

The default option: `lm()`

R's `base`[†] option for estimating **l**inear regression **m**odels is `lm()`.

You use a formula to specify your linear regression model in `lm()`, *e.g.*,

```
lm(y ~ x)
```

- Estimates $y_i = \beta_0 + \beta_1 x_i + u_i$ (R automatically includes an intercept)[††]
- using the data stored in the objects `y` and `x`.

```
lm(y ~ x, data = amazing_df)
```

- Estimates $y_i = \beta_0 + \beta_1 x_i + u_i$
- using the variables (columns) `y` and `x` from the object `amazing_df`.

† `base` is R's basic (default) package—loaded on opening.

†† You can remove the intercept by adding `-1` into the formula, *e.g.*, `lm(y ~ -1 + x)`.

Regression in R

More `lm()`

Need include more variables? Easy.

```
lm(y ~ x1 + x2 + x3, data = some_df)
```

- Estimates $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + u_i$
- referencing the object `some_df` for the named variables.

Even more `lm()`

Do you want to transform/interact variables? Also easy: use `I()`.

```
lm(y ~ x1 + x2 + I(x1^2) + I(x2^2) + I(x1*x2), data = poly_df)
```

- Estimates $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{1i}^2 + \beta_4 x_{2i}^2 + \beta_5 x_{1i} x_{2i} + u_i$
- using variables named in object `poly_df`
- or created via `I()`

Note The following are equivalent

- `lm(y ~ x1 + x2 + I(x1*x2))`
- `lm(y ~ x1 + x2 + x1:x2)`
- `lm(y ~ x1*x2)`

Transforming variables with `lm()`

Notice that in our call

```
lm(y ~ x1 + x2 + I(x1^2) + I(x2^2) + I(x1*x2), data = poly_df)
```

we did not need to create x_1^2, x_2^2, and $x_1 \times x_2$ in the dataset.

R will do the calculation (as long as `x1` and `x2` exist somewhere).

This is true for any transformation of variables/objects.

- Math./stat. transformations: `I(x^2)`, `I(x/3)`, `I((x - mean(x))/sd(x))`
- Log/exponential transformations: `log(x)`, `exp(x)`
- Indicators: `I(x < 100)`, `I(x == "Oregon")`

Need data

Before we can go any further, we need data.

Let's use data from **LaLonde (1986)**.

This (famous) paper compared experimental and non-experimental estimates of the effect of a randomized jobs program called the National Supported Work Demonstration (NSW).

The data are available online as a `.dta` file.

```
# Load the 'haven' package
p_load(haven)
# Load treatment data
lalonde_df ← read_dta("http://users.nber.org/~rdehejia/data/nsw.dta")
```

Show 7 entries Search:

	treat	age	education	black	hispanic
1	1	37	11	1	0
2	1	22	9	0	1
3	1	30	12	1	0
4	1	27	11	1	0
5	1	33	8	1	0
6	1	22	9	1	0
7	1	23	12	1	0

Showing 1 to 7 of 722 entries

Previous 1 2 3 4 5 ... 104 Next

Show 7 entries Search:

	married	nodegree	re75	re78
1	1	1	0	9930.0458984375
2	0	1	0	3595.89404296875
3	0	0	0	24909.44921875
4	0	1	0	7506.14599609375
5	0	1	0	289.789886474609
6	0	1	0	4056.49389648438
7	0	0	0	0

Showing 1 to 7 of 722 entries

Output from `lm()`

Back to `lm()`.

The information that `lm()` prints to screen is underwhelming.

```
lm(re78 ~ treat, data = lalonde_df)
```

```
#>
#> Call:
#> lm(formula = re78 ~ treat, data = lalonde_df)
#>
#> Coefficients:
#> (Intercept)        treat
#>      5090.0        886.3
```

But there's a lot more under the hood.

Hidden information

Let's save the output of our call to `lm()`.

```
est_lalonde ← lm(re78 ~ treat, data = lalonde_df)
```

What **class** is `est_lalonde`?

```
est_lalonde %>% class()
```

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

which means there's probably a lot more going on than what printed.

Regression in R

Does it have **names**?

```
est_lalonde %>% names()
```

```
#>  [1] "coefficients"  "residuals"     "effects"       "rank"
#>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#>  [9] "xlevels"       "call"          "terms"         "model"
```

Can we **tidy** it?

```
est_lalonde %>% tidy()
```

```
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)    5090.      303.     16.8  9.54e-54
#> 2 treat           886.      472.      1.88 6.09e- 2
```

You'll generally see folks take the `summary()` of an `lm` object.

```
est_lalonde %>% summary()
```

```
#>
#> Call:
#> lm(formula = re78 ~ treat, data = lalonde_df)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#>  -5976  -5090  -1519   3361  54332
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   5090.0      302.8  16.811   <2e-16 ***
#> treat          886.3      472.1   1.877   0.0609 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 6242 on 720 degrees of freedom
#> Multiple R-squared:  0.004872,    Adjusted R-squared:  0.003489
#> F-statistic: 3.525 on 1 and 720 DF,  p-value: 0.06086
```

Regression in R

`summary()`

Interestingly, `summary()` contains additional information.

```
est_lalonde %>% names()
```

```
#>  [1] "coefficients"  "residuals"     "effects"       "rank"
#>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#>  [9] "xlevels"       "call"          "terms"         "model"
```

```
est_lalonde %>% summary() %>% names()
```

```
#>  [1] "call"          "terms"         "residuals"     "coefficients"
#>  [5] "aliased"       "sigma"         "df"            "r.squared"
#>  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

Regression in R

`tidy()`

That said, `summary()`'s output is a bit overwhelming.

As we saw, the output from `tidy()` contained everything we generally want.

```
est_lalonde %>% tidy()
```

```
#> # A tibble: 2 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
#> 1 (Intercept)    5090.      303.     16.8  9.54e-54
#> 2 treat           886.      472.      1.88 6.09e- 2
```

Non-standard standard errors

Q Which estimator does `lm()` use for its standard errors?

A Spherical/classical/homoskedastic.

Q What if we want something else?

Outside options

Q What if we want something else?

A This is why you'll end up replacing `lm()` with one of two different linear-regression packages.

1. `felm()` from the `lfe` package is incredibly fast with high-dimensional fixed effects, easily implements IV/2SLS, and offers heteroskedasticity- and cluster-robust standard errors estimators.

1. `lm_robust()` from the `estimatr` package is fast, has a sister function named `iv_robust()` for IV/2SLS, and allows for a wide range of heteroskedasticity- and cluster-robust standard error estimators.

Both packages maintain the same `y ~ x` formula (plus additional features).

Regression in R

`estimatr`

Let's check out `lm_robust()` from `estimatr`.

We still need input a `formula` and `data`, but now we have options for the type of standard error estimator (`se_type`).

> The sort of standard error sought. If `clusters` is not specified the options are "HC0", "HC1" (or "stata", the equivalent), "HC2" (default), "HC3", or "classical". If `clusters` is specified the options are "CR0", "CR2" (default), or "stata". Can also specify "none", which may speed up estimation of the coefficients.

`estimatr` documentation for `lm_robust()`

Regression in R

`estimatr`

Now for the magic.

```
# Load 'estimatr' package
p_load(estimatr)
# Estimate
robust_lalonde ← lm_robust(re78 ~ treat, data = lalonde_df)
# Tidy results
tidy(robust_lalonde)
```

```
#>          term  estimate std.error statistic      p.value   conf.low
#> 1 (Intercept) 5090.0483  277.3680 18.351243 5.335543e-62 4545.50153
#> 2       treat  886.3037  488.2045  1.815435 6.987292e-02  -72.17078
#>   conf.high  df outcome
#> 1  5634.595 720    re78
#> 2  1844.778 720    re78
```

Prediction

So what if we want to get the predictions from a regression?

You have *a lot* of options. Here are two (for different scenarios).

Scenario 1 You want predictions for the original $\boldsymbol{X}$,

```
robust_lalonde$fitted.values
```

Scenario 2 You want predictions (with prediction intervals) for new data,

```
predict(
  object = robust_lalonde,
  newdata = new_df,
  interval = "prediction"
)
```

Other regressions

Ordinary least squares (OLS) is only one of many types of regressions.

If you can run one regression in R, you can run any regression in R.[†]

E.g., logistic regression ("logit")[††] in R uses the `glm()` function.

To estimate the probability of treatment on observables in LaLonde's data,

```
glm(
  treat ~ age + education + black + hispanic + married,
  family = "binomial",
  data = lalonde_df
)
```

† This is not to say that you should or that you'll know what you're doing.

†† Logit models are a popular nonlinear regression model for binary outcomes.

```r
# Estimate logit model
trt_logit ← glm(
  treat ~ age + education + black + hispanic + married,
  family = "binomial",
  data = lalonde_df
)
# Tidy logit results
trt_logit %>% tidy()
```

```
#> # A tibble: 6 x 5
#>   term        estimate std.error statistic p.value
#>   <chr>          <dbl>     <dbl>     <dbl>   <dbl>
#> 1 (Intercept) -0.888      0.611     -1.45    0.146
#> 2 age          0.00229    0.0119     0.193   0.847
#> 3 education    0.0606     0.0456     1.33    0.183
#> 4 black       -0.163      0.259     -0.629   0.529
#> 5 hispanic    -0.289      0.346     -0.835   0.404
#> 6 married      0.0600     0.210      0.285   0.775
```

(Not too surprising, since treatment was randomly assigned.)

Additional resources

There's more

General resources

- The `swirl` package will teach you R in R.

Regression with `estimatr`

- Getting started with `estimatr`
- A cheatsheet for `estimatr`

Logit models (logistic regression)

- Examples and discussion
- Interpretting results from logit models

Table of contents

Regression in R