Lab #7 - More on Regression in R

Econ 224

September 18th, 2018

Robust Standard Errors

Your reading assignment from Chapter 3 of ISL briefly discussed two ways that the standard regression inference formulas built into R can go wrong: (1) non-constant error variance, and (2) correlation between regression errors. Today we'll briefly look at the first of these problems and how to correct for it.

Consider the simple linear regression $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$. If the variance of ϵ_i is unrelated to the value of the predictor x_i, we say that the regression errors are *homoskedastic*. This is just a fancy Greek work for *constant variance*. If instead, the variance of ϵ_i depends on the value of x_i, we say that the regression errors are *heteroskedastic*. This is just a fancy Greek word for *non-constant variance*. Heteroskedasticity does not invalidate our least squares estimates of β_0 and β_1, but it does invalidate the formulas used by `lm` to calculate standard errors and p-values.

Let's look at a simple simulation example:

```
set.seed(4321)
n <- 100
x <- runif(n)
e1 <- rnorm(n, mean = 0, sd = sqrt(2 * x))
e2 <- rnorm(n, mean = 0, sd = 1)
intercept <- 0.2
slope <- 0.9
y1 <- intercept + slope * x + e1
y2 <- intercept + slope * x + e2
library(tidyverse)
mydat <- tibble(x, y1, y2)
rm(x, y1, y2)
```

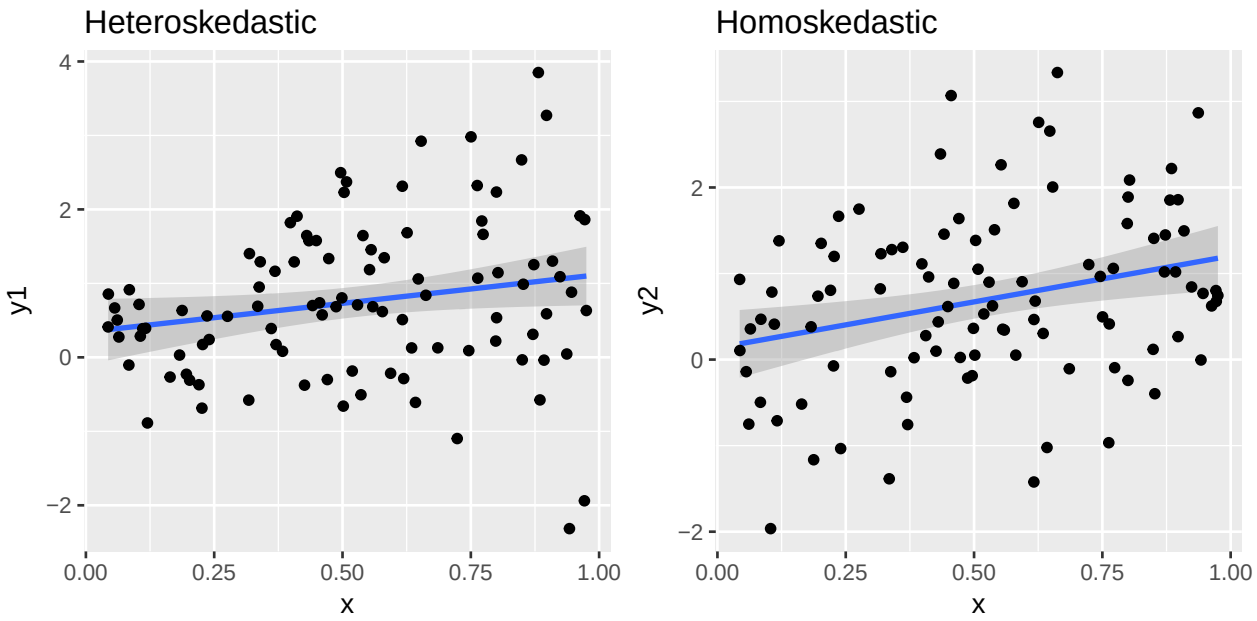
From the simulation code, we see that the errors `e1` are heteroskedastic since their standard deviation is a function of `x`. In contrast, the errors `e2` are homoskedastic since their standard deviation is *not* a function of `x`. This means that a regression of `y1` on `x` will exhibit heteroskedasticity but a regression of `y2` on `x` will not. We can see this from a plot of the data:

```
library(ggplot2)
library(gridExtra)

heterosked_plot <- ggplot(mydat, aes(x, y1)) +
  geom_smooth(method = 'lm') +
  geom_point() +
  ggtitle('Heteroskedastic')

homosked_plot <- ggplot(mydat, aes(x, y2)) +
  geom_smooth(method = 'lm') +
  geom_point() +
  ggtitle('Homoskedastic')

grid.arrange(heterosked_plot, homosked_plot, ncol = 2)
```



The values of `y1` "fan out" around the regression line since `e1` becomes *more variable* as `x` increases. In contrast, the values of `y2` do not show such a pattern: the variability in `e2` is unrelated to `x`.

`lm_robust`

We'll use the function `lm_robust` in the package `estimatr` to calculate the appropriate standard errors for a regression with heteroskedasticity. Make sure to install this package before proceeding. Such standard errors are often called *robust* because they do not depend on the assumption of homoskedasticity. The function `lm_robust` is nearly identical to `lm` but it allows us to specify a new argument `se_type` to indicate what kinds of standard errors we want to use. If we set `se_type = 'classical'` we'll get exactly the same standard errors as if we had used `lm`, namely standard errors that assume homoskedasticity:

```
library(estimatr)
reg_classical <- lm_robust(y1 ~ x, mydat, se_type = 'classical')
summary(reg_classical)
```

```
Call:
lm_robust(formula = y1 ~ x, data = mydat, se_type = "classical")

Standard error type:  classical

Coefficients:
            Estimate Std. Error t value Pr(>|t|) CI Lower CI Upper DF
(Intercept)   0.3418     0.2240   1.526  0.13027 -0.10273   0.7863 98
x             0.7766     0.3785   2.052  0.04286  0.02548   1.5277 98

Multiple R-squared:  0.04119 ,  Adjusted R-squared:  0.0314
F-statistic:  4.21 on 1 and 98 DF,  p-value: 0.04286
```

If we set `se_type = 'stata'` we'll get heteroskedasticity-robust standard errors identical to those calculated by the command `reg, robust` in Stata. Since many economists still use Stata, this is handy for being able

to replicate their results. Notice that the robust standard errors are *larger* than the classical ones. This is fairly typical in applications:

```
reg_robust <- lm_robust(y1 ~ x, mydat, se_type = 'stata')
summary(reg_robust)
```

```
Call:
lm_robust(formula = y1 ~ x, data = mydat, se_type = "stata")

Standard error type:  HC1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  CI Lower CI Upper DF
(Intercept)   0.3418     0.1739   1.966  0.05215 -0.003241   0.6868 98
x             0.7766     0.4068   1.909  0.05919 -0.030707   1.5839 98

Multiple R-squared:  0.04119 ,  Adjusted R-squared:  0.0314
F-statistic: 3.644 on 1 and 98 DF,  p-value: 0.05919
```

You should go back through the two preceding sets of regression results carefully and verify that the estimates are *identical* in each case. Because the standard errors are different, however, so are the test statistics and p-values. For example, `x` is significant at the 5% level when we use classical standard errors, but not when we use heteroskedasticity-robust standard errors. In general, failing to account for heteroskedasticity leads us to *understate* the true sampling uncertainty in our regression estimates.

F-tests with `lm_robust`

Heteroskedasticity doesn't just invalidate inference based on the t-tests from the `lm` summary output; it also invalidates any F-tests that we construct by passing these results to `linearHypothesis`. Fortunately, `lm_robust` makes it easy to fix this problem: as long as we fit our regression using `lm_robust` in place of `lm` and choose robust standard errors, when we pass the regression object to `linearHypothesis`, it will automatically make the appropriate adjustments. For example, notice that these *do not* give the same results:

```
library(car)
linearHypothesis(reg_classical, 'x = 0')
```

```
Linear hypothesis test

Hypothesis:
x = 0

Model 1: restricted model
Model 2: y1 ~ x

  Res.Df Df  Chisq Pr(>Chisq)
1     99
2     98  1 4.2098    0.04019 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
linearHypothesis(reg_robust, 'x = 0')
```

```
Linear hypothesis test

Hypothesis:
x = 0

Model 1: restricted model
Model 2: y1 ~ x

  Res.Df Df  Chisq Pr(>Chisq)
1     99
2     98  1 3.6442    0.05626 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

That is because the first one uses classical standard errors while the second uses heteroskedasticity-robust standard errors.

Exercise #1

Go back and do robust standard errors for one of the questions from the previous problem set.

Publication-quality Tables with `stargazer`

A crucial part of communicating our results in a statistical analysis creating tables that are clear, and easy to read. To make publication-quality tables like those that appear in academic journals, we'll use the package `stargazer`. Make sure to install this package before proceeding.

A Table of Summary Statistics

We'll start by learning how to make a simple table of summary statistics. There are a few quirks to be aware of here, so **please read this paragraph carefully!** The first thing you should know is that `stargazer` can only construct summary statistics for a *dataframe*. For almost all intents and purposes a tibble *is* a dataframe, but `stargazer` is an exception to this rule. If you have a tibble called, say `tbl`, then you will need to pass it into `stargazer` wrapped inside `as.data.frame()`. The second thing you should know is that using `stargazer` with `knitr` will not work unless you set the chunk option `results = 'asis'`. The third thing you need to know is that `stargazer` requires you to explicitly specify the kind of output that you want it to produce. If you will be knitting a `pdf` you'll need to set `type = 'latex'`. If you will be knitting an `html` document, you'll need to set `type = 'html'`. Finally, if you just want to display your table *without knitting it*, e.g. as a preview in RStudio, you'll need to set `type = 'text'`. Here is an example of the code that I ran to generate a pdf version of this document:

```
library(stargazer)
stargazer(mtcars, type = 'latex')
```

```
% Table created by stargazer v.5.2.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvard.edu
% Date and time: Sat, Sep 08, 2018 - 07:30:07 PM
```

Table 1:

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
mpg	32	20.091	6.027	10	15.4	22.8	34
cyl	32	6.188	1.786	4	4	8	8
disp	32	230.722	123.939	71	120.8	326	472
hp	32	146.688	68.563	52	96.5	180	335
drat	32	3.597	0.535	2.760	3.080	3.920	4.930
wt	32	3.217	0.978	1.513	2.581	3.610	5.424
qsec	32	17.849	1.787	14.500	16.892	18.900	22.900
vs	32	0.438	0.504	0	0	1	1
am	32	0.406	0.499	0	0	1	1
gear	32	3.688	0.738	3	3	4	5
carb	32	2.812	1.615	1	2	4	8

The `stargazer` command provides dozens of options for customizing the appearance of the output it generates. Here's a nicer version of the preceding table that uses some of these options:

```
mylabels <- c('Miles/gallon',
              'No. of cylinders',
              'Displacement (cubic inches)',
              'Horsepower',
              'Rear axle ratio',
              'Weight (1000lb)',
              '1/4 Mile Time',
              'V/S',
              'Manual Transmission? (1 = Yes)',
              'No. forward gears',
              'No. carburetors')
stargazer(mtcars,
          type = 'latex',
          title = 'Summary Statistics: Motor Trend Cars Dataset',
          digits = 1,
          header = FALSE,
          covariate.labels = mylabels)
```

Table 2: Summary Statistics: Motor Trend Cars Dataset

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
Miles/gallon	32	20.1	6.0	10	15.4	22.8	34
No. of cylinders	32	6.2	1.8	4	4	8	8
Displacement (cubic inches)	32	230.7	123.9	71	120.8	326	472
Horsepower	32	146.7	68.6	52	96.5	180	335
Rear axle ratio	32	3.6	0.5	2.8	3.1	3.9	4.9
Weight (1000lb)	32	3.2	1.0	1.5	2.6	3.6	5.4
1/4 Mile Time	32	17.8	1.8	14.5	16.9	18.9	22.9
V/S	32	0.4	0.5	0	0	1	1
Manual Transmission? (1 = Yes)	32	0.4	0.5	0	0	1	1
No. forward gears	32	3.7	0.7	3	3	4	5
No. carburetors	32	2.8	1.6	1	2	4	8

Notice how I reduced the number of significant figures presented in the table, added a caption and meaningful variable names. We can also customize which summary statistics are reported using the options `summary.stat` and `omit.summary.stat`. For example, if we only wanted to show the mean, standard deviation, and quartiles of the data, we could use the following:

```
stargazer(mtcars,
          type = 'latex',
          title = 'Summary Statistics: Motor Trend Cars Dataset',
          digits = 1,
          header = FALSE,
          covariate.labels = mylabels,
          summary.stat = c('mean',
                           'sd',
                           'p25',
                           'median',
                           'p75'))
```

Table 3: Summary Statistics: Motor Trend Cars Dataset

Statistic	Mean	St. Dev.	Pctl(25)	Median	Pctl(75)
Miles/gallon	20.1	6.0	15.4	19.2	22.8
No. of cylinders	6.2	1.8	4	6	8
Displacement (cubic inches)	230.7	123.9	120.8	196.3	326
Horsepower	146.7	68.6	96.5	123	180
Rear axle ratio	3.6	0.5	3.1	3.7	3.9
Weight (1000lb)	3.2	1.0	2.6	3.3	3.6
1/4 Mile Time	17.8	1.8	16.9	17.7	18.9
V/S	0.4	0.5	0	0	1
Manual Transmission? (1 = Yes)	0.4	0.5	0	0	1
No. forward gears	3.7	0.7	3	4	4
No. carburetors	2.8	1.6	2	2	4

Exercise #2

Make a pretty table of summary statistics from one of the recent problem set questions.

Regression Output

As we have seen, when you pass a dataframe to `stargazer`, its default is to construct a table of summary statistics. If you instead pass a *regression* object, it will make a regression table. For example: Run a bunch of regressions using `mtcars`

```
reg1 <- lm(mpg ~ disp, mtcars)
stargazer(reg1, type = 'latex',
          header = FALSE,
          digits = 1,
          title = 'Predicting Fuel Economy from Displacement')
```

Let's run a few more regressions and make a table that summarizes the results of *all* of them:

Table 4: Predicting Fuel Economy from Displacement

	Dependent variable:
	mpg
disp	-0.04^{***}
	(0.005)
Constant	29.6^{***}
	(1.2)
Observations	32
R^2	0.7
Adjusted R^2	0.7
Residual Std. Error	3.3 (df = 30)
F Statistic	76.5^{***} (df = 1; 30)
Note:	*p<0.1; **p<0.05; ***p<0.01

```
reg2 <- lm(mpg ~ wt, mtcars)
reg3 <- lm(mpg ~ disp + wt, mtcars)
stargazer(reg1, reg2, reg3,
          type = 'latex',
          digits = 1,
          header = FALSE,
          title = 'Regression Results for Motor Trend Dataset',
          covariate.labels = c('Displacement (cubic inches)', 'Weight (1000lb)'),
          dep.var.caption = 'Miles/gallon',
          notes = c('Data are courtesy of Motor Trend Magazine. Also, R rules!'))
```

Notice how I added a label for the dependent variable and appended a *note* to the regression table.

Exercise

Make a pretty regression table for one of the problems on the last problem set. Use robust standard errors.

Table 5: Regression Results for Motor Trend Dataset

	Miles/gallon		
	mpg		
	(1)	(2)	(3)
Displacement (cubic inches)	-0.04^{***}		-0.02^{*}
	(0.005)		(0.01)
Weight (1000lb)		-5.3^{***}	-3.4^{***}
		(0.6)	(1.2)
Constant	29.6^{***}	37.3^{***}	35.0^{***}
	(1.2)	(1.9)	(2.2)
Observations	32	32	32
R^2	0.7	0.8	0.8
Adjusted R^2	0.7	0.7	0.8
Residual Std. Error	3.3 (df = 30)	3.0 (df = 30)	2.9 (df = 29)
F Statistic	76.5^{***} (df = 1; 30)	91.4^{***} (df = 1; 30)	51.7^{***} (df = 2; 29)

Note: *p<0.1; **p<0.05; ***p<0.01

Data are courtesy of Motor Trend Magazine. Also, R rules!