Introduction to R

Module 5: Regression analysis and data visualization

Andrew Proctor

andrew.proctor@phdstudent.hhs.se

February 5, 2018

Intro

Goals for today

1. Introduce basics of linear regression models in R, including model diagnostics and specifying error variance structures.
2. Introduce further methods for panel data and instrumental variables.
3. Explore data visualization methods using the ggplot2 package.

Regression Basics

Linear Regression

The basic method of performing a linear regression in R is to the use the lm() function.

- To see the parameter estimates alone, you can just call the `lm()` function. But much more results are available if you save the results to a regression output object, which can then be accessed using the summary() function.

Syntax:

```
myregobject <- lm(y ~ x1 + x2 + x3 + x4,
                  data = mydataset)
```

CEX linear regression example

```
lm(expenditures ~ educ_ref, data=cex_data)
```

```
##
## Call:
## lm(formula = expenditures ~ educ_ref, data = cex_data)
##
## Coefficients:
## (Intercept)     educ_ref
##      -641.1        109.3
```

```
cex_linreg <- lm(expenditures ~ educ_ref,
                 data=cex_data)
```

CEX linear regression example ctd

```r
summary(cex_linreg)
```

```
##
## Call:
## lm(formula = expenditures ~ educ_ref, data = cex_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -541109     -899     -690     -506   965001
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -641.062     97.866   -6.55 5.75e-11 ***
## educ_ref     109.350      7.137   15.32  < 2e-16 ***
## ---
```

Formatting regression output: tidyr

With the tidy() function from the broom package, you can easily create standard regression output tables.

```
library(broom)
tidy(cex_linreg)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-641.0622	97.866411	-6.550381	0
educ_ref	109.3498	7.137046	15.321432	0

Formatting regression output: stargazer

Another really good option for creating compelling regression and summary output tables is the stargazer package.

- If you write your reports in LaTex, it's especially useful.

```
# From console:  install.packages("stargazer")

library(stargazer)

stargazer(cex_linreg, header=FALSE, type='latex')
```

Stargazer output

Table 2

	Dependent variable:
	expenditures
educ_ref	109.350***
	(7.137)
Constant	−641.062***
	(97.866)
Observations	305,972
R^2	0.001
Adjusted R^2	0.001
Residual Std. Error	7,024.151 (df = 305970)

Interactions and indicator variables

Including interaction terms and indicator variables in R is very easy.

- Including any variables coded as factors (ie categorical variables) will automatically include indicators for each value of the factor.
- To specify interaction terms, just specify `varX1*varX2`.
- To specify higher order terms, write it mathematically inside of **I()**.

Example:

```
wages_reg <- lm(wage ~ schooling + sex +
         schooling*sex + I(exper^2), data=wages)
```

Example with interactions and factors

```r
tidy(wages_reg)
```

term	estimate	std.error	statistic	p.value
(Intercept)	-2.0530687	0.6110201	-3.3600672	0.0007881
schooling	0.5672762	0.0500783	11.3277746	0.0000000
sexmale	-0.3256979	0.7790055	-0.4180945	0.6759053
I(exper^2)	0.0075173	0.0014436	5.2072237	0.0000002
schooling:sexmale	0.1431400	0.0659669	2.1698748	0.0300877

Setting reference groups for factors

By default, when including factors in R regression, the first *level* of the factor is treated as the omitted reference group.

- An easy way to instead specify the omitted reference group is to use the relevel() function.

Example:

```
wages$sex <- wages$sex %>% relevel(ref="male")
wagereg2 <- lm(wage ~ sex, data=wages); tidy(wagereg2)
```

term	estimate	std.error	statistic	p.value
(Intercept)	6.313021	0.0774650	81.49511	0
sexfemale	-1.166097	0.1122422	-10.38912	0

Useful output from regression

A couple of useful data elements that are created with a regression output object are fitted values and residuals. You can easily access them as follows:

- **Residuals:** Use the residuals() function.

```
myresiduals <- residuals(myreg)
```

- **Predicted values:** Use the fitted() function.

```
myfittedvalues <- fitted(myreg)
```

Model Testing

Using the lmtest package

The main package for specification testing of linear regressions in R is the lmtest package.

With it, you can:

- test for heteroskedasticity
- test for autocorrelation
- test functional form (eg Ramsey RESET test)
- discriminate between non-nested models and more

All of the tests covered here are from the lmtest package. As usual, you need to install and initialize the package:

```
## In the console:  install.packages("lmtest")
library(lmtest)
```

Testing for heteroskedasticity

Testing for heteroskedasticity in R can be done with the bptest() function from the lmtest to the model object.

- By default, using a regression object as an argument to bptest() will perform the Koenker-Bassett version of the Breusch-Pagan test (aka 'generalized' or 'studentized' Breusch-Pagan Test):

```
bptest(wages_reg)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  wages_reg
## BP = 22.974, df = 4, p-value = 0.0001282
```

Testing for heteroskedasticity ctd

- If you want the "standard" form of the Breusch-Pagan Test, just use:

```
bptest(myreg, studentize = FALSE)
```

- You can also perform the White Test of Heteroskedasticity using bptest() by manually specifying the regressors of the auxiliary regression inside of `bptest`:
  - That is, specify the distinct regressors from the main equation, their squares, and cross-products.

```
bptest(myreg, ~ x1 + x2 + x1*x2 + I(x1^2) +
         I(x2^2), data=mydata)
```

Functional form

The **Ramsey RESET Test** tests functional form by evaluating if higher order terms have any explanatory value.

```
resettest(wages_reg)
```

```
##
##  RESET test
##
## data:  wages_reg
## RESET = 7.1486, df1 = 2, df2 = 3287, p-value = 0.000798
```

Testing for autocorrelation: Breusch-Godfrey test

```
bgtest(wages_reg)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up
##
## data:  wages_reg
## LM test = 7.0938, df = 1, p-value = 0.007735
```

Testing for autocorrelation: Durbin-Watson test

```
dwtest(wages_reg)
```

```
##
##  Durbin-Watson test
##
## data:  wages_reg
## DW = 1.9073, p-value = 0.003489
## alternative hypothesis: true autocorrelation is greater
```

Specifying the variance structure

In practice, errors should *almost always* be specified in a manner that is heteroskedasticity and autocorrelation consistent.

- In Stata, you can pretty much always use the **robust** option.
- In R, you should more explicitly specify the variance structure.
  - The sandwich allows for specification of heteroskedasticity-robust, cluster-robust, and heteroskedasticity and autocorrelation-robust error structures.
  - These can then be used with t-tests [coeftest()] and F-tests [waldtest()] from lmtest.

Heteroskedasticity-robust errors

HC_1 Errors (MacKinnon and White, 1985): $\Sigma = \frac{n}{n-k} diag\{\hat{u_i}^2\}$

- Default heteroskedasticity-robust errors used by Stata with **robust**

HC_3 Errors (Davidson and MacKinnon, 1993): $\Sigma = diag\{(\frac{\hat{u}_i}{1-h_i})^2\}$

- Approximation of the jackknife covariance estimator
- Recommended in some studies over HC_1 because it is better at keeping nominal size with only a small loss of power in the presence of heteroskedasticity.

Heteroskedasticity-robust errors example

```
cex_reg <- lm(expenditures ~ hh_size + educ_ref +
               region, data=cex_data)
tidy(coeftest(cex_reg, vcov =
               vcovHC(cex_reg, type="HC1")))
```

term	estimate	std.error	statistic	p.value
(Intercept)	-553.26201	94.106216	-5.879123	0e+00
hh_size	-298.33622	14.262224	-20.917932	0e+00
educ_ref	109.46626	7.190421	15.223901	0e+00
region	83.15485	15.274695	5.443962	1e-07

Computing marginal effects

In linear regressions where the regressors and regressors are in "levels", the coefficients are of course equal to the marginal effects.

- But if the regression is nonlinear or a regressor enter in e.g. in logs or quadratics, then marginal effects may be more important than coefficients.
- You can use the package margins to get marginal effects.

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

Marginal effects example

We can get the Average Marginal Effects by using `summary` with **margins**:

```
summary(margins(wages_reg))
```

factor	AME	SE	z	p	lower
exper	0.1209297	0.0232234	5.207226	2e-07	0.0754126
schooling	0.6422357	0.0334052	19.225648	0e+00	0.5767628
sexfemale	-1.3390973	0.1077331	-12.429771	0e+00	-1.5502502

Further regression methods

Panel regression: first differences

The package plm provides a wide variety of estimation methods and diagnostics for panel data.

- We will cover two common panel data estimators, first-differences regression and fixed effects regression.
- To estimate first-differences estimator, use the plm() in the plm package.

```
library(plm)
```

Syntax:

```
myreg <- plm(y ~ x1 + x2 + x3, data = mydata,
            index=c("groupvar", "timevar"), model="fd")
```

Panel regression: fixed effects

Of course, in most cases fixed effects regression is a more efficient alternative to first-difference regression.

To use fixed effects regression, instead specify the argument **model = “within”**.

- Use the option **effect = “twoway”** to include group and year fixed effects.

```
myreg <- plm(y ~ x1 + x2 + x3, data = mydata,
             index=c("groupvar", "timevar"),
             model="within", effect = "twoway")
```

A crime example

```
crime_NC <- Crime %>% as.tibble() %>%
  select(county, year, crmrte, polpc, region, smsa,
  taxpc) %>% rename(crimerate=crmrte,
  police_pc = polpc, urban=smsa, tax_pc=taxpc)
crime_NC[1:2,]
```

county	year	crimerate	police_pc	region	urban	tax_pc
1	81	0.0398849	0.0017868	central	no	25.69763
1	82	0.0383449	0.0017666	central	no	24.87425

First differences regression on the crime dataset

```r
crime_reg <- plm(crimerate ~ police_pc + tax_pc +
                 region + urban, data=crime_NC,
                 index=c("county", "year"), model="fd")
tidy(crime_reg)
```

term	estimate	std.error	statistic	p.value
police_pc	2.0596639	0.1995562	10.3212212	0.0000000
tax_pc	0.0000068	0.0000486	0.1408233	0.8880622

Fixed effects regression on the crime dataset

```
crime_reg <- plm(crimerate ~ police_pc + police_pc +
                 tax_pc + urban, data=crime_NC,
                 index=c("county", "year"),
                 model="within", effect="twoway")
tidy(crime_reg)
```

term	estimate	std.error	statistic	p.value
police_pc	1.7782245	0.1437963	12.366272	0.0000000
tax_pc	0.0000627	0.0000450	1.391503	0.1646546

Instrumental variables regression

The most popular function for doing IV regression is the ivreg() in the AER package.

```
library(AER)
```

Syntax:

```
myivreg <- ivreg(y ~ x1 + x2 | z1 + z2 + z3,
                 data = mydata)
```

IV diagnostics

Three common diagnostic tests are available with the **summary** output for regression objects from ivreg().

- **Durbin-Wu-Hausman Test of Endogeneity:** Tests for endogeneity of suspected endogenous regressor under assumption that instruments are exogenous.
- **F-Test of Weak Instruments**: Typical rule-of-thumb value of 10 to avoid weak instruments, although you can compare again Stock and Yogo (2005) critical values for more precise guidance concerning statistical size and relative bias.
- **Sargan-Hansen Test of Overidentifying Restrictions:** In overidentified case, tests if some instruments are endogenous under the initial assumption that all instruments are exogenous.

IV regression example

Let's look at an IV regression from the seminal paper "The Colonial Origins of Comparative Development" by Acemogulu, Johnson, and Robinson (AER 2001)

```
col_origins <- import("./data/maketable5.dta") %>%
  as.tibble() %>% filter(baseco==1) %>%
  select(logpgp95, avexpr, logem4, shortnam) %>%
  rename(logGDP95 = logpgp95, country = shortnam,
    legalprotect = avexpr, log.settler.mort = logem4)

col_origins_iv <- ivreg(logGDP95 ~ legalprotect |
     log.settler.mort, data = col_origins)
```

IV regression example: estimates

```
IVsummary <- summary(col_origins_iv, diagnostics = TRUE)
IVsummary["coefficients"]
```

```
## $coefficients
##                Estimate Std. Error  t value     Pr(>|t|)
## (Intercept)   1.9096665  1.0267273 1.859955 6.763720e-02
## legalprotect 0.9442794  0.1565255 6.032753 9.798645e-08
```

IV regression example: diagnostics

```r
IVsummary["diagnostics"]
```

```
## $diagnostics
##                     df1 df2 statistic      p-value
## Weak instruments      1  62  22.94680 1.076546e-05
## Wu-Hausman            1  61  24.21962 6.852437e-06
## Sargan                0  NA        NA           NA
```

Further regression methods

Some useful functions for nonlinear regression include:

- **Quantile Regression:** rq() in the quantreg package.
- **Limited Dependent Variable Models:**
  - These models, such as logit and probit (binary choice), or Poisson (count model) are incorporated in R as specific cases of a *generalized linear model* (GLM).
  - GLM models are estimated in R using the glm() function in base R.
- **Regression Discontinutiy:**
  - RDD designs can easily be performed in R through a few different packages.
  - I suggest using the function rdrobust() from the package of the same name.

Graphs in R

Data visualizaiton overview

One of the strong points of R is creating very high-quality data visualization.

- R is very good at both "static" data visualization and interactive data visualization designed for web use.
- Today, I will be covering static data visualization, but here are a couple of good resources for interactive visualization: [1], [2]

ggplot2 for data visualization

The main package for publication-quality static data visualization in R is ggplot2, which is part of the tidyverse collection of packages.

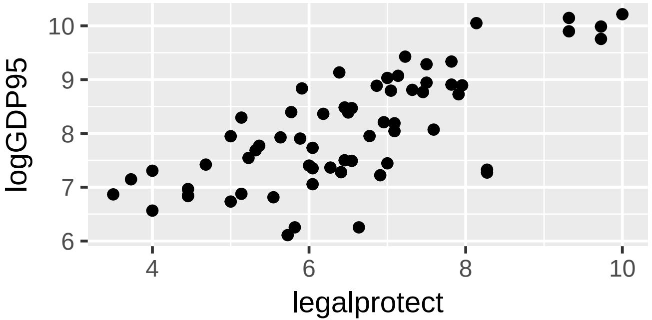
- The workhorse function of ggplot2 is ggplot(), response for creating a very wide variety of graphs.
- The "gg" stands for "grammar of graphics". In each ggplot() call, the appearance of the graph is determined by specifying:
  - The **data**(frame) to be used.
  - The **aes**(thetics)s of the graph — like size, color, x and y variables.
  - The **geom**(etry) of the graph — type of data to be used.

```
mygraph <- ggplot(mydata, aes(...)) + geom(...) + ...
```

Scatterplots

First, let's look at a simple scatterplot, which is defined by using the geometry **geom_point()**.

```
ggplot(col_origins, aes(x=legalprotect,
            y = logGDP95)) + geom_point()
```

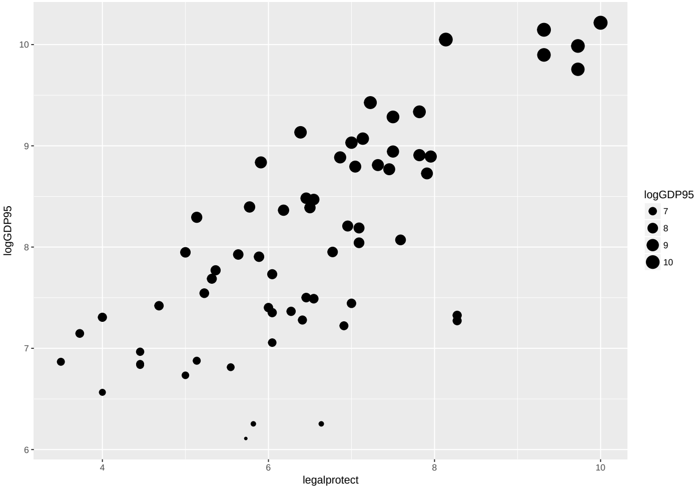


Adding an aesthetic option to the points

Graphs can be extensively customized using additional arguments inside of elements:

```
ggplot(col_origins, aes(x=legalprotect,y = logGDP95)) +
  geom_point(aes(size=logGDP95))
```

Adding an aesthetic option to the points

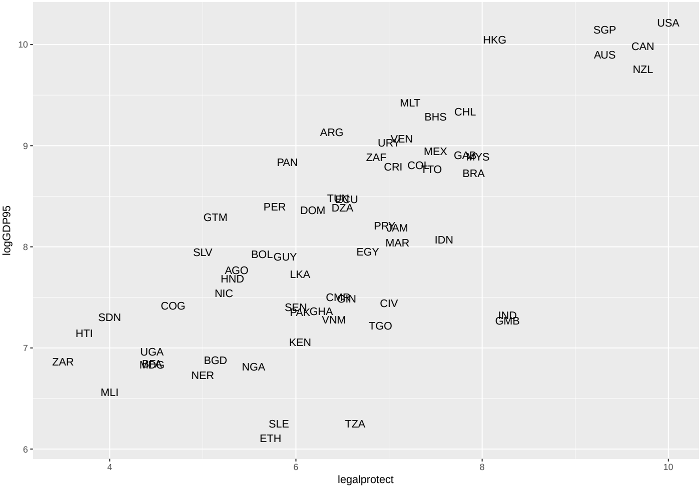


Using country names instead of points

Instead of using a scatter plot, we could use the names of the data points in place of the dots.

```
ggplot(col_origins,
       aes(x=legalprotect, y = logGDP95,
       label=country)) +  geom_text()
```

Using country names instead of points ctd

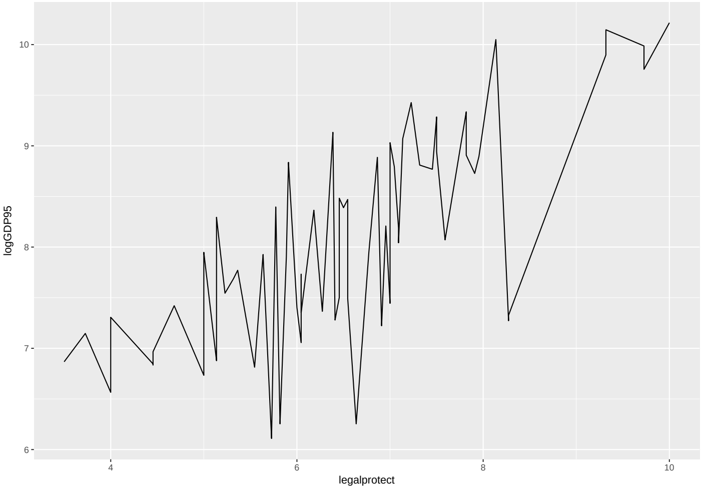


Line graph

A line graph uses the geometry **geom_line()**.

```
ggplot(col_origins, aes(x=legalprotect,
            y = logGDP95)) + geom_line()
```

Line graph



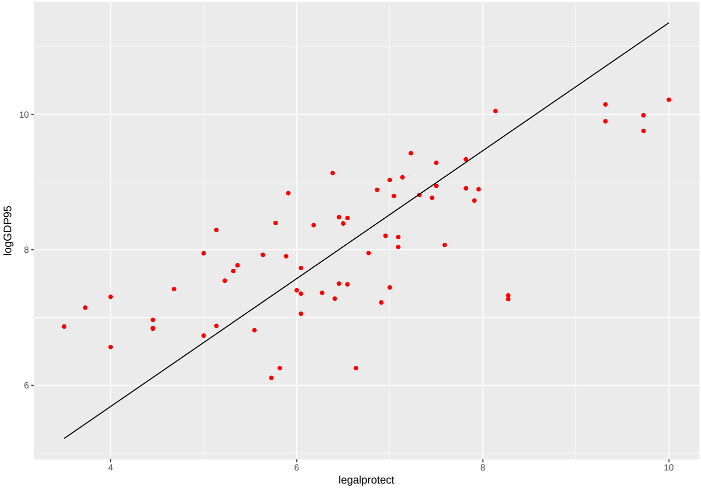
Plotting a regression line

A more useful line is the fitted values from the regression. Here's a plot of that line with the points from the scatterplot for the Acemoglu IV:

```
IV_fitted <- tibble(col_origins$legalprotect,
                    fitted(col_origins_iv))
colnames(IV_fitted) <- c("legalprotect", "hat")

ggplot(col_origins, aes(x=legalprotect,
  y = logGDP95))  +  geom_point(color="red")  +
  geom_line(data = IV_fitted, aes(x=legalprotect,
                                   y=hat))
```

Plotting a regression line

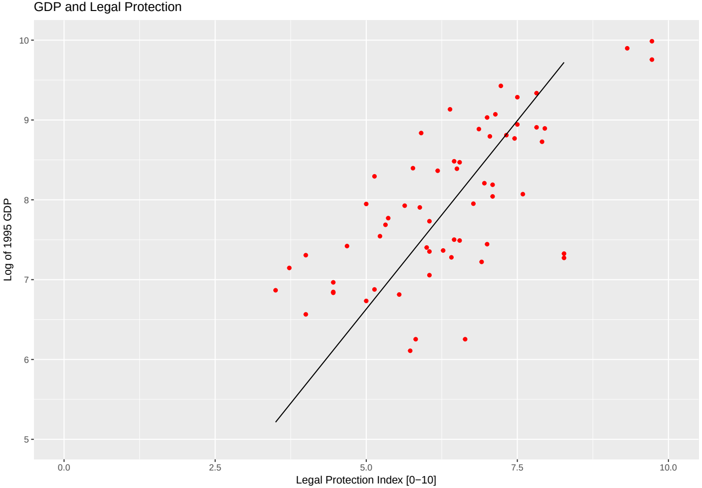


Specifying axis and titles

A standard task in making the graph is specifying graph titles (main and axes), as well as potentially specifying the scale of the axes.

```
ggplot(col_origins,
  aes(x=legalprotect, y = logGDP95))  +
  geom_point(color="red") +
  geom_line(data = IV_fitted,
  aes(x=legalprotect, y=hat))  +
  ggtitle("GDP and Legal Protection") +
  xlab("Legal Protection Index [0-10]") +
  ylab("Log of 1995 GDP") +
  xlim(0, 10) + ylim(5,10)
```

Specifying axis and titles

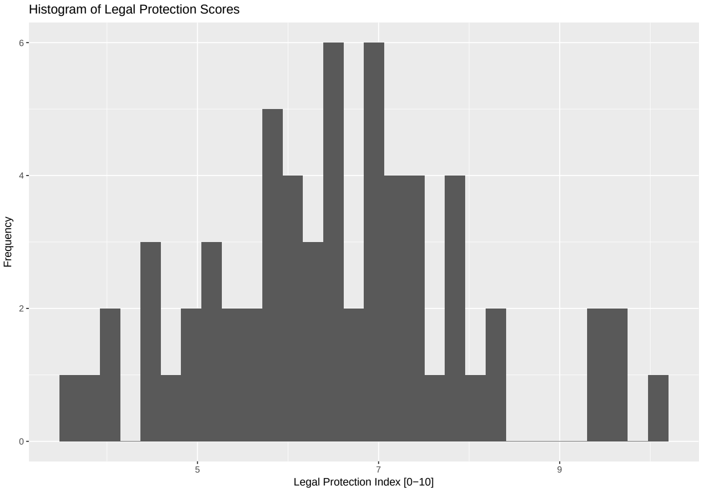


Histogram

The geometry point for histogram is **geom_histogram()**.

```
ggplot(col_origins, aes(x=legalprotect)) +
  geom_histogram() +
  ggtitle("Histogram of Legal Protection Scores") +
  xlab("Legal Protection Index [0-10]") +
  ylab("Frequency")
```

Histogram


Histogram of Legal Protection Scores
Frequency
6
4
2
0
5
7
9
Legal Protection Index [0−10]

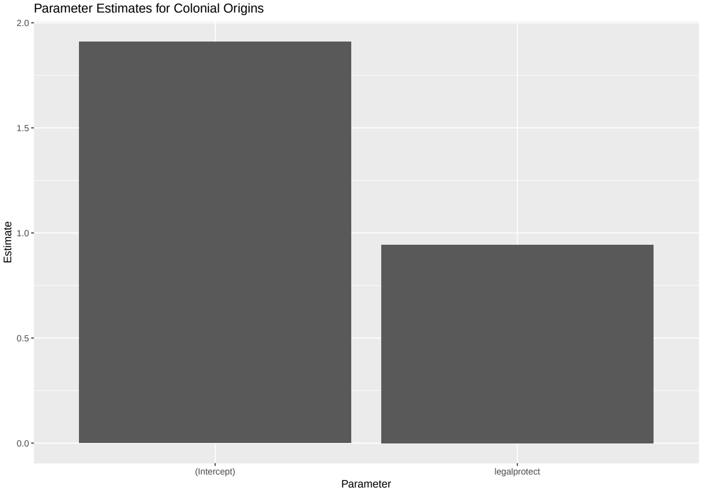
Bar plot

The geometry for a bar plot is **geom_bar()**. By default, a bar plot uses frequencies for its values, but you can use values from a column by specifying **stat = "identity"** inside **geom_bar()**.

```
coeffs_IV <- tidy(col_origins_iv)

ggplot(coeffs_IV,
  aes(x=term, y=estimate)) +
  geom_bar(stat = "identity") +
  ggtitle("Parameter Estimates for Colonial Origins") +
  xlab("Parameter") + ylab("Estimate")
```

Bar plot

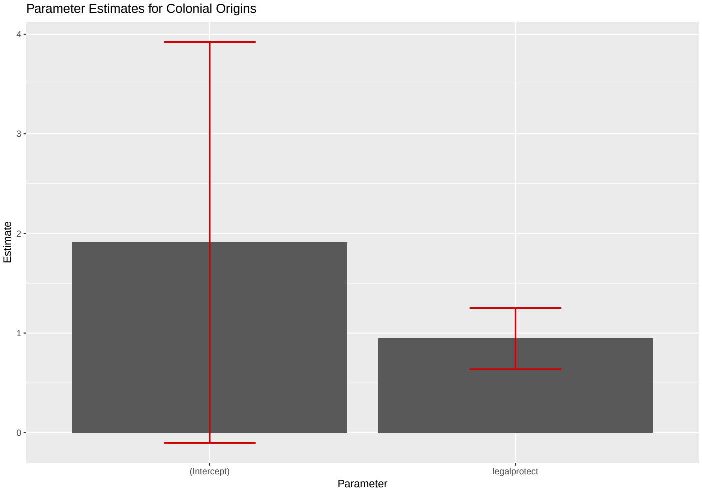

Parameter Estimates for Colonial Origins
2.0
1.5
1.0
0.5
0.0
Estimate
(Intercept)
legalprotect
Parameter

Adding error bars

You can easily add error bars by specifying the values for the error bar inside of **geom_errorbar()**.

```
ggplot(coeffs_IV,
  aes(x=term, y=estimate)) +
  geom_bar(stat = "identity") +
  ggtitle("Parameter Estimates for Colonial Origins") +
  xlab("Parameter") + ylab("Estimate") +
  geom_errorbar(aes(ymin=estimate - 1.96 * std.error,
                    ymax=estimate + 1.96 * std.error),
                    size=.75, width=.3, color="red3")
```

Adding error bars

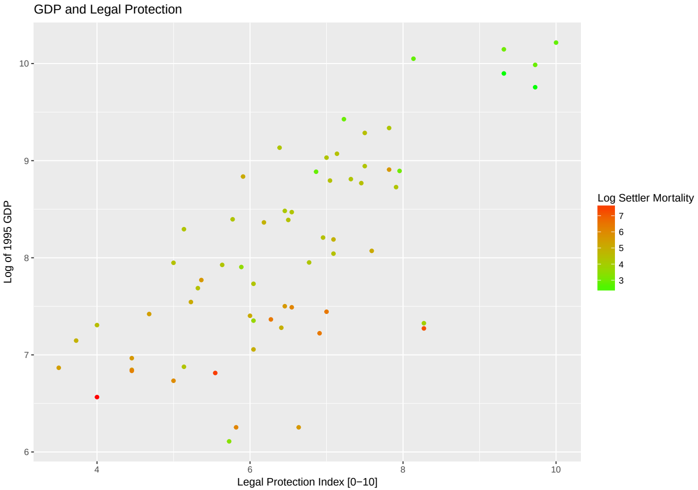


Adding colors

You can easily add color to graph points as well. There are a lot of aesthetic options to do that — here I demonstrate adding a color *scale* to the graph.

```
ggplot(col_origins, aes(x=legalprotect,
  y = logGDP95 , col= log.settler.mort)) +
  geom_point() +
  ggtitle("GDP and Legal Protection") +
  xlab("Legal Protection Index [0-10]") +
  ylab("Log of 1995 GDP") +
  scale_color_gradient(low="green",high="red3",
                       name="Log Settler Mortality")
```

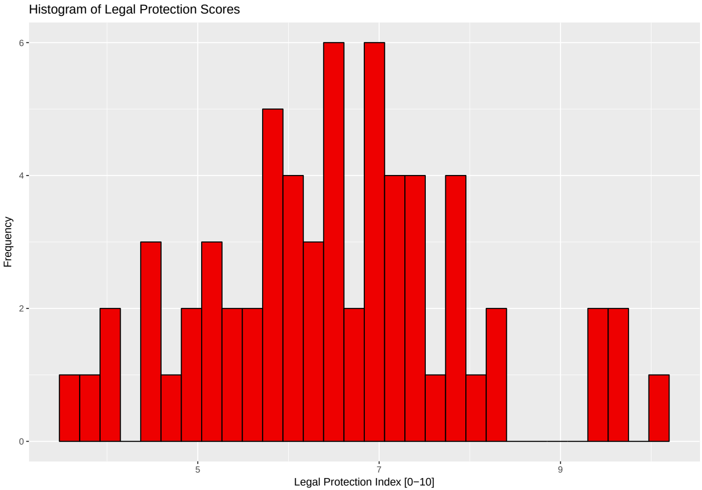
Adding colors



Adding colors: example 2

```
ggplot(col_origins, aes(x=legalprotect)) +
  geom_histogram(col="black", fill="red2") +
  ggtitle("Histogram of Legal Protection Scores") +
  xlab("Legal Protection Index [0-10]") +
  ylab("Frequency")
```

Adding colors: example 2



Adding themes

Another option to affect the appearance of the graph is to use **themes**, which affect a number of general aspects concerning how graphs are displayed.

- Some default themes come installed with ggplot2/tidyverse, but some of the best in my opinion come from the package ggthemes.

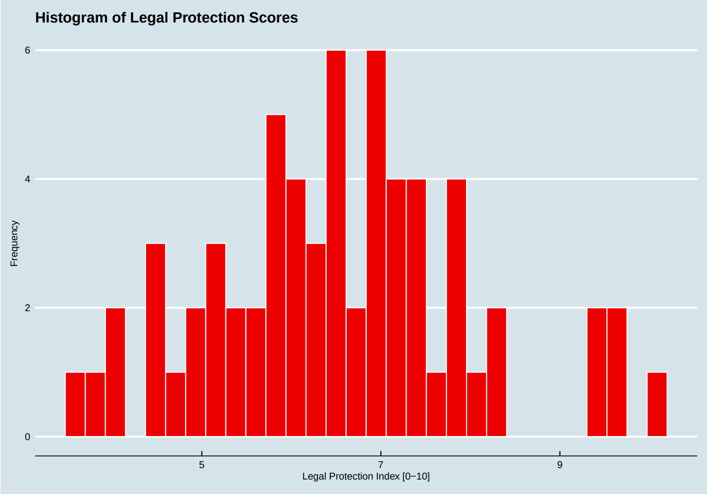
```
library(ggthemes)
```

Adding themes

- To apply a theme, just add **+ themename()** to your ggplot graphic.

```
ggplot(col_origins, aes(x=legalprotect)) +
  geom_histogram(col="white", fill="red2") +
  ggtitle("Histogram of Legal Protection Scores") +
  xlab("Legal Protection Index [0-10]") +
  ylab("Frequency") +
  theme_economist()
```

Adding themes



More with ggplot2

This has been just a small overview of things you can do with ggplot2. To learn more about it, here are some useful references:

The ggplot2 website:

- Very informative although if you don't know what you're looking for, you can be a bit inundated with information.

STHDA Guide to ggplot2:

- A bit less detailed, but a good general guide to ggplot2 that is still pretty thorough.

RStudio's ggplot2 cheat sheet:

- As with all the cheat sheets, very concise but a great short reference to main options in the package.