First Regression Example

Steve Buyske

We'll take an in-depth look at a Bayesian data analysis of per-country vital statistics.

Set up.

First load some required packages. You can run the entire chunk by clicking on the green arrow below.

The data.

We will work with the data frame `mini_gapminder`, which is already loaded in your workspace and is drawn from data collected by Gapminder.

Typing the name of the data frame will show some of the data, while the `glimpse()` function or the `skim()` function will show the first few entries or a summary, respectively. You can see all of the data, spreadsheet style, with the `View()` function; it will open a new tab.

Try this chunk line by line, by putting the cursor on the line and hitting `Cmd+Enter` or `Control+Enter`

```
mini_gapminder <- read_csv("mini_gapminder.csv")

mini_gapminder
```

```
## # A tibble: 177 x 10
##    country life_expectancy population gdp_per_capita  gini continent
##    <chr>             <dbl>      <dbl>          <dbl> <dbl> <chr>
##  1 Albania            78      2930000          12400  29   Europe
##  2 Algeria            77.9   42000000          13700  27.6 Africa
##  3 Angola             65.2   30800000           5850  42.6 Africa
##  4 Antigu~            77.6     103000          21000  40   Americas
##  5 Argent~            77     44700000          18900  42.4 Americas
##  6 Armenia            76      2930000           8660  32.6 Asia
##  7 Austra~            82.9   24800000          45800  32.3 Oceania
##  8 Austria            81.8    8750000          44600  30.5 Europe
##  9 Azerba~            72.3    9920000          16600  32.4 Asia
## 10 Bahamas            74.1     399000          21900  43.7 Americas
## # ... with 167 more rows, and 4 more variables: infant_mortality <dbl>,
## #   fertility <dbl>, region <chr>, log10_gdp_per_capita <dbl>
```

```
mini_gapminder %>% glimpse()
```

```
## Rows: 177
## Columns: 10
## $ country              <chr> "Albania", "Algeria", "Angola", "Antigua and B...
## $ life_expectancy      <dbl> 78.0, 77.9, 65.2, 77.6, 77.0, 76.0, 82.9, 81.8...
## $ population           <dbl> 2.93e+06, 4.20e+07, 3.08e+07, 1.03e+05, 4.47e+...
## $ gdp_per_capita       <dbl> 12400, 13700, 5850, 21000, 18900, 8660, 45800,...
```

```
## $ gini                 <dbl> 29.0, 27.6, 42.6, 40.0, 42.4, 32.6, 32.3, 30.5...
## $ continent            <chr> "Europe", "Africa", "Africa", "Americas", "Ame...
## $ infant_mortality     <dbl> 12.5, 21.9, 96.0, 5.8, 11.1, 12.6, 3.0, 2.9, 2...
## $ fertility            <dbl> 1.78, 2.71, 5.65, 2.06, 2.15, 1.41, 1.88, 1.50...
## $ region               <chr> "Southern Europe", "Northern Africa", "Middle ...
## $ log10_gdp_per_capita <dbl> 4.093422, 4.136721, 3.767156, 4.322219, 4.2764...
```

```
mini_gapminder %>% skim_without_charts()
```

Table 1: Data summary

Name	Piped data
Number of rows	177
Number of columns	10
Column type frequency:	
character	3
numeric	7
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
country	0	1	3	30	0	177	0
continent	0	1	4	8	0	5	0
region	0	1	9	25	0	22	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
life_expectancy	0	1	72.92	7.17	51.10	6.73e+01	7.43e+01	7.82e+01	8.42e+01
population	0	1	42067464.41	151675057.29	95200.00	2.69e+06	9.42e+06	3.05e+07	1.42e+09
gdp_per_capita	0	1	18527.24	19732.47	689.00	3.83e+03	1.24e+04	2.55e+04	1.21e+05
gini	0	1	38.80	7.66	25.00	3.30e+01	3.84e+01	4.31e+01	6.30e+01
infant_mortality	0	1	23.60	21.83	1.50	5.90e+00	1.44e+01	3.55e+01	9.60e+01
fertility	0	1	2.77	1.34	1.30	1.78e+00	2.26e+00	3.66e+00	7.51e+00
log10_gdp_per_capita	0	1	4.01	0.52	2.84	3.58e+00	4.09e+00	4.41e+00	5.08e+00

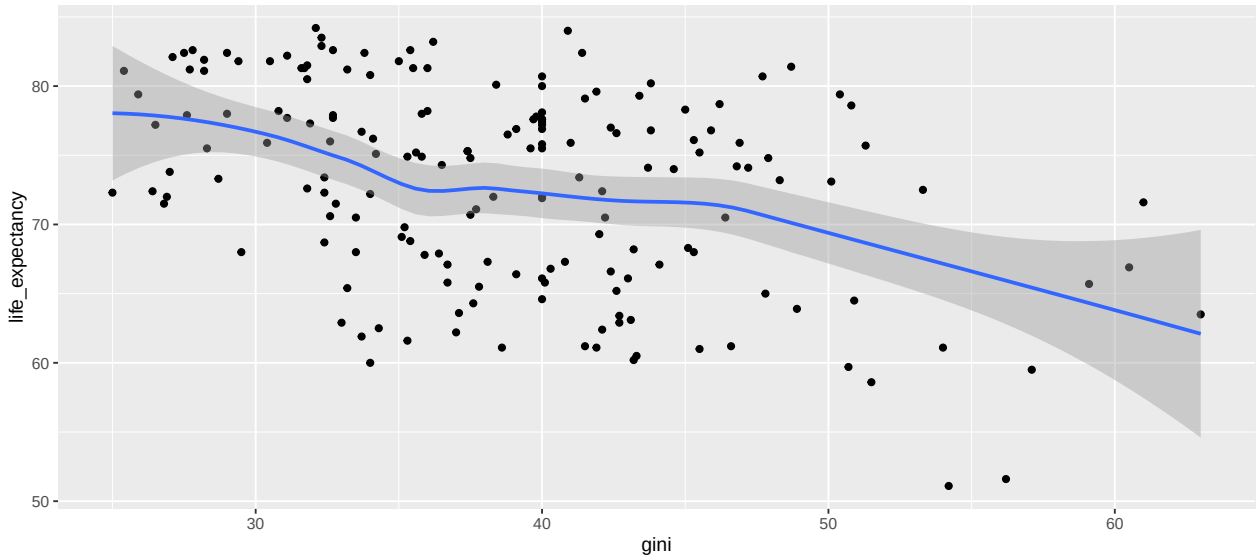
```
# View(mini_gapminder) # Comment this line out, by putting a hashmark at the start, if you want to knit
```

Traditional analysis.

We will look at the regression of `life_expectancy` against `gini`, which is the Gini index, a measure of economic inequality (higher Gini index = more inequality). Let's start with a scatterplot and then a quick traditional analysis.

```
mini_gapminder %>%
  ggplot(aes(gini, life_expectancy)) +
```

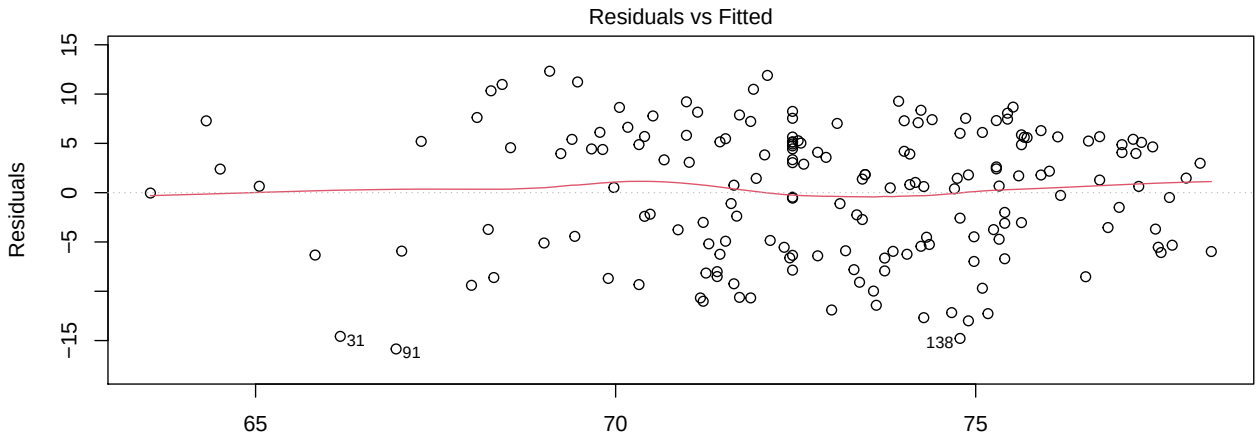
```
  geom_point() +
  geom_smooth()
```

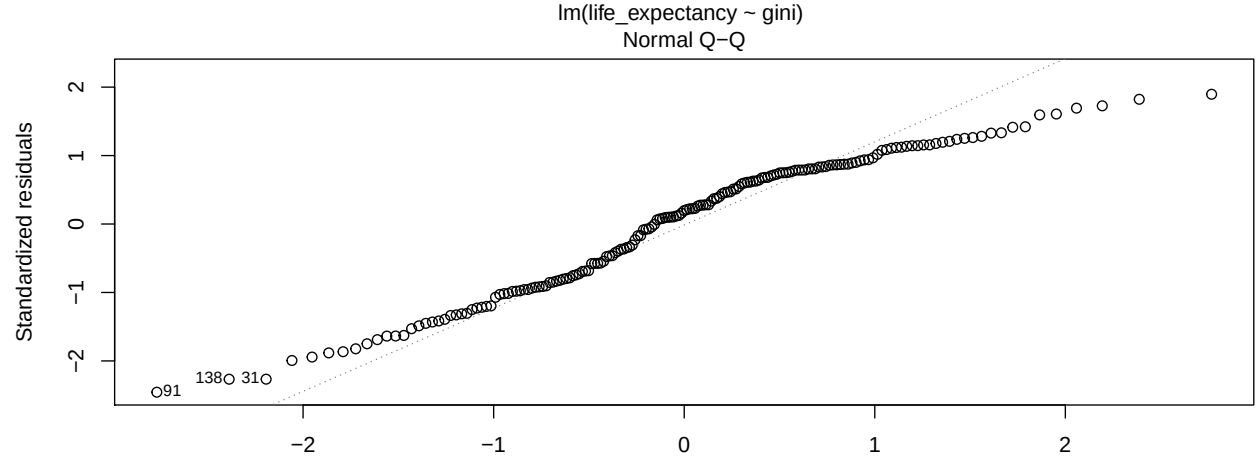


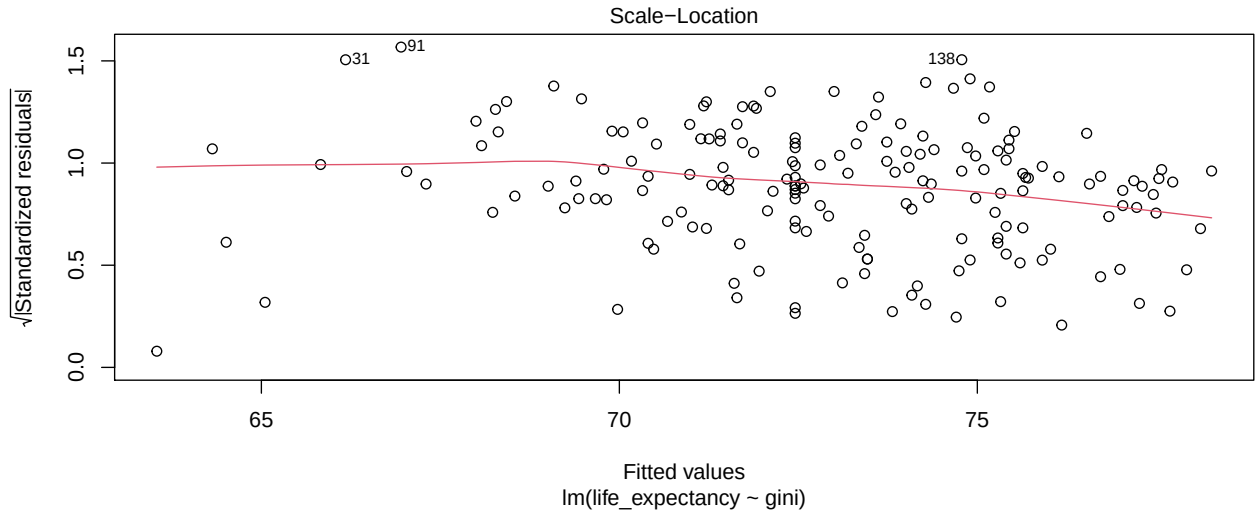

```
gini_lm <- lm(life_expectancy ~ gini, data = mini_gapminder)

summary(gini_lm)
```

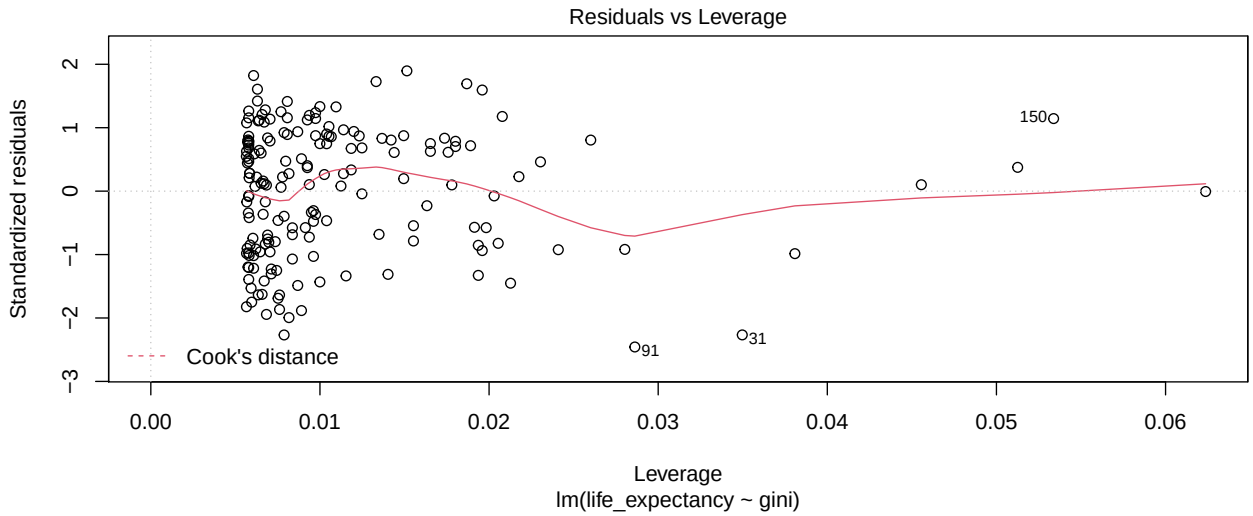
```
##
## Call:
## lm(formula = life_expectancy ~ gini, data = mini_gapminder)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.851  -5.440   1.278   5.234  12.316
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 87.96480    2.54734  34.532  < 2e-16 ***
## gini        -0.38770    0.06442  -6.019 1.01e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.546 on 175 degrees of freedom
## Multiple R-squared:  0.1715, Adjusted R-squared:  0.1668
## F-statistic: 36.22 on 1 and 175 DF,  p-value: 1.006e-08
```

```
plot(gini_lm)
```


Residuals vs Fitted
Residuals
15
10
5
0
−5
−15
31
91
138
65
70
75
Fitted values
lm(life_expectancy ~ gini)


Normal Q−Q
Standardized residuals
2
1
0
−1
−2
91
138
31
−2
−1
0
1
2
Theoretical Quantiles
lm(life_expectancy ~ gini)


Scale−Location
√|Standardized residuals|
1.5
1.0
0.5
0.0
31
91
138
65
70
75
Fitted values
lm(life_expectancy ~ gini)



The QQ plot looks a bit underdispersed, but otherwise everything looks okay.

Bayesian analysis.

We will use the `stan_glm()` function in the `rstanarm` package for our analysis. (There is also a `stan_lm()` function that we could use, but `stan_glm()` is better for our purposes.) We will mostly use the defaults.

```
gini_stan <- stan_glm(life_expectancy ~ gini, data = mini_gapminder)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.36 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.027369 seconds (Warm-up)
## Chain 1:                0.041672 seconds (Sampling)
## Chain 1:                0.069041 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
```

```
## Chain 2: Gradient evaluation took 1.4e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.029333 seconds (Warm-up)
## Chain 2:                0.039857 seconds (Sampling)
## Chain 2:                0.06919 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1.4e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.028439 seconds (Warm-up)
## Chain 3:                0.040416 seconds (Sampling)
## Chain 3:                0.068855 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
```

```
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.041312 seconds (Warm-up)
## Chain 4:                0.041108 seconds (Sampling)
## Chain 4:                0.08242 seconds (Total)
## Chain 4:
```

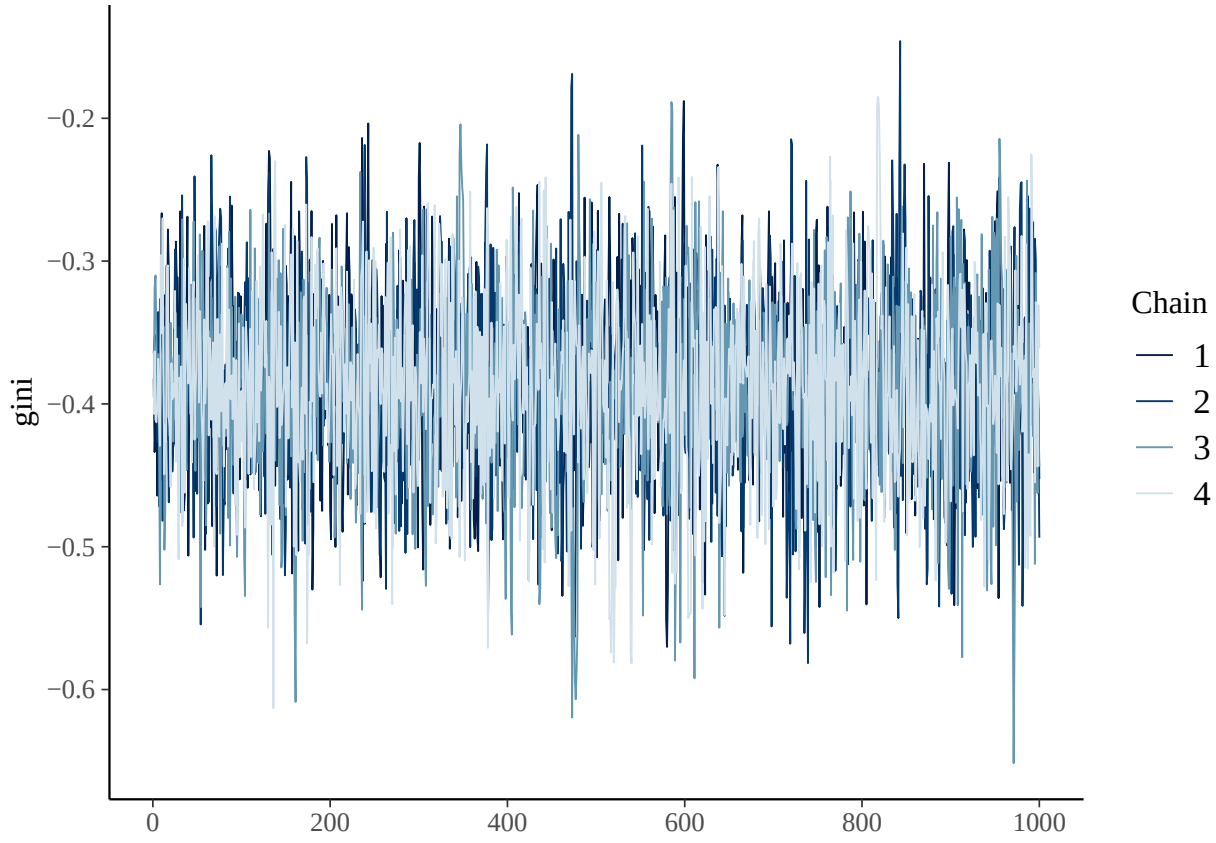
```
summary(gini_stan, digits = 4)
```

```
##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      life_expectancy ~ gini
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 177
##  predictors:   2
##
## Estimates:
##               mean    sd      10%     50%     90%
## (Intercept) 87.9569  2.5625 84.6557 87.9607 91.2337
## gini        -0.3873  0.0657 -0.4722 -0.3868 -0.3026
## sigma        6.5799  0.3469  6.1428  6.5665  7.0261
##
## Fit Diagnostics:
##            mean    sd      10%     50%     90%
## mean_PPD 72.9366  0.7047 72.0344 72.9429 73.8172
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##               mcse   Rhat   n_eff
## (Intercept)   0.0418 1.0003 3765
## gini          0.0011 1.0006 3774
## sigma         0.0056 0.9997 3850
## mean_PPD      0.0109 0.9995 4170
## log-posterior 0.0296 1.0010 1720
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

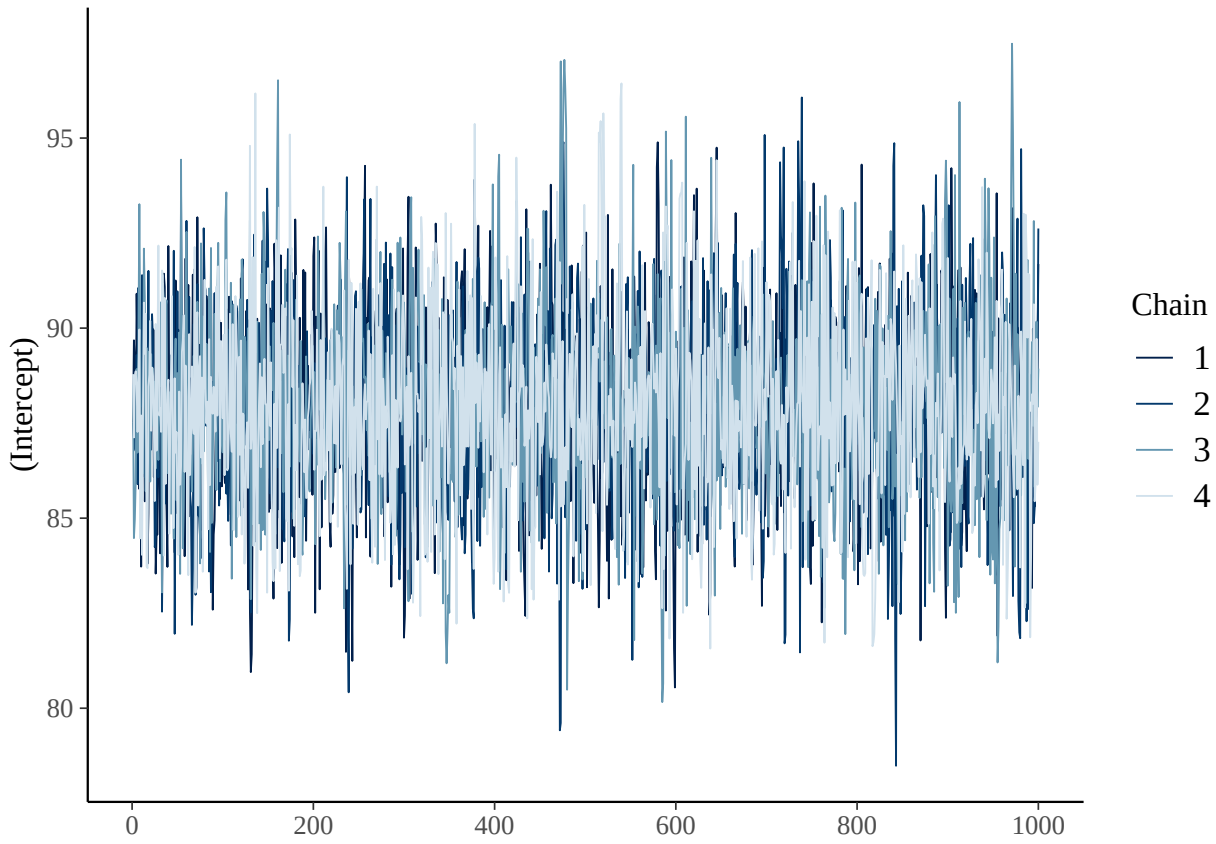
For MCMC diagnostics, we would like to see the `Rhat` statistic be less than 1.1 (we'll discuss what it measures

later in class). For visual diagnostics, we will look at the trace lines and the posterior predictive check.

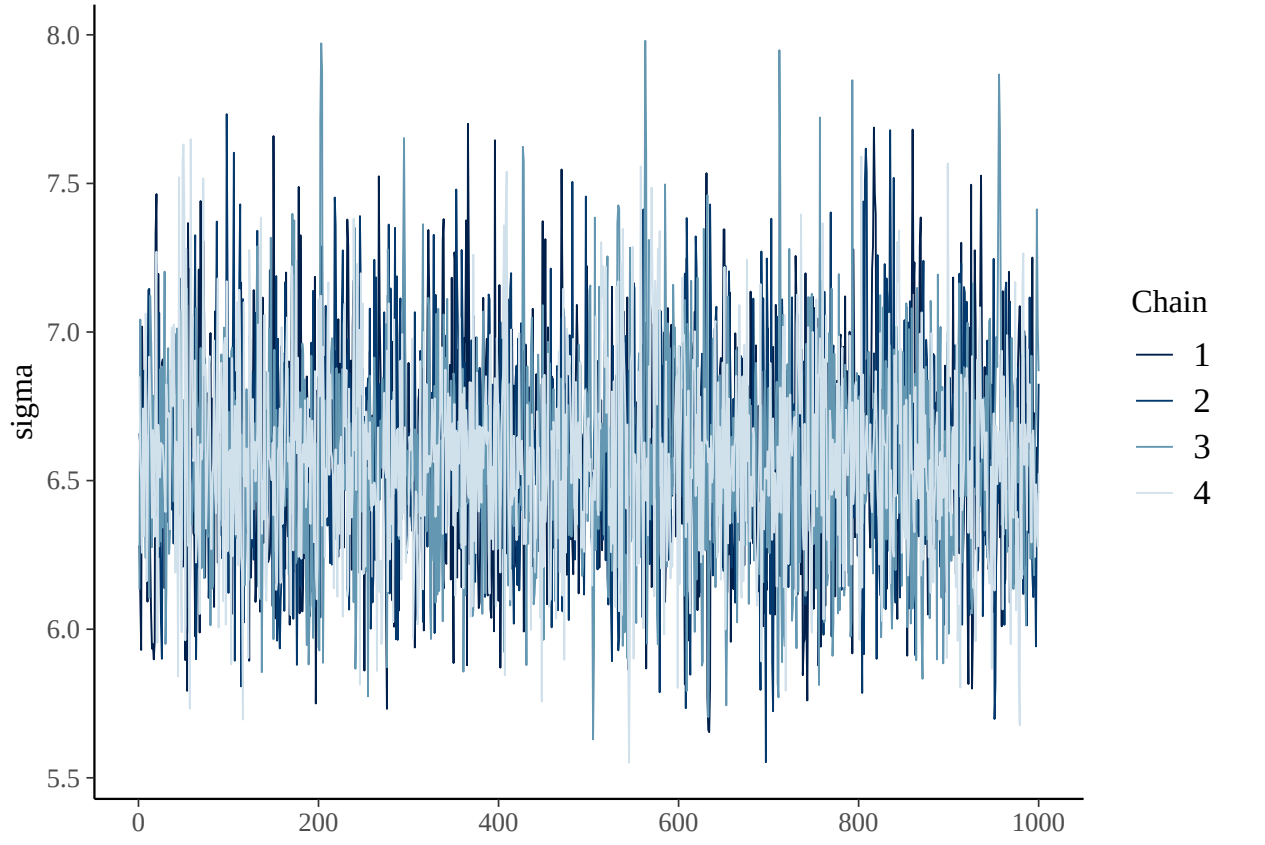
```
plot(gini_stan, plotfun = "trace", pars = "gini")
```



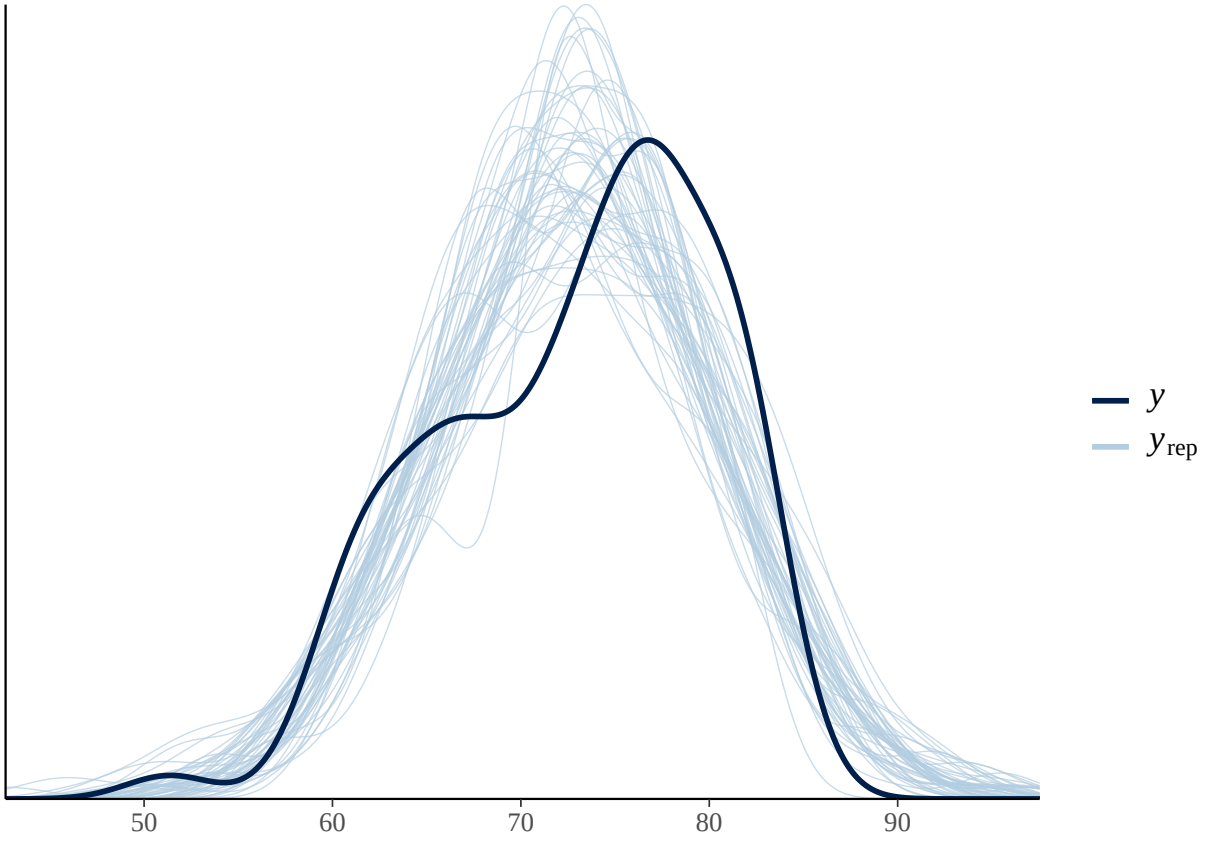

```
plot(gini_stan, plotfun = "trace", pars = "(Intercept)")
```




```
plot(gini_stan, plotfun = "trace", pars = "sigma")
```




```
pp_check(gini_stan)
```



The trace plots look good, but the posterior predictive check plot is not very good. In the the posterior predictive check plot, the black line is the density of the outcome variable, while each blue line comes from drawing a set of parameters from the posterior and then simulating the outcome variable according to those parameters. In a good fit, the black curve and the blue curves would look similar. In this case, though, the side-hump in the density of the original data is not reflected in the posterior predictive draws. We'll ignore that for now.

Now let's look at some of the model summaries. First, a look at the priors (we're not interested in studying these carefully right now—we're just trying to see how the code works).

```
summary(gini_stan, digits = 4)
```

```
##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      life_expectancy ~ gini
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 177
##  predictors:   2
##
## Estimates:
##               mean    sd      10%     50%     90%
## (Intercept) 87.9569  2.5625 84.6557 87.9607 91.2337
## gini        -0.3873  0.0657 -0.4722 -0.3868 -0.3026
## sigma        6.5799  0.3469  6.1428  6.5665  7.0261
```

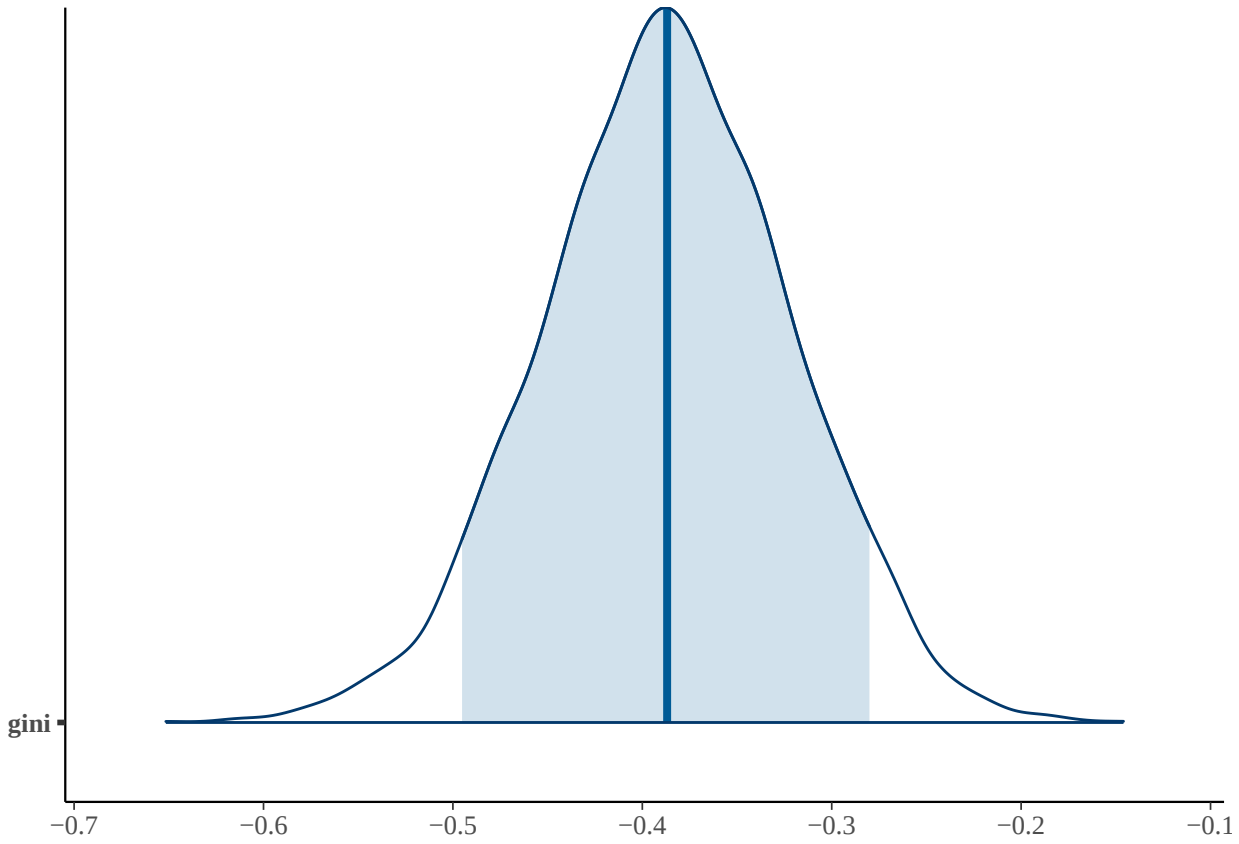
```
##
## Fit Diagnostics:
##            mean    sd      10%     50%     90%
## mean_PPD 72.9366  0.7047 72.0344 72.9429 73.8172
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##               mcse   Rhat   n_eff
## (Intercept)   0.0418 1.0003 3765
## gini          0.0011 1.0006 3774
## sigma         0.0056 0.9997 3850
## mean_PPD      0.0109 0.9995 4170
## log-posterior 0.0296 1.0010 1720
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

```
prior_summary(gini_stan)
```

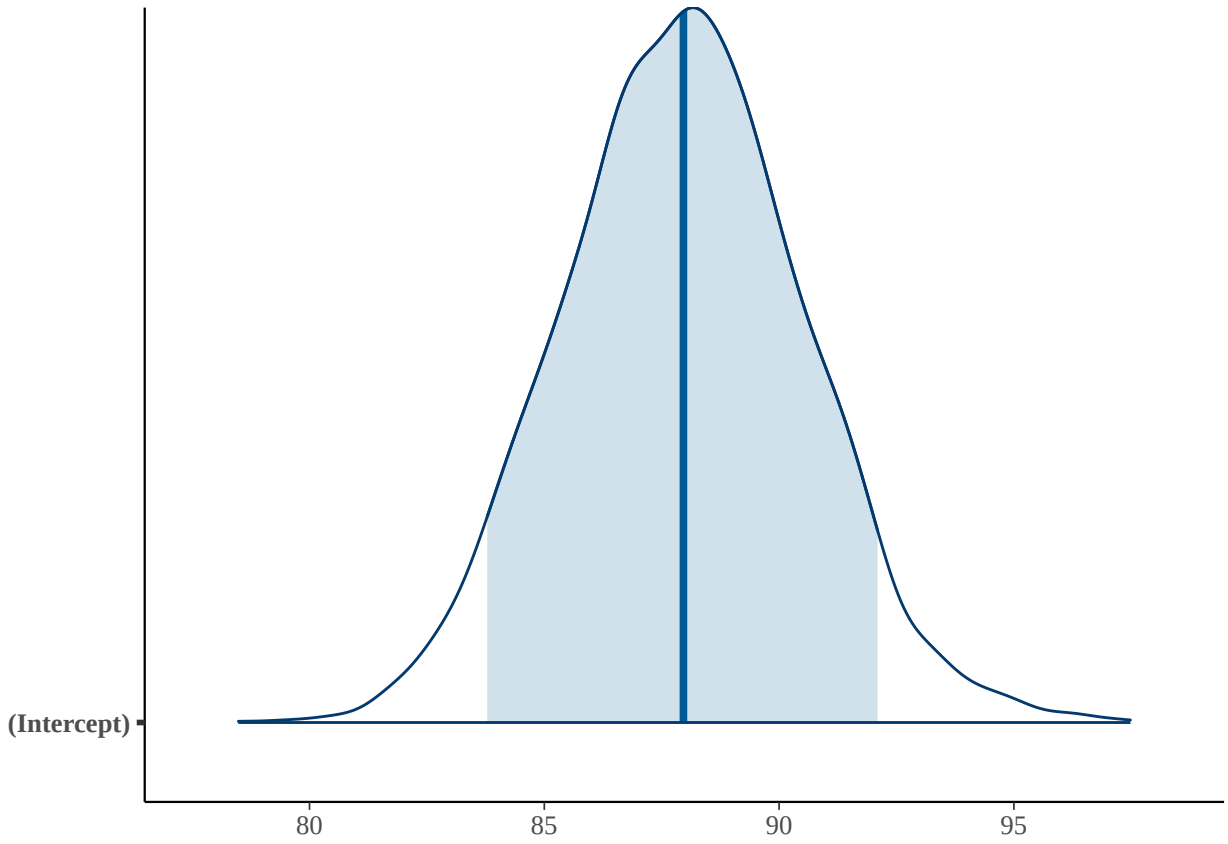
```
## Priors for model 'gini_stan'
## ------
## Intercept (after predictors centered)
##   Specified prior:
##     ~ normal(location = 73, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 73, scale = 18)
##
## Coefficients
##   Specified prior:
##     ~ normal(location = 0, scale = 2.5)
##   Adjusted prior:
##     ~ normal(location = 0, scale = 2.3)
##
## Auxiliary (sigma)
##   Specified prior:
##     ~ exponential(rate = 1)
##   Adjusted prior:
##     ~ exponential(rate = 0.14)
## ------
## See help('prior_summary.stanreg') for more details
```

Next let's look at the posteriors, along with the median and the 90% equal tails credible intervals indicated. (Your plots will look slightly different, since there is a random element and the default does not use a particularly large number of MCMC samples.)

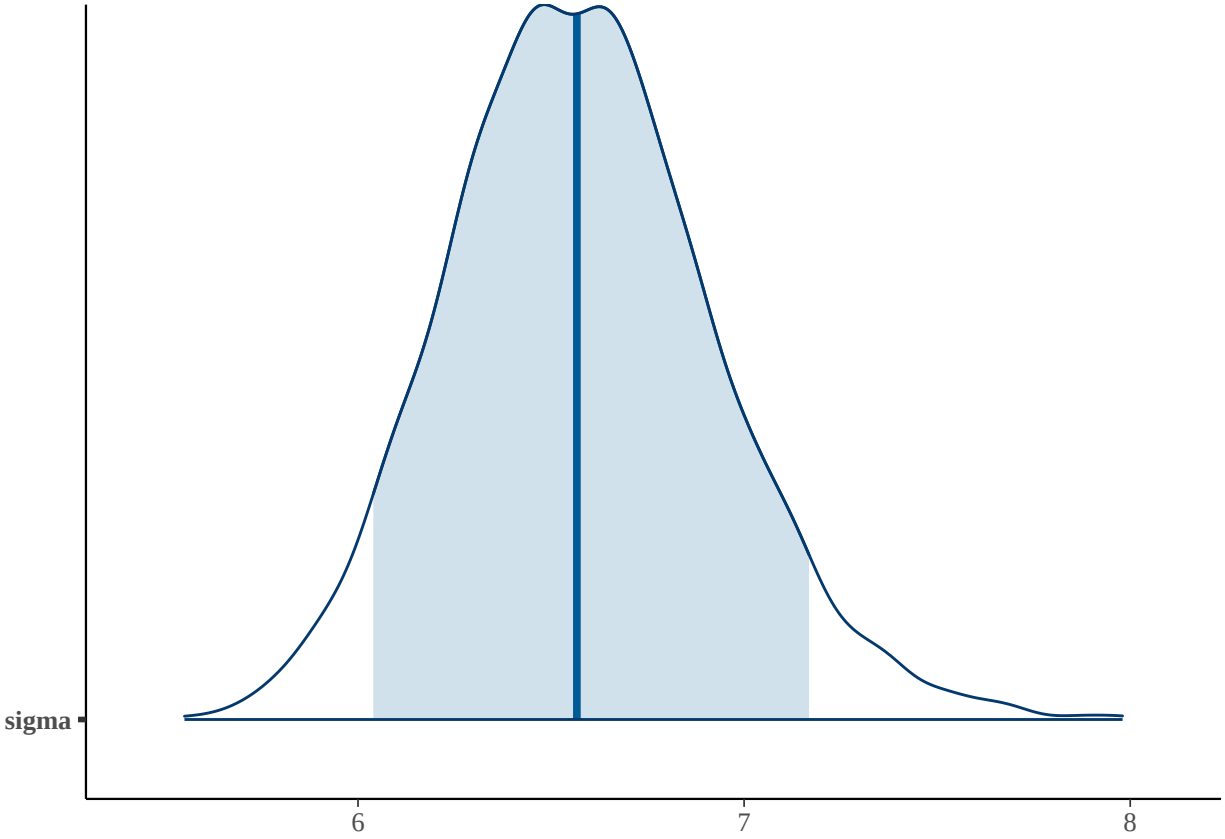
```
mcmc_areas(gini_stan, pars = "gini", prob = 0.9)
```




```
mcmc_areas(gini_stan, pars = "(Intercept)", prob = 0.9)
```

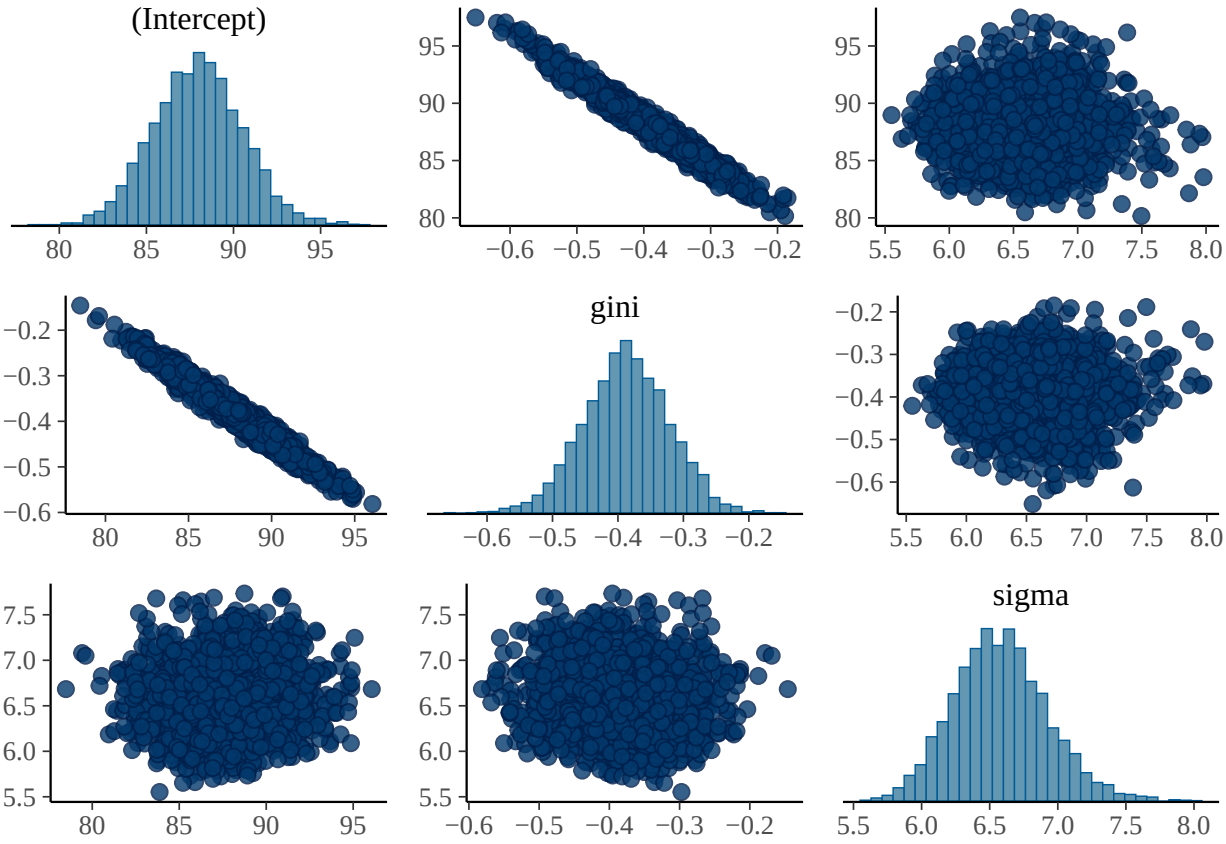



```
mcmc_areas(gini_stan, pars = "sigma", prob = 0.9)
```



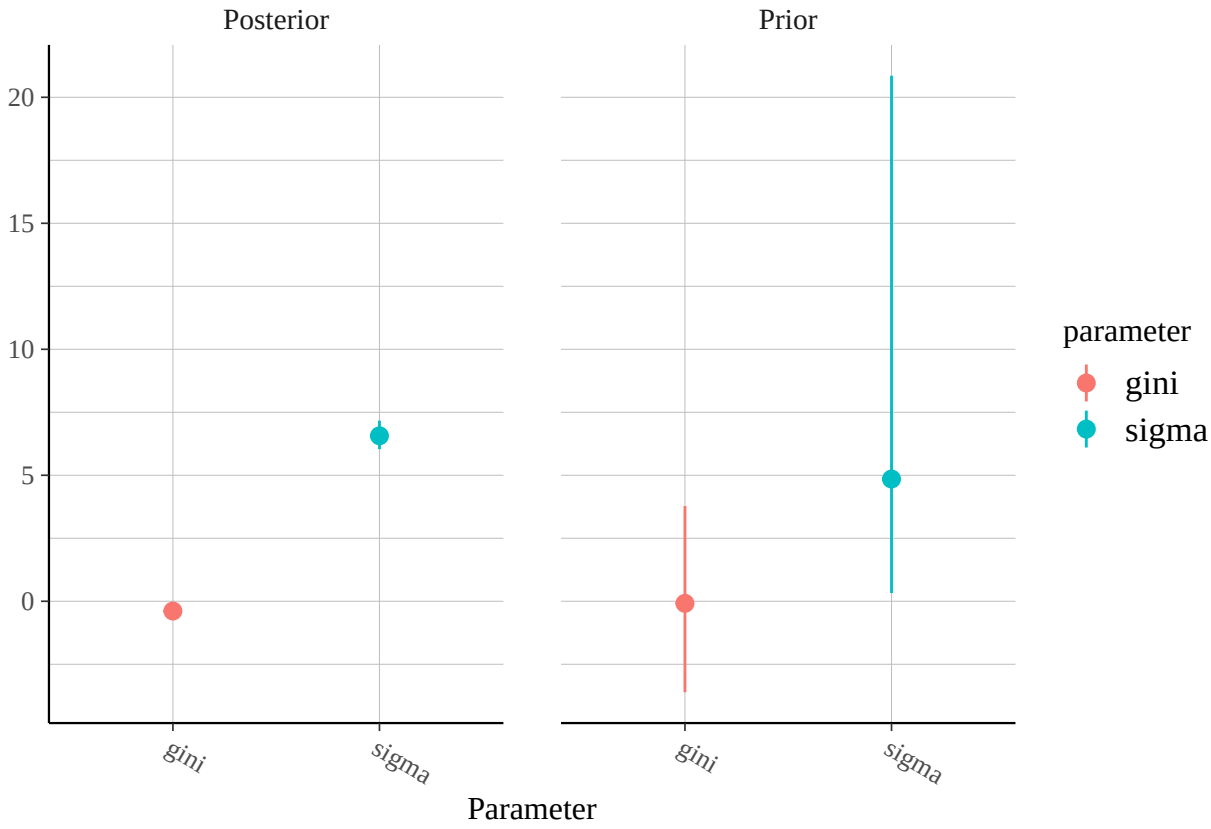
Now some pairwise plots of the parameters in the posterior—you can see that they are not independent (even though they were in the prior).

```
mcmc_pairs(gini_stan)
```



Finally, a comparison of the medians and 90% central intervals of the prior and the posterior.

```
posterior_vs_prior(gini_stan, pars = c("gini", "sigma"))
```



Notice how much less spread there is in the posteriors than the priors; it indicates that the posterior is mostly determined by the data rather the prior.

There is a nice interactive tool we can use as well, called `shinystan`. If the pop-up window is blocked, try the instructions at https://www.isc.upenn.edu/how-to/configuring-your-web-browser-allow-pop-windows

```
# launch_shinystan(gini_stan)  # Comment this line out, by putting a hashmark at the start,
                               # if you want to knit the whole file
```