Simulation_DS1

Jiafei Li

6/18/2020

Simulation: SLR for one n

In writing functions we wrote a short function to simulate data from a simple linear regression, fit the regression model, and return estimates of regression coefficients. Specifically, we generate data from $y_i = 0 + \beta_1 x_i + e_i$

```
sim_regression = function(n, beta0 = 2, beta1 = 3) {

  #generate simulation data
  sim_data = tibble(
    x = rnorm(n, mean = 1, sd = 1),
    y = beta0 + beta1 * x + rnorm(n, 0, 1)
  )

  # fit data with linear model
  ls_fit = lm(y ~ x, data = sim_data)

  tibble(
    beta0_hat = coef(ls_fit)[1],
    beta1_hat = coef(ls_fit)[2]
  )
}
```

rerun using for loop

```
output = vector("list", 1000)

for (i in 1:1000) {
  output[[i]] = sim_regression(n = 30)
}

sim_results = bind_rows(output) # bind_rows: useful function to combine the rows
```

rerun simulation using 'purrr'

Taking a look at the for loop we used to create these results, you might notice that there's no input list – the sequence is used to keep track of the output but doesn't affect the computation performed inside the for loop. In cases like these, the purrr::rerun function is very handy.

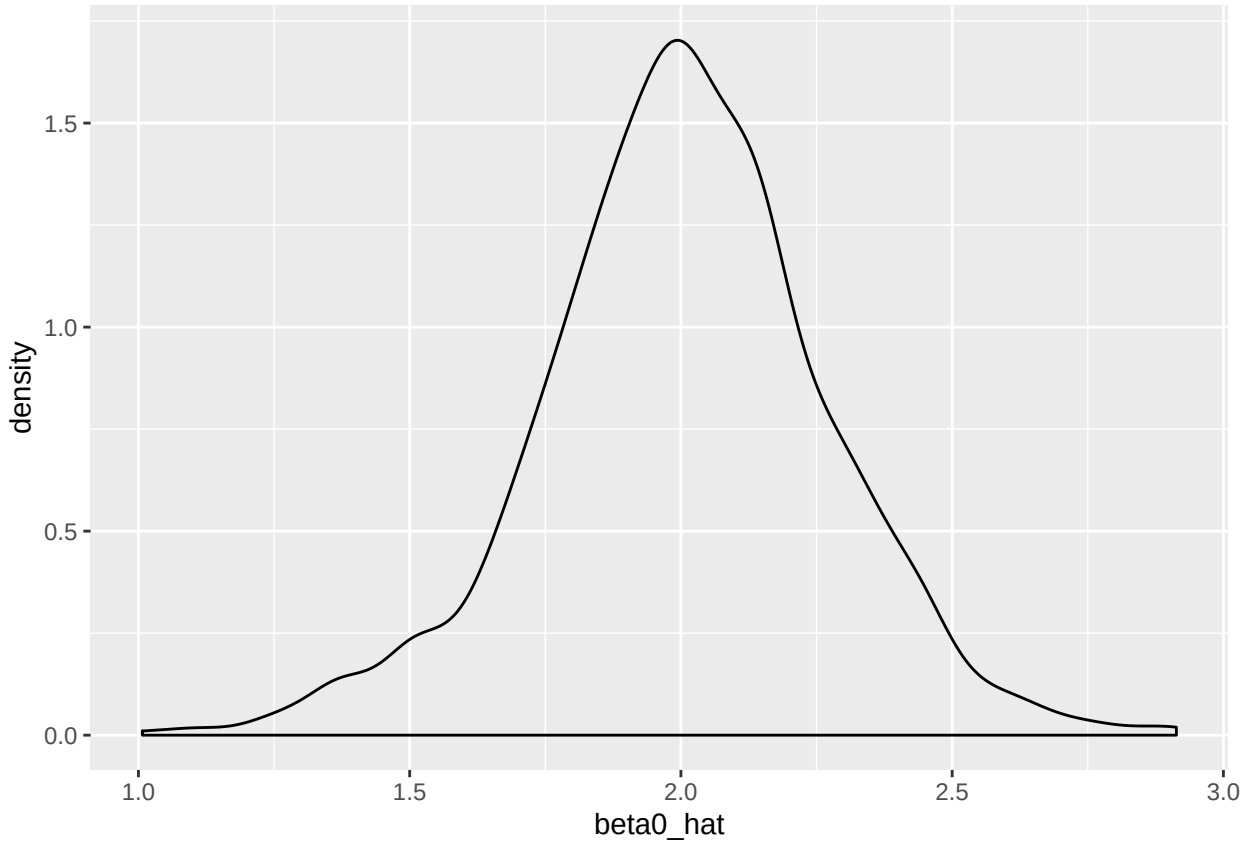
```
sim_results =
  rerun(1000, sim_regression(30, 2, 3)) %>%
  bind_rows()
```

Structurally, rerun is a lot like map – the first argument defines the amount of iteration and the second argument is the function to use in each iteration step. As with map, we've replaced a for loop with a segment of code that makes our purpose much more transparent but both approaches give the same results.

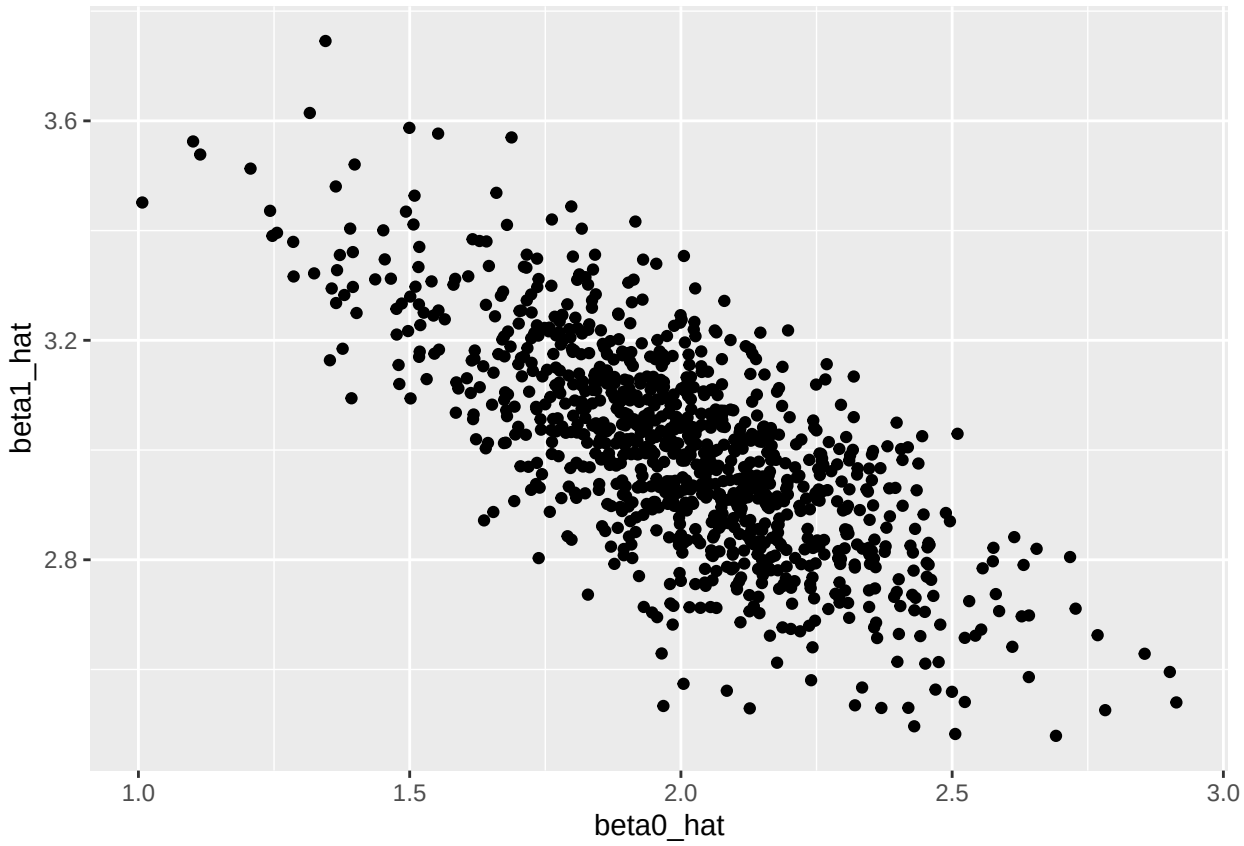
plot

summaries for our simulation results

```
# ensure beta0_hat follows normal distribution
sim_results %>%
  ggplot(aes(x = beta0_hat)) +
  geom_density()
```




```
# plot the result
sim_results %>%
  ggplot(aes(x = beta0_hat, y = beta1_hat)) +
  geom_point()
```



Simulation: SLR for several ns

To simulate with different sample sizes

```
n_list = list("n_30"  = 30,
              "n_60"  = 60,
              "n_120" = 120,
              "n_240" = 240)
output = vector("list", length = 4)

for (i in 1:4) {
  output[[i]] = rerun(100, sim_regression(n_list[[i]])) %>%
    bind_rows
}
```

After this loop, output is a list of 4 data frames; each data frame contains the results of 100 simulations at different sample sizes.

using list columns and map

```
simulate_n_regressions = function(n_runs = 100, n, beta0 = 2, beta1 = 3) {

  rerun(n_runs, sim_regression(n, beta0, beta1)) %>%
    bind_rows()

}
```

```r
sim_results =
  tibble(sample_size = c(30, 60, 120, 240)) %>%
  mutate(estimate_dfs = map(.x = sample_size, ~simulate_n_regressions(n = .x))) %>%
  unnest
```

```
## Warning: `cols` is now required.
## Please use `cols = c(estimate_dfs)`
```

Using a different call to increase the number of simulation runs or vary the parameters in the regression model:

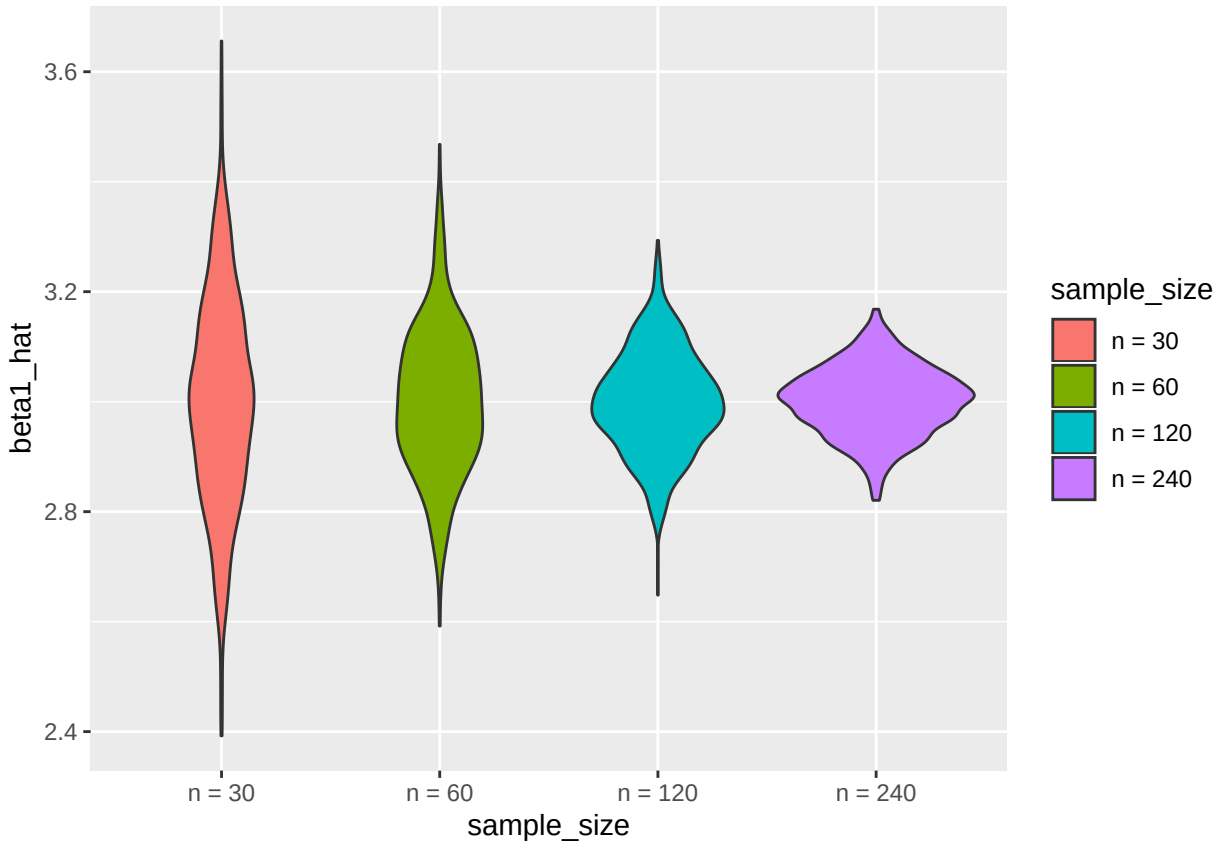
```r
sim_results =
  tibble(sample_size = c(30, 60, 120, 240)) %>%
  mutate(
    estimate_dfs = map(.x = sample_size, ~simulate_n_regressions(n_runs = 1000, n = .x))
  ) %>%
  unnest
```

```
## Warning: `cols` is now required.
## Please use `cols = c(estimate_dfs)`
```

Results

the distribution of slope estimates across sample sizes

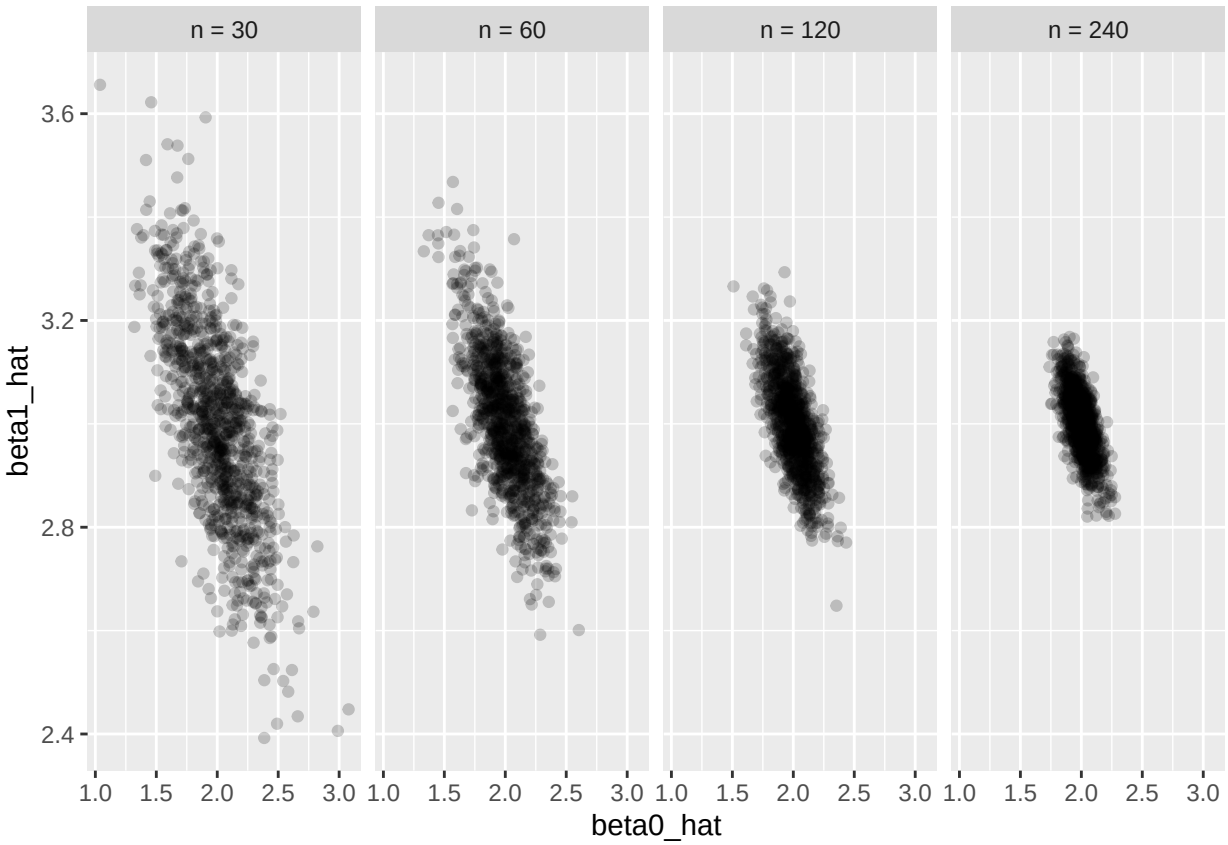
```r
sim_results %>%
  mutate(
    sample_size = str_c("n = ", sample_size),
    sample_size = fct_inorder(sample_size)) %>%
  ggplot(aes(x = sample_size, y = beta1_hat, fill = sample_size)) +
  geom_violin()
```



These estimates are centered around the truth (3) for each sample size, and the width of the distribution shrinks as sample size grows.

the bivariate distribution of intercept and slope estimates across sample sizes

```
sim_results %>%
  mutate(
    sample_size = str_c("n = ", sample_size),
    sample_size = fct_inorder(sample_size)) %>%
  ggplot(aes(x = beta0_hat, y = beta1_hat)) +
  geom_point(alpha = .2) +
  facet_grid(~sample_size)
```



Estimates of the intercept and slope are correlated with each other; this is expected from theoretical results describing the joint distribution of estimated regression coefficients.

the empirical mean and variance of these estimates.

```
sim_results %>%
  gather(key = parameter, value = estimate, beta0_hat:beta1_hat) %>%
  group_by(parameter, sample_size) %>%
  summarize(emp_mean = mean(estimate),
            emp_var = var(estimate)) %>%
  knitr::kable(digits = 3)
```

parameter	sample_size	emp_mean	
beta0_hat	30	1.998	
beta0_hat	60	2.000	
beta0_hat	120	1.999	
beta0_hat	240	2.000	
beta1_hat	30	3.000	
beta1_hat	60	3.002	
beta1_hat	120	3.000	
beta1_hat	240	3.002	
These values	are consisten	t with the	formulas presented above. This kind of check is a useful way to support derivat