Understanding the bias/variance tradeoff in ridge regression

Jaime Davila/Matthew Richey

4/13/2021

Introduction

We are interested in generating simulated data where we get to see the advantages of the ridge model over a simple linear regression. The simplest simulation we can get is by generating N points $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, where the x_i are generated from a normal distribution with mean 0 and standard deviation 1. We will be generating the y_i by multiplying the x_i by b and then adding an error term (which we call ϵ). Our epsilon will also have a normal distribution with mean 0 and standard deviation σ.

One of the instances where ridge regression can outperform linear regression is in the case where the number of points in our dataset is very small. So for the sake of our simulation let's set $N = 3$ (the number of observations), $b = 1$ (the slope) and $\sigma = 2$ (the standard deviation of our error term)

```
N <- 3
sig <- 2
b <- 1
```

And let's write a function `build_sim` which will create the simulated data according to our equation. We will be using this function many times later on, so we will add a column `id` that will allow us to label each different simulation

```
build_sim <- function(id){
  x1 <- rnorm(N,0,1)
  y <- b*x1+rnorm(N,0,sig)
  tibble(id,x1,y)
}
set.seed(12345)
(sim.tbl.1 <- build_sim(1))
```

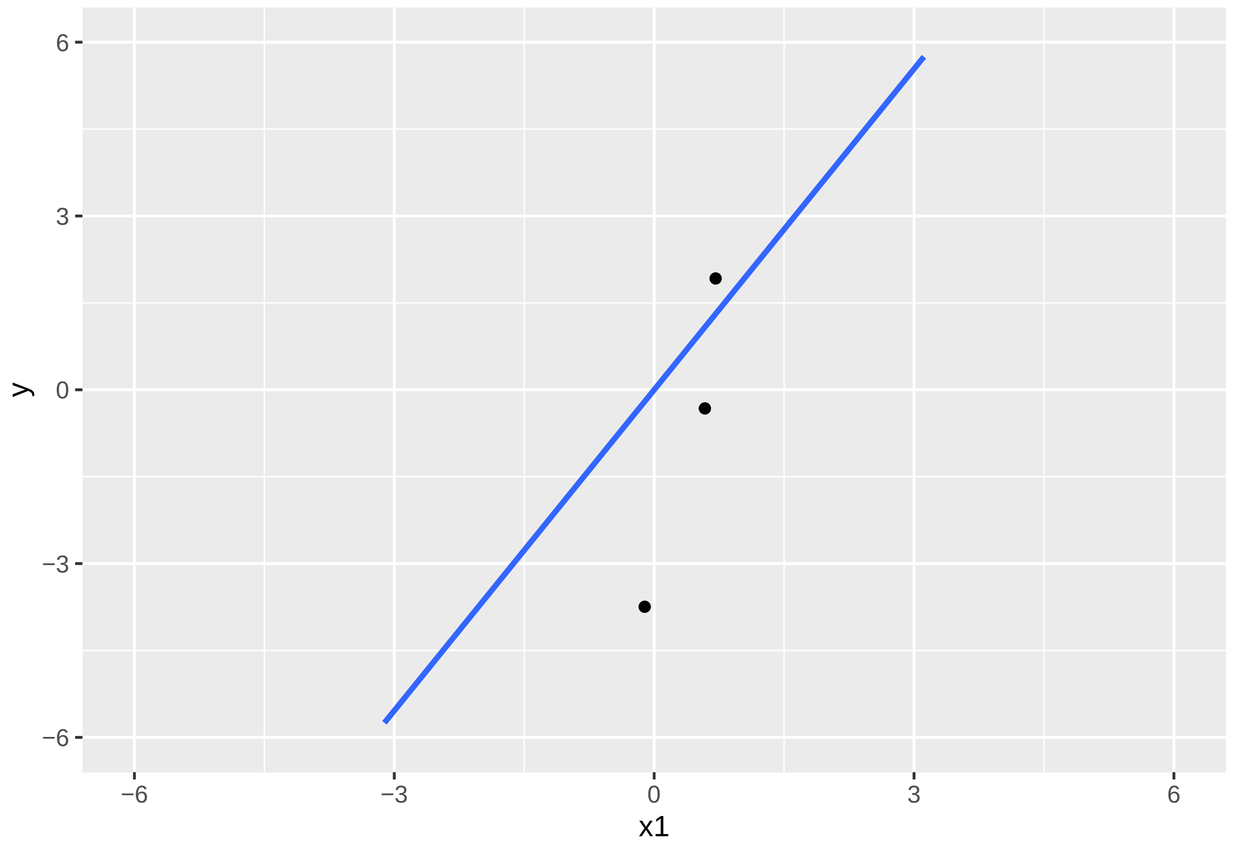
```
## # A tibble: 3 x 3
##      id     x1      y
##   <dbl>  <dbl>  <dbl>
## 1     1  0.586 -0.321
## 2     1  0.709  1.92
## 3     1 -0.109 -3.75
```

```
(sim.tbl.2 <- build_sim(2))
```

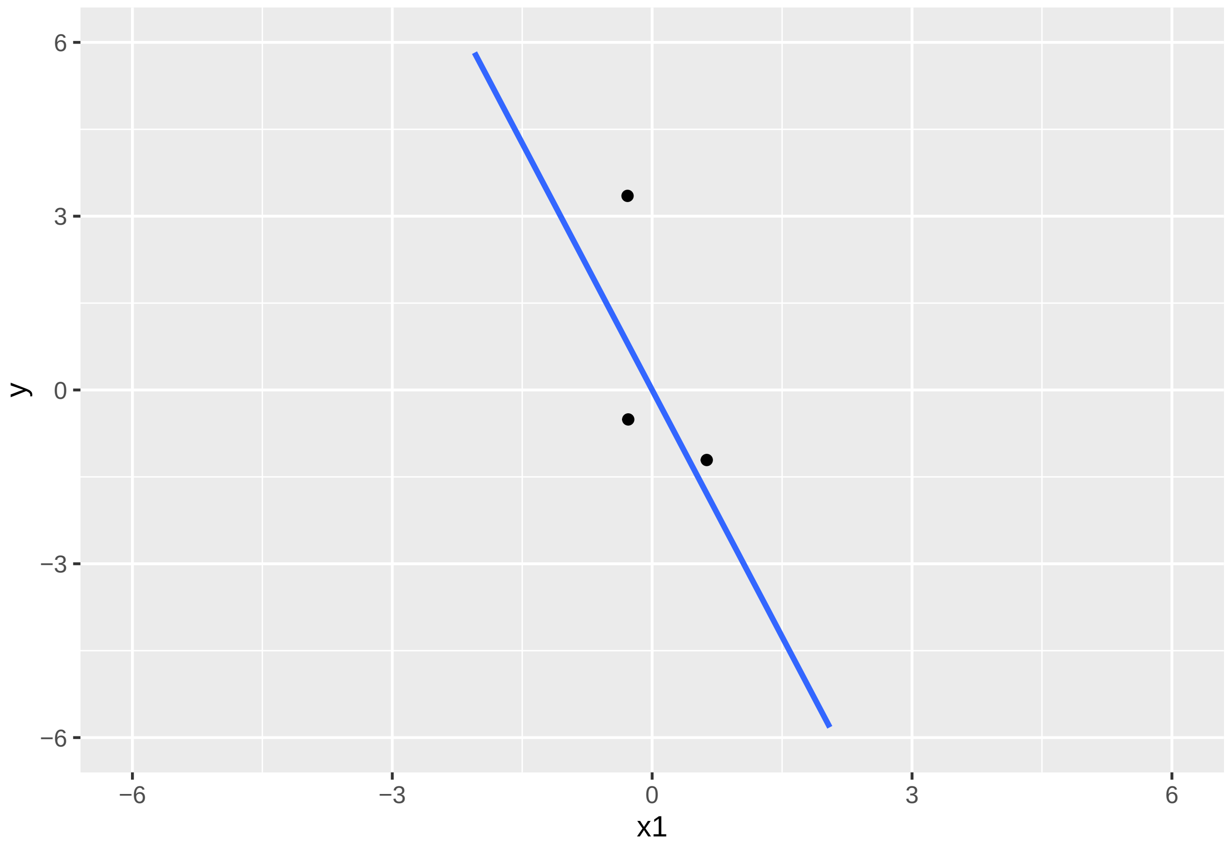
```
## # A tibble: 3 x 3
##      id     x1      y
##   <dbl>  <dbl>  <dbl>
## 1     2  0.630 -1.21
## 2     2 -0.276 -0.509
## 3     2 -0.284  3.35
```

And let's plot our two simulated tables and their linear trends.

```
ggplot(sim.tbl.1, aes(x1,y)) +
      geom_point() +
      xlim(-6,6)+
      ylim(-6,6)+
      geom_smooth(method=lm, formula = y~0+x,
                  se=FALSE, fullrange=TRUE)
```



```
ggplot(sim.tbl.2, aes(x1,y)) +
      geom_point() +
      xlim(-6,6)+
      ylim(-6,6)+
      geom_smooth(method=lm, formula = y~0+x,
                  se=FALSE, fullrange=TRUE)
```



Notice how our trend lines look very different in these two instances. Notice that is partly due to the fact that we have a small number of points, so the slope estimates can change by a lot.

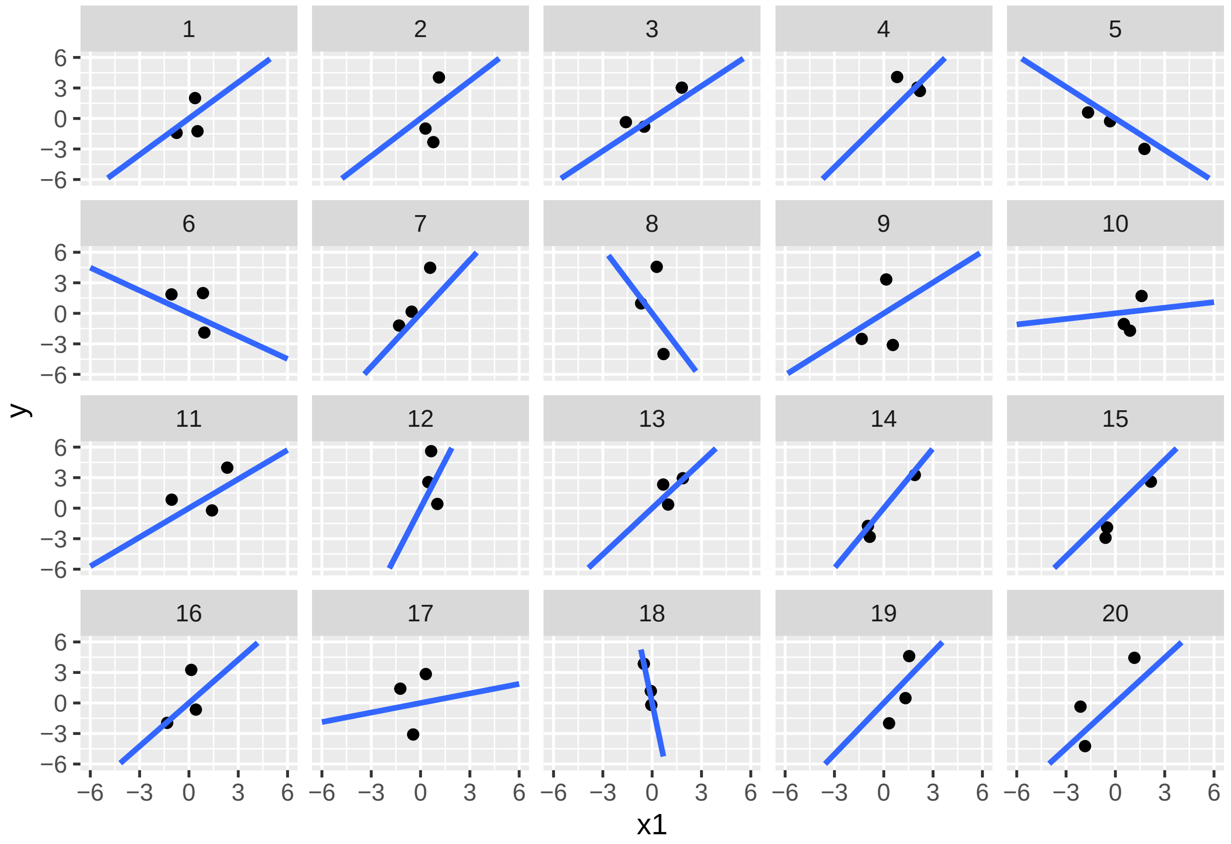
We would like to generate 20 simulation and plot all of them at the same time. In our first step we will create our 20 simulations by using the function `map_dfr()`. `map_dfr(x,f)` works by applying the function `f` for every element of `x` and then putting the results in dataframe. In particular it assumes that the results of `f(x)` are a dataframe (that's why the suffix `_dfr` in `map_dfr`)

```
(sim.tbl <- map_dfr(1:20, build_sim))
```

```
## # A tibble: 60 x 3
##       id     x1      y
##    <int>  <dbl>  <dbl>
##  1     1  0.371  2.00
##  2     1  0.520 -1.25
##  3     1 -0.751 -1.41
##  4     2  1.12   4.03
##  5     2  0.299 -0.990
##  6     2  0.780 -2.33
##  7     3 -1.60  -0.357
##  8     3  1.81   3.03
##  9     3 -0.482 -0.806
## 10     4  0.812  4.08
## # ... with 50 more rows
```

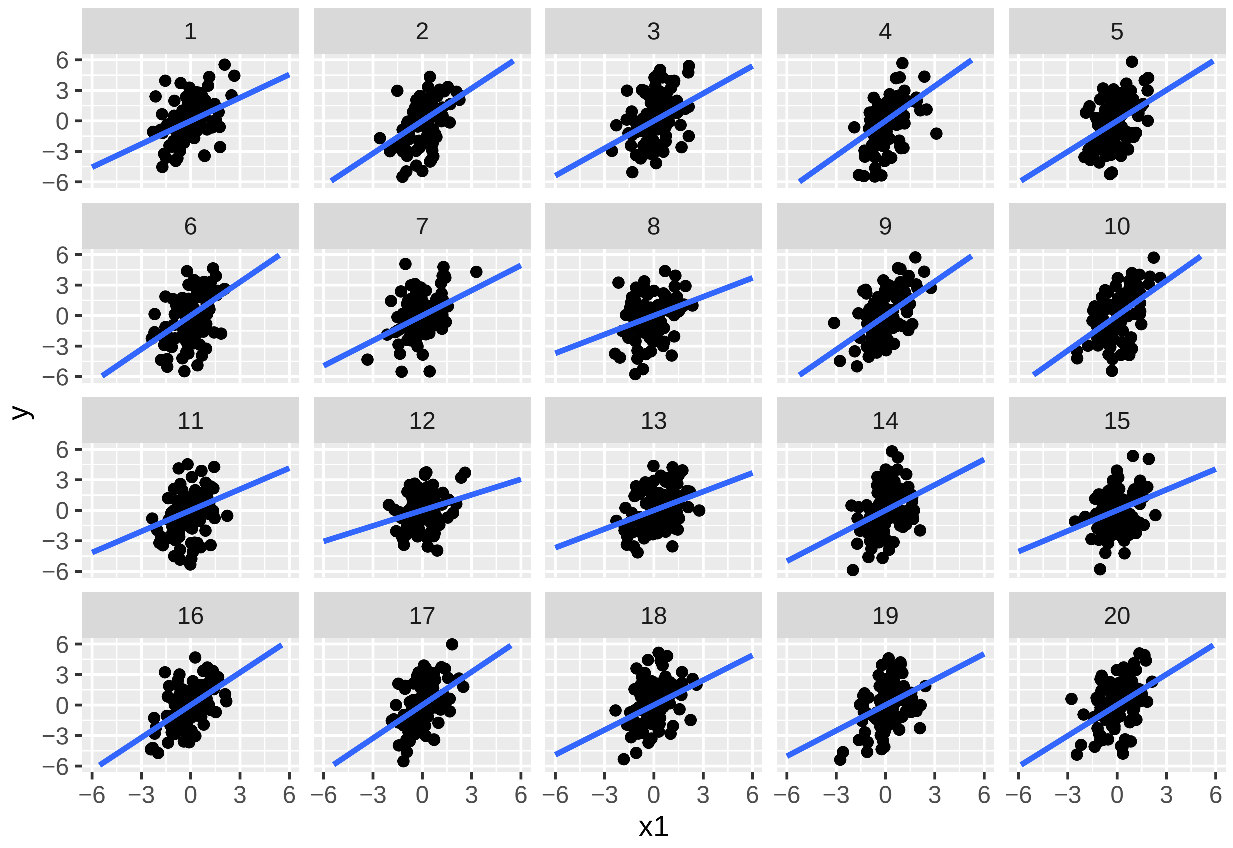
Finally we can show graphically the results of those simulations below. Notice how the slope varies according to each simulation

```
ggplot(sim.tbl,aes(x1,y)) +
      geom_point() +
      xlim(-6,6)+
      ylim(-6,6)+
      geom_smooth(method=lm, formula = y~0+x,
                  se=FALSE, fullrange=TRUE)+
      facet_wrap(vars(id))
```



Let's also point out that if the number of points N increases we get a more uniform set of models

```
N <- 100
map_dfr(1:20, build_sim)  %>%
  ggplot(aes(x1,y)) +
      geom_point() +
      xlim(-6,6)+
      ylim(-6,6)+
      geom_smooth(method=lm, formula = y~0+x,
                  se=FALSE, fullrange=TRUE)+
      facet_wrap(vars(id))
```



Before we continue our exercises, let's set up N equal to 3 again

```
N <- 3
```

1. Create a function `calc_slope_lm()` that creates a simulated dataset using the function `build_sim()` and returns a tibble with the `id` and the slope of the linear model with 0 intercept.

```
calc_slope_lm <- function(id) {
    tibble (id=id,x=slope)
}
```

```
calc_slope_lm <- function(id) {
  # Generate simulated data
  sim.tbl <- build_sim(id)
  # Create and fit the linear model
  lm.model <- linear_reg() %>%
    set_engine("lm")
  recipe <- recipe(y ~ 0+x1, data=sim.tbl)
  wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(lm.model)
  fit <- fit(wflow, sim.tbl)
  # Extract the slope coefficient
  x <- tidy(fit) %>%
    filter(term=="x1") %>%
    pull(estimate)

  tibble (id=id,x=x)
}

set.seed(54321)
calc_slope_lm(1)
```

```
## # A tibble: 1 x 2
##      id     x
##   <dbl> <dbl>
## 1     1 -1.88
```

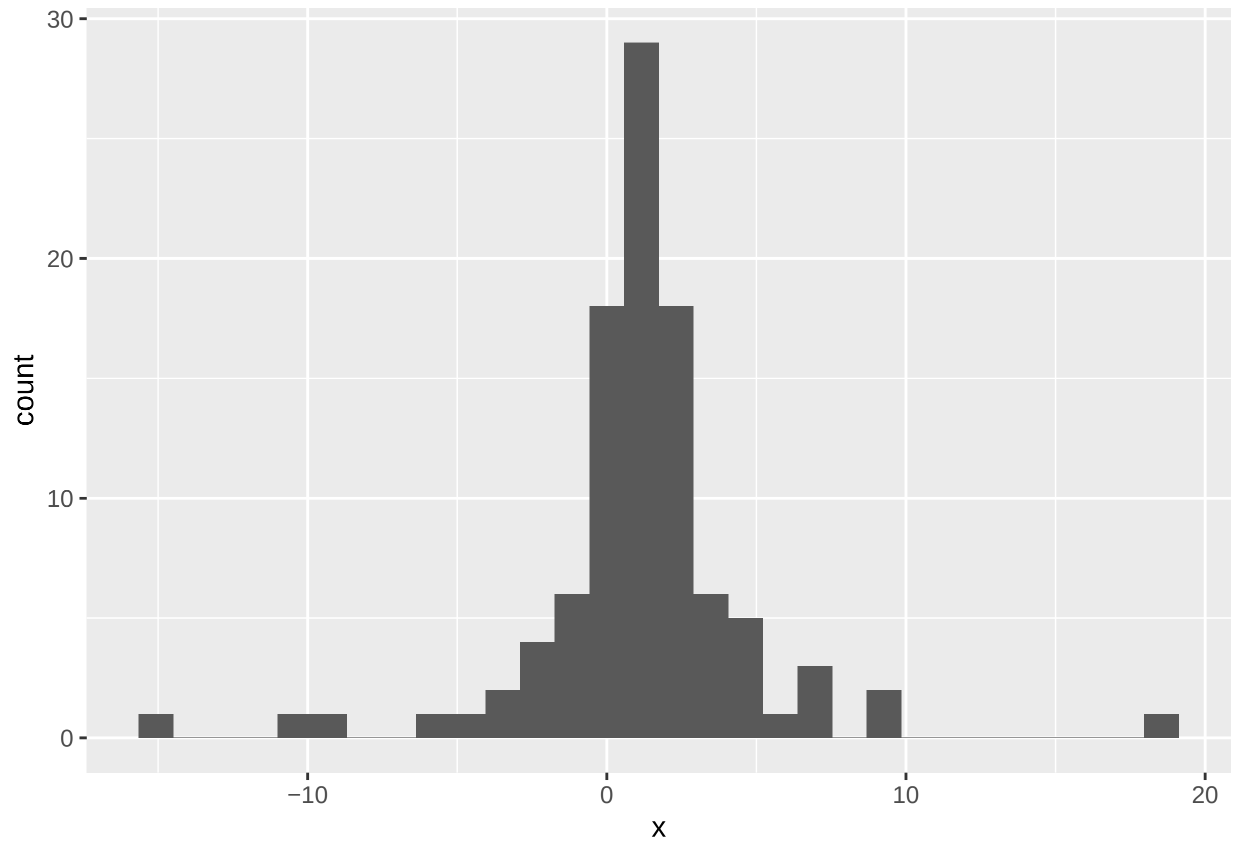
```
calc_slope_lm(2)
```

```
## # A tibble: 1 x 2
##      id     x
##   <dbl> <dbl>
## 1     2 0.722
```

2. Use `map_dfr` on `calc_slope_lm()` to create a table with the values of the slope for 100 simulations. Do a histogram of the slope and calculate its mean and standard deviation. We can define the bias as the mean of the slope minus the actual slope value (1). What is the bias of the simulated data?

```
set.seed(54321)
```

```
sim.tbl <- map_dfr(1:100, calc_slope_lm)
sim.tbl %>%
  ggplot(aes(x))+
  geom_histogram()
```



```
mean(sim.tbl$x)
```

```
## [1] 1.140671
```

```
sd(sim.tbl$x)
```

```
## [1] 3.755043
```

```
mean((sim.tbl$x-1))
```

```
## [1] 0.1406707
```

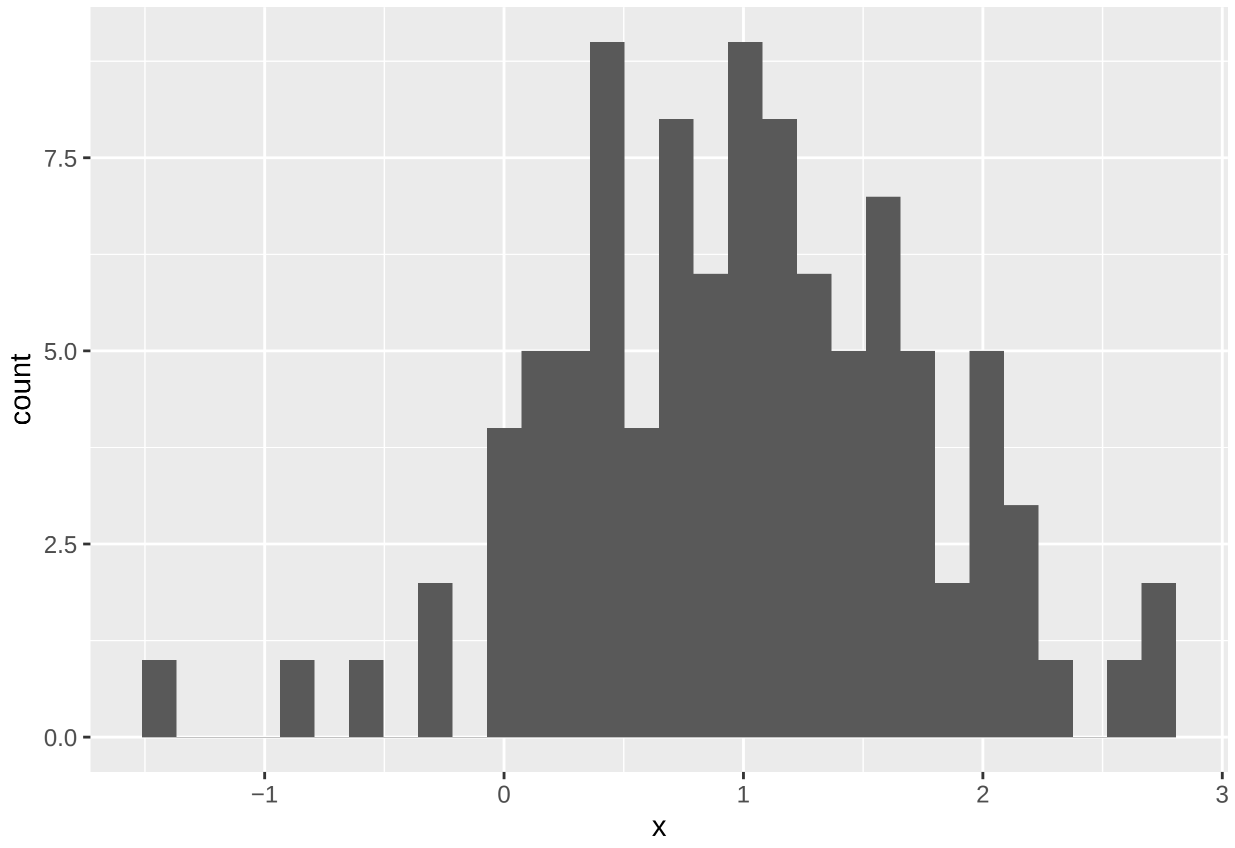
Notice that the bias is small, however there is a quite a bit of variation.

3. Create a new simulated dataset by setting up $N = 10$ and $N = 100$. How does the slope histogram changed when compared with exercise 2?

```
set.seed(55057)
```

```
N <- 10

sim.tbl <- map_dfr(1:100, calc_slope_lm)
sim.tbl %>%
  ggplot(aes(x))+
  geom_histogram()
```



```
mean(sim.tbl$x)
```

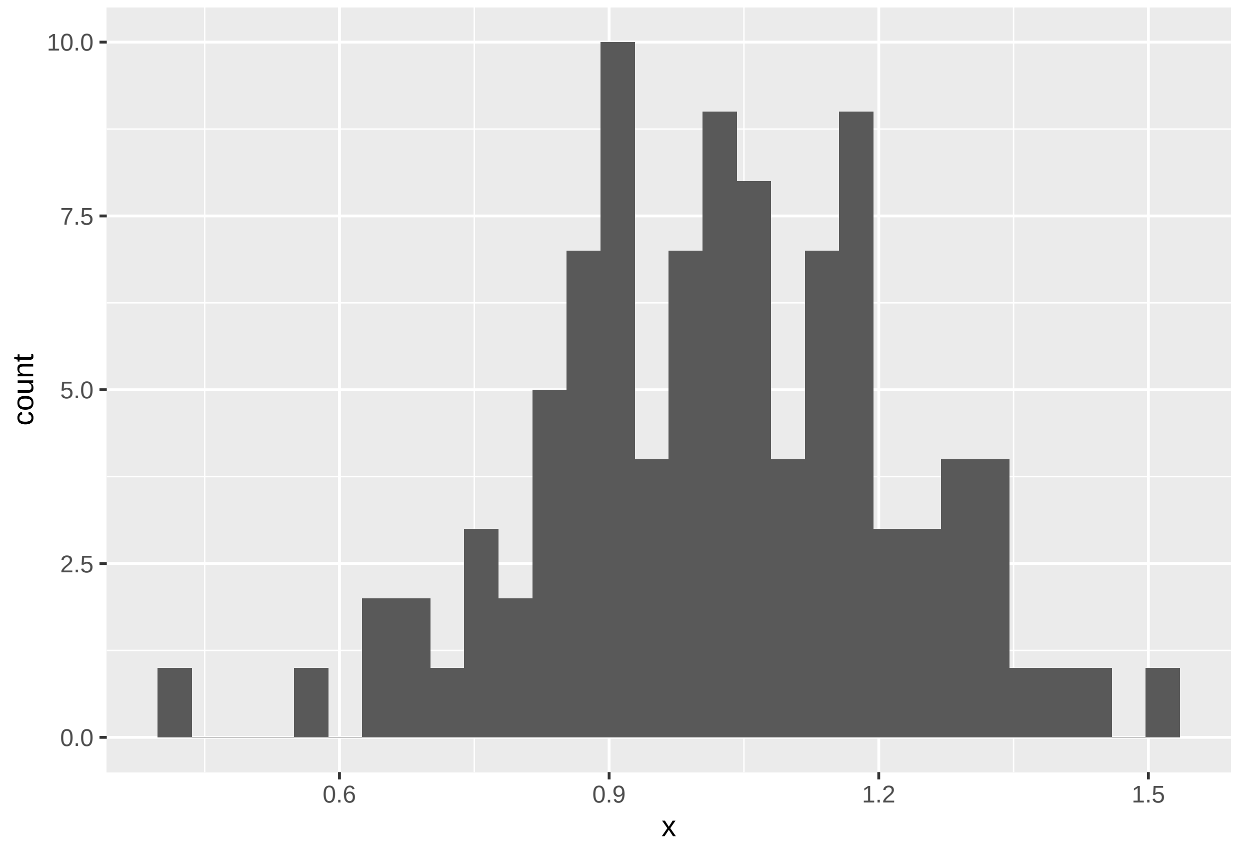
```
## [1] 1.006688
```

```
sd(sim.tbl$x)
```

```
## [1] 0.763125
```

```
N <- 100

sim.tbl <- map_dfr(1:100, calc_slope_lm)
sim.tbl %>%
  ggplot(aes(x))+
  geom_histogram()
```



```
mean(sim.tbl$x)
```

```
## [1] 1.023338
```

```
sd(sim.tbl$x)
```

```
## [1] 0.2000687
```

Comparing linear regression and ridge

We would like to compare the estimated slope that we would get from our dataset using both ridge and linear models. In the `glmnet` implementation the ridge model needs to have at least two explanatory variables x_1 and x_2 so we will be modifying our simulation function to create an output of the form $b \cdot x_1 + b \cdot x_2 + \epsilon$. Our first step is to create a `build_sim2d` function as below

```
N <- 5
build_sim2d <- function(id){
  x1 <- rnorm(N,0,1)
  x2 <- rnorm(N,0,1)
  y <- b*x2+b*x1+rnorm(N,0,sig)

  tibble(id,x1,x2,y)
}
set.seed(123)
build_sim2d(1)
```

```
## # A tibble: 5 x 4
##      id      x1     x2      y
##   <dbl>   <dbl>  <dbl>  <dbl>
## 1     1 -0.560   1.72   3.60
## 2     1 -0.230   0.461  0.950
## 3     1  1.56   -1.27   1.10
## 4     1  0.0705 -0.687 -0.395
## 5     1  0.129  -0.446 -1.43
```

And we will be creating a `calc_coefs` function that outputs the estimates of the coefficients for a given model.

```
calc_coefs <- function(id, model) {
  # We generate a simulated dataset
  sim.tbl <- build_sim2d(id)

  # We create a workflow for our model and fit it on the simulated data
  recipe <- recipe(y ~ 0+x1+x2, data=sim.tbl) %>%
    step_normalize(all_predictors())
  wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(model)
  fit <- fit(wflow, sim.tbl)

  # We pull the coefficients from x1 and x2 and put them in a tibble
  x1 <- tidy(fit) %>%
    filter(term=="x1") %>%
    pull(estimate)
  x2 <- tidy(fit) %>%
    filter(term=="x2") %>%
    pull(estimate)
  tibble (id=id,x1=x1, x2=x2)
}
```

Notice that using `calc_coefs` we can calculate the coefficients for a linear model as follows

```
lm.model <- linear_reg() %>%
    set_engine("lm")

calc_coefs(1,lm.model)
```

```
## # A tibble: 1 x 3
##      id     x1    x2
##   <dbl>  <dbl> <dbl>
## 1     1 -0.252  2.07
```

Or we can calculate the coefficients for a ridge model with penalty equal to 1.

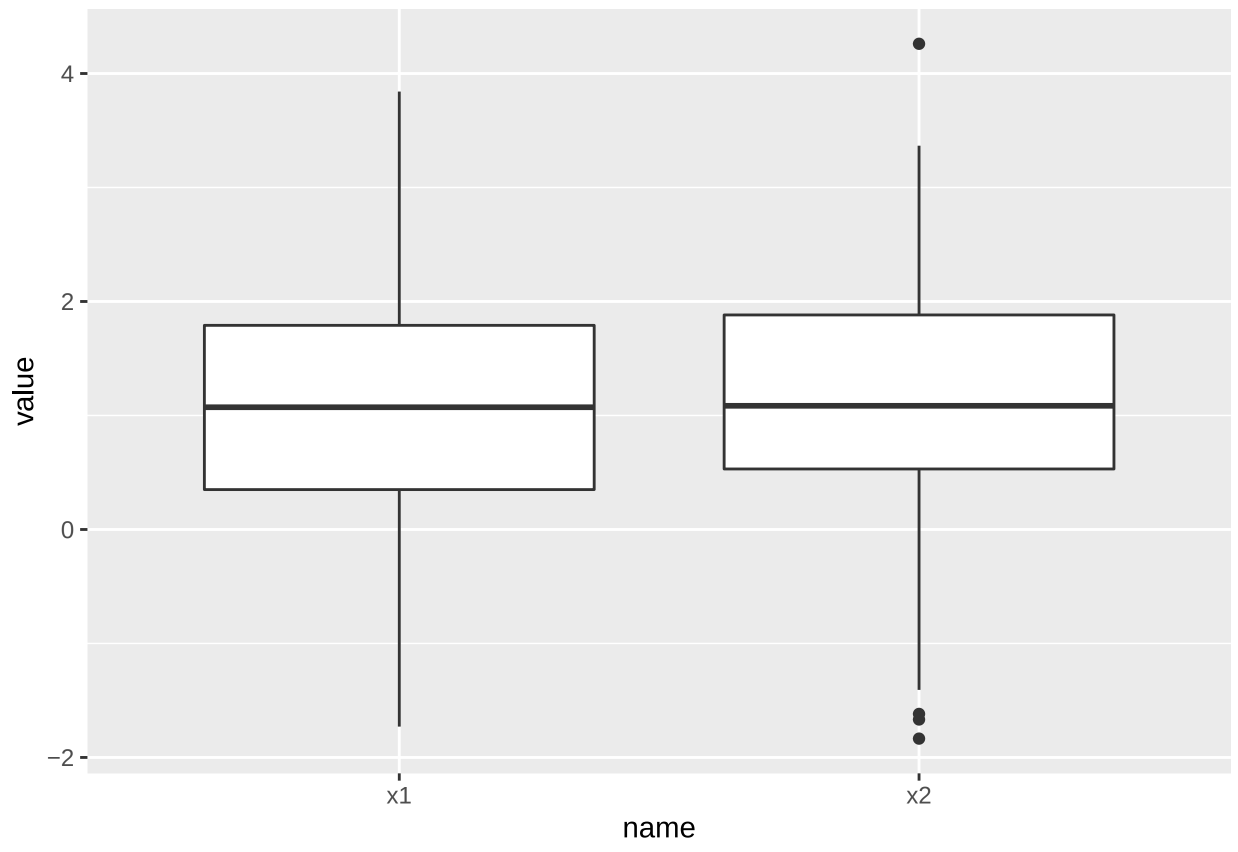
```r
ridge.model <-
  linear_reg(mixture = 0, penalty=1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

calc_coefs(1,ridge.model)
```

```
## # A tibble: 1 x 3
##      id    x1    x2
##   <dbl> <dbl> <dbl>
## 1     1  1.16 0.798
```

3. Use the functions `map_dfr` and `calc_coefs` to obtain the coefficients `x1` and `x2` from the linear model and from the ridge model with penalty 1 for 100 simulations. Do a boxplot of the values of the coefficients for each type of model. Calculate the mean, bias and standard deviation. How do those values compare across models?

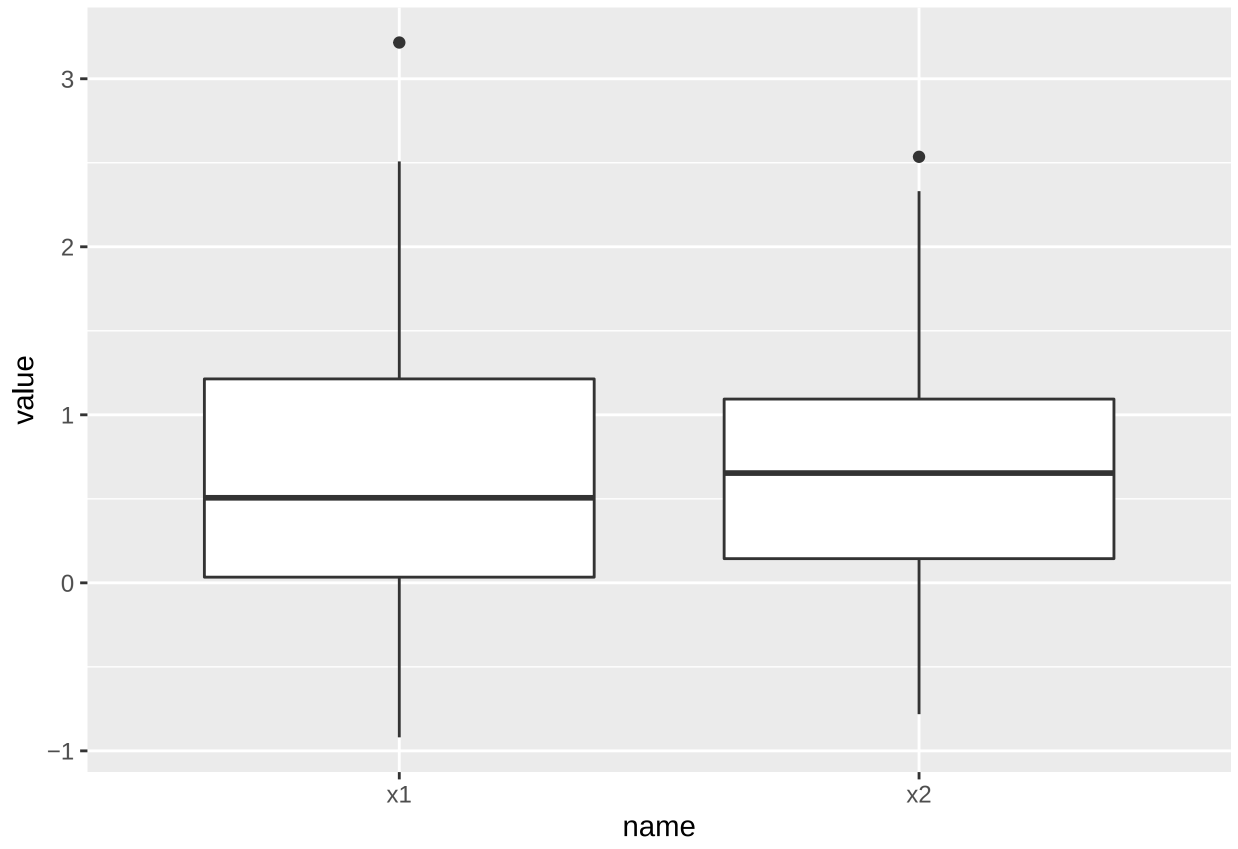
```r
coef.lm.tbl <- map_dfr(1:100, calc_coefs, lm.model)
coef.lm.tbl %>%
  pivot_longer(x1:x2) %>%
  ggplot(aes(name,value)) +
  geom_boxplot()
```



```r
coef.lm.tbl %>%
  pivot_longer(x1:x2) %>%
  group_by(name) %>%
  summarize(mean=mean(value),
            bias=mean(value-1),
            sd = sd(value))
```

```
## # A tibble: 2 x 4
##   name   mean   bias    sd
##   <chr> <dbl>  <dbl> <dbl>
## 1 x1     1.04 0.0391  1.12
## 2 x2     1.13 0.131   1.11
```

```
coef.ridge.tbl <- map_dfr(1:100, calc_coefs, ridge.model)
coef.ridge.tbl %>%
  pivot_longer(x1:x2) %>%
  ggplot(aes(name,value)) +
  geom_boxplot()
```



```
coef.ridge.tbl %>%
  pivot_longer(x1:x2) %>%
  group_by(name) %>%
  summarize(mean=mean(value),
            bias=mean(value-1),
            sd = sd(value))
```

```
## # A tibble: 2 x 4
##   name   mean   bias    sd
##   <chr> <dbl>  <dbl> <dbl>
## 1 x1    0.642 -0.358 0.843
## 2 x2    0.683 -0.317 0.689
```

4. Use penalty values of 0.1, 1, 10, 100 and see what is the effect on the mean and standard deviation of the coefficients for x1 and x2 when using a ridge model.

```
for (penalty in c(0.1,1,10,100)){
  ridge.model <-
    linear_reg(mixture = 0, penalty=penalty) %>%
    set_mode("regression") %>%
    set_engine("glmnet")

  coef.ridge.tbl <- map_dfr(1:30, calc_coefs, ridge.model)
  stats.tbl <-  coef.ridge.tbl %>%
    pivot_longer(x1:x2) %>%
    group_by(name) %>%
    summarize(mean=mean(value),
              bias=mean(value-1),
              sd = sd(value))

  print(penalty)
  print(stats.tbl)
}
```

```
## [1] 0.1
## # A tibble: 2 x 4
##   name   mean    bias    sd
##   <chr> <dbl>   <dbl> <dbl>
## 1 x1    1.17   0.169  0.873
## 2 x2    0.931 -0.0686 1.19
## [1] 1
## # A tibble: 2 x 4
##   name   mean   bias    sd
##   <chr> <dbl>  <dbl> <dbl>
## 1 x1    0.558 -0.442 0.865
## 2 x2    0.688 -0.312 0.787
## [1] 10
## # A tibble: 2 x 4
##   name   mean   bias    sd
##   <chr> <dbl>  <dbl> <dbl>
## 1 x1    0.124 -0.876 0.214
## 2 x2    0.171 -0.829 0.191
## [1] 100
## # A tibble: 2 x 4
##   name    mean   bias     sd
##   <chr>  <dbl>  <dbl>  <dbl>
## 1 x1    0.0274 -0.973 0.0334
## 2 x2    0.0159 -0.984 0.0298
```

The Bias-Variance Trade-off

Imagine a fixed prediction value $x_0 := (x_{0,1}, x_{0,2})$\$ and an observed value $y_0 = f(x_0) + \epsilon$. And let $\hat{f}(x)$ be that value that our model predicts.

The bias-variance trade-off equation says:

$$E[(y_0 - \hat{f}(x_0))^2] = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

Let’s look closely at each of these components.

- $E[(y_0 - \hat{f}(x_0)^2]$ is the expected (average) value $(y_0 - \hat{f}(x_0))^2$. This a gauge of how close our prediction comes to the actual value. We also know this as the **Mean Squared Error (MSE)**.
- $\text{Var}(\hat{f}(x_0))$ is the variance of the values $\hat{f}(x_0)$ generated from each training set. This is a gauge of how “wiggly” the prediction is as we build $\hat{f}(x)$ with different training data sets.
- $\text{Bias}(\hat{f}(x_0))$ is the expected value of $\hat{f}(x_0) - f(x_0)$ over the training data, i.e., $\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0) - f(x_0)]$. This is a gauge how the predicted values misses the actual value. Note that the bias could be zero but we still miss by a lot!
- σ^2 is the “noise’.The noise reflects all the influences that our $\hat{f}(x)$ misses. Generally, we assume that the noise has mean zero and $\text{Var}(\epsilon) = \sigma^2$.

Let’s create a function `calc_errors` that calculates the mse, variance and bias-squared for a given model.

```r
calc_errors <- function(id, model) {
  # Simulate our testing/training dataset
  train.tbl <- build_sim2d(id)
  test.tbl <- build_sim2d(id)

  # Create a workflow and fit using your training
  recipe <- recipe(y ~ 0+x1+x2, data=train.tbl)
  wflow <- workflow() %>%
    add_recipe(recipe) %>%
    add_model(model)
  fit <- fit(wflow, train.tbl)

  # Calculate the predictions on the testing dataset
  predict.tbl <- augment(fit, test.tbl)

  # Calculate mse, var, bias and put everything on a tibble
  mse = mean((predict.tbl$y-predict.tbl$.pred)^2)
  var = mean(predict.tbl$.pred^2)
  bias = mean(predict.tbl$y-predict.tbl$.pred)^2
  tibble(mse=mse, var=var, bias=bias)
}
calc_errors(1,lm.model)
```

```
## # A tibble: 1 x 3
##     mse   var  bias
##   <dbl> <dbl> <dbl>
## 1  10.6  3.83  1.30
```

Let’s calculate those parameters on the linear model

```r
calc_errors(1,lm.model)
```

```
## # A tibble: 1 x 3
##     mse   var  bias
##   <dbl> <dbl> <dbl>
## 1  1.95  7.36 0.380
```

```r
set.seed(12345)
errors.lm <- map_dfr(1:20, calc_errors, lm.model)
errors.lm %>%
  pivot_longer(1:3) %>%
  group_by(name) %>%
  summarize(mean = mean(value))
```

```
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   3.44
## 2 mse   17.2
## 3 var   21.5
```

And let's calculate those values on different ridge models with different penalties.

```r
for (penalty in c(0.1,1,10,100,1000)){
  ridge.model <-
    linear_reg(mixture = 0, penalty=penalty) %>%
    set_mode("regression") %>%
    set_engine("glmnet")

  set.seed(12345)
  errors.ridge <- map_dfr(1:20, calc_errors, ridge.model)
  errors.tbl <- errors.ridge %>%
    pivot_longer(1:3) %>%
    group_by(name) %>%
    summarize(mean = mean(value))

  print(penalty)
  print(errors.tbl)
}
```

```
## [1] 0.1
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   3.02
## 2 mse   14.9
## 3 var   18.4
## [1] 1
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   2.16
## 2 mse    9.39
## 3 var   10.5
## [1] 10
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   1.72
## 2 mse    5.91
## 3 var    1.94
## [1] 100
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   2.05
## 2 mse    7.06
## 3 var    1.32
## [1] 1000
## # A tibble: 3 x 2
##   name   mean
##   <chr> <dbl>
## 1 bias   2.11
## 2 mse    7.29
## 3 var    1.32
```

Notice how although ridge it has more bias than linear regression, it also has less variation, which results in a smaller overall mse.