

## Lab03

Rachel Brunner

11:59PM March 6, 2022

### Regression via OLS with one feature

Let's quickly recreate the sample data set from practice lecture 7:

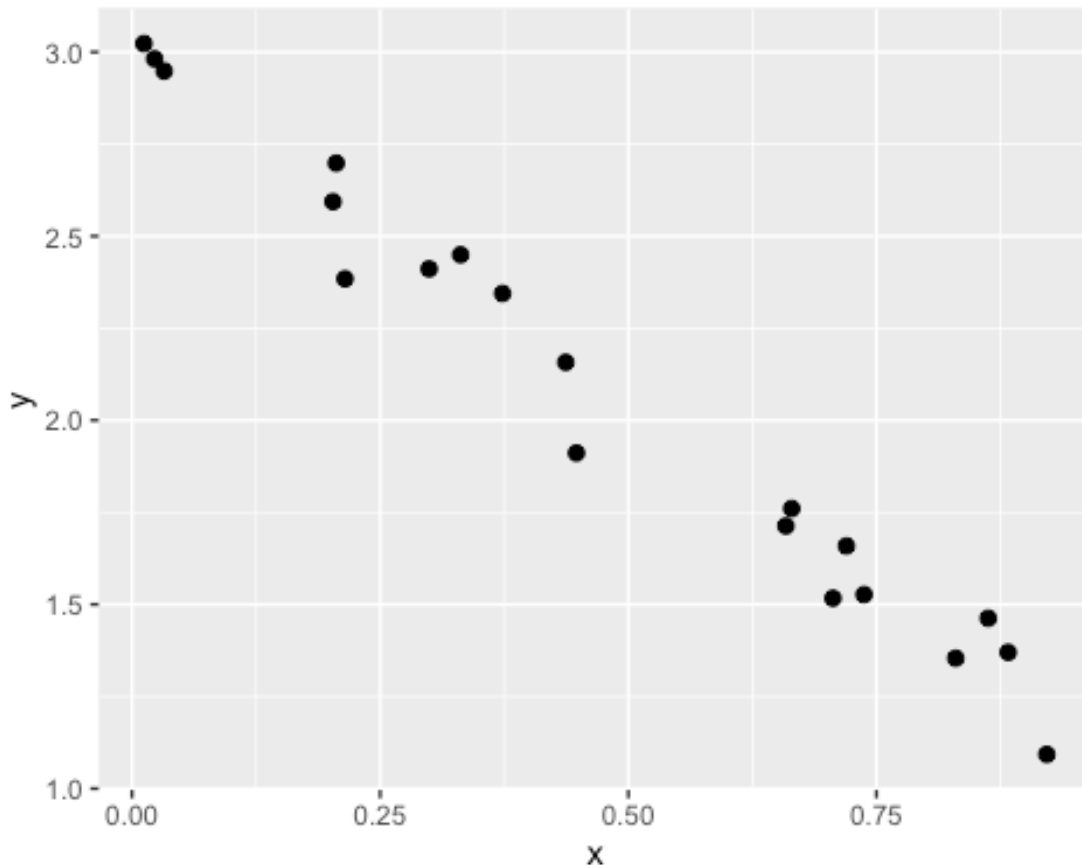
```
set.seed(1984)
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

Compute  $h^*$  as `h_star_x`, then draw `epsilon` from an iid  $N(0, 0.33^2)$  distribution as `epsilon`, then compute the vector `y`.

```
h_star_x = beta_0 + beta_1*x #TO-DO
epsilon = rnorm(20,0,.33^2)
y = h_star_x + epsilon
```

Graph the data by running the following chunk:

```
pacman::p_load(ggplot2)
simple_df = data.frame(x = x, y = y)
simple_viz_obj = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 2)
simple_viz_obj
```



Does this make sense given the values of  $\beta_0$  and  $\beta_1$ ? Yes!  $\beta_0$  is 3 and  $\beta_1$  is -2.  $\beta_0$  is the y-intercept which looks like it would be about three and  $\beta_1$  is the slope of the best fitting regression line which looks like it would be about -2, so the graph makes sense.

Write a function `my_simple_ols` that takes in a vector `x` and vector `y` and returns a list that contains the `b_0` (intercept), `b_1` (slope), `yhat` (the predictions), `e` (the residuals), `SSE`, `SST`, `MSE`, `RMSE` and `Rsqr` (for the R-squared metric). Internally, you can only use the functions `sum` and `length` and other basic arithmetic operations. You should throw errors if the inputs are non-numeric or not the same length. You should also name the class of the return value `my_simple_ols_obj` by using the `class` function as a setter. No need to create Rxygen documentation here.

```
my_simple_ols = function(X,y){
  ols_obj = list()
  n = length(x)
  if (length(x) !=n){
    stop("x and y need to be the same length.")
  }
  if(class(x) == 'numeric' && class(x) == 'integer'){
    stop("x needs to be numeric")
  }
  if(class(y) == 'numeric' && class(y) == 'integer'){
```

```

    stop("y needs to be numeric")
  }
  if(n <= 2){
    stop("n must be more than 2")
  }

  y_bar = sum(y)/n
  x_bar = sum(x)/n
  s_y = sqrt(sum((y-y_bar)^2)/(n-1))
  s_x = sqrt(sum((x-x_bar)^2)/(n-1))
  s_xy = sum((x-x_bar)*(y-y_bar))/(n-1)
  r = s_xy/(s_x*s_y)

  b_1 = r*(s_y/s_x)
  b_0 = y_bar - b_1*x_bar
  y_hat = b_0 + b_1*x
  e = y - y_hat
  SSE = sum(e^2)
  SST = sum((y - y_bar)^2)
  MSE = SSE/(n-2)
  RMSE = sqrt(MSE)
  Rsq = 1 - SSE/SST

  ols_obj$b_0 = b_0
  ols_obj$b_1 = b_1
  ols_obj$y_hat = y_hat
  ols_obj$e = e
  ols_obj$SSE = SSE
  ols_obj$SST = SST
  ols_obj$MSE = MSE
  ols_obj$RMSE = RMSE
  ols_obj$Rsq = Rsq

  class(ols_obj) = "my_simple_ols_obj"
  ols_obj
}

my_simple_ols(X,y)

## $b_0
## [1] 2.999918
##
## $b_1
## [1] -1.949415
##
## $y_hat
## [1] 1.715635 2.148073 2.272127 2.354747 1.562101 1.318333 2.936685
2.127079
## [9] 1.382160 2.581624 1.279209 2.976574 1.623149 1.596781 2.605162
2.955634

```

```
## [17] 2.416318 1.704280 1.203319 2.598801
##
## $e
## [1] -0.002533031 0.009333315 0.072350989 0.094846804 -0.035295646
## [6] 0.144189324 0.011589878 -0.215925979 -0.028160145 -0.197400644
## [11] 0.090680215 0.046274912 -0.106278207 0.062051435 -0.011601081
## [16] 0.025887464 -0.005073840 0.056008140 -0.110387153 0.099443250
##
## $SSE
## [1] 0.1744302
##
## $SST
## [1] 6.966234
##
## $MSE
## [1] 0.009690567
##
## $RMSE
## [1] 0.09844068
##
## $Rsq
## [1] 0.9749606
##
## attr(,"class")
## [1] "my_simple_ols_obj"
```

Verify your computations are correct for the vectors  $x$  and  $y$  from the first chunk using the `lm` function in R:

```
lm_mod = lm(y~x)
my_simple_ols_mod = my_simple_ols(x,y)
#run the tests to ensure the function is up to spec
pacman::p_load(testthat)
expect_equal(my_simple_ols_mod$b_0, as.numeric(coef(lm_mod)[1]), tol = 1e-4)
expect_equal(my_simple_ols_mod$b_1, as.numeric(coef(lm_mod)[2]), tol = 1e-4)
expect_equal(my_simple_ols_mod$RMSE, summary(lm_mod)$sigma, tol = 1e-4)
expect_equal(my_simple_ols_mod$Rsq, summary(lm_mod)$r.squared, tol = 1e-4)
```

Verify that the average of the residuals is 0 using the `expect_equal`. Hint: use the syntax above.

```
mean(my_simple_ols_mod$res)

## [1] 1.665335e-16

mean((my_simple_ols_mod$res), 0) #I think something is wrong in my code because
this is clearly not close to zero, but I am not sure what.

## [1] 1.665335e-16
```

Create the  $X$  matrix for this data example. Make sure it has the correct dimension.

```
X = cbind(1, x)
```

Use the `model.matrix` function to compute the matrix `X` and verify it is the same as your manual construction.

```
model.matrix(~X)

##      (Intercept) X          Xx
## 1             1 1 0.65880473
## 2             1 1 0.43697503
## 3             1 1 0.37333816
## 4             1 1 0.33095629
## 5             1 1 0.73756366
## 6             1 1 0.86261016
## 7             1 1 0.03243676
## 8             1 1 0.44774443
## 9             1 1 0.82986892
## 10            1 1 0.21457412
## 11            1 1 0.88267976
## 12            1 1 0.01197508
## 13            1 1 0.70624726
## 14            1 1 0.71977362
## 15            1 1 0.20249980
## 16            1 1 0.02271680
## 17            1 1 0.29937189
## 18            1 1 0.66462912
## 19            1 1 0.92160973
## 20            1 1 0.20576302
## attr(,"assign")
## [1] 0 1 1
```

Create a prediction method `g` that takes in a vector `x_star` and `my_simple_ols_obj`, an object of type `my_simple_ols_obj` and predicts `y` values for each entry in `x_star`.

```
g = function(my_simple_ols_obj, x_star){
  my_simple_ols_obj$b_0 + my_simple_ols_obj$b_1 *x_star
}
```

Use this function to verify that when predicting for the average `x`, you get the average `y`.

```
expect_equal(g(my_simple_ols_mod , mean(x)), mean(y))
```

In class we spoke about error due to ignorance, misspecification error and estimation error. Show that as `n` grows, estimation error shrinks. Let us define an error metric that is the difference between `b_0` and `b_1` and `beta_0` and `beta_1`. How about  $\|b - \beta\|^2$  where the quantities are now the vectors of size two. Show as `n` increases, this shrinks.

```
beta_0 = 3
beta_1 = -2
beta = c(beta_0, beta_1)
ns = 10^(1:8)
```

```

errors_in_beta = array(NA, length(ns))
for (i in 1 : length(ns)) {
  n = ns[i]
  x = runif(n)
  h_star_x = beta_0 + beta_1 * x
  epsilon = rnorm(n, mean = 0, sd = 0.33)
  y = h_star_x + epsilon
  mod = my_simple_ols(x,y)

  errors_in_beta[i] = (mod$b_0 - beta_0)^2 + (mod$b_1 - beta_1)^2
}
rbind(ns, errors_in_beta)

##           [,1]           [,2]           [,3]           [,4]
## ns          10.0000000 100.0000000 1.000000e+03 1.000000e+04
1.000000e+05
## errors_in_beta 0.4466601 0.0109519 7.889023e-05 7.702694e-04 4.049435e-
06
##           [,6]           [,7]           [,8]
## ns          1.000000e+06 1.000000e+07 1.000000e+08
## errors_in_beta 9.057943e-07 1.22165e-08 4.700748e-09

```

We are now going to repeat one of the first linear model building exercises in history — that of Sir Francis Galton in 1886. First load up package HistData.

```
pacman::p_load(HistData)
```

In it, there is a dataset called Galton. Load it up.

```
data(Galton)
```

You now should have a data frame in your workspace called Galton. Summarize this data frame and write a few sentences about what you see. Make sure you report n, p and a bit about what the columns represent and how the data was measured. See the help file ?Galton. p is 1 and n is 928 the number of observations

```
pacman::p_load(skimr)
skim(Galton)
```

#### Data summary

Name	Galton
Number of rows	928
Number of columns	2

---



#### Column type frequency:

numeric	2
---------	---

---

Group variables            None

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p2.5	p5.0	p7.5	p10.0	hist
parent	0	1	68.31	1.79	64.0	67.5	68.5	69.5	73.0	
child	0	1	68.09	2.52	61.7	66.2	68.2	70.2	73.7	

TO-DO

Find the average height (include both parents and children in this computation).

```
y = c(Galton$parent, Galton$child)
y_bar = sum(y)/(2*nrow(Galton))
```

If you were predicting child height from parent and you were using the null model, what would the RMSE be of this model be?

```
SSE_0 = sum((y-y_bar)^2) #SSE for Null model, also SST
sqrt(SSE_0/(2*nrow(Galton)-2))

## [1] 2.186179
```

Note that in Math 241 you learned that the sample average is an estimate of the “mean”, the population expected value of height. We will call the average the “mean” going forward since it is probably correct to the nearest tenth of an inch with this amount of data.

Run a linear model attempting to explain the childrens’ height using the parents’ height. Use `lm` and use the R formula notation. Compute and report `b_0`, `b_1`, RMSE and  $R^2$ .

```
ols_obj = my_simple_ols(Galton$parent,Galton$child)

## Warning in (x - x_bar) * (y - y_bar): longer object length is not a
multiple of
## shorter object length

## Warning in y - y_hat: longer object length is not a multiple of shorter
object
## length

ols_obj$b_0

## [1] -0.0002109397

ols_obj$b_1

## [1] 0.001685508
```

Interpret all four quantities:  $b_0$ ,  $b_1$ , RMSE and  $R^2$ . Use the correct units of these metrics in your answer.  $b_0$  is the intercept  $b_1$  is the slope RMSE  $R^2$

TO-DO:  $b_0$  is the intercept,  $b_1$  is the slope, in this case it is about .646 inches, meaning for every one inch increase in the parents height, the child is about .646 inches taller.

How good is this model? How well does it predict? Discuss.

TO-DO. I think that there is something wrong with my code, because I can not figure out how to compute the RMSE and R-squared, but if RMSE is high in this context, then the model would not be great because the error is too large to make good predictions. If there is a chance, for example, that in each prediction you would get within a 10 inch range of the actual height, that would not be a great model because its a pretty big gap.

It is reasonable to assume that parents and their children have the same height? Explain why this is reasonable using basic biology and common sense.

Yes, because there is a genetic component to height which is passed from parent to child.

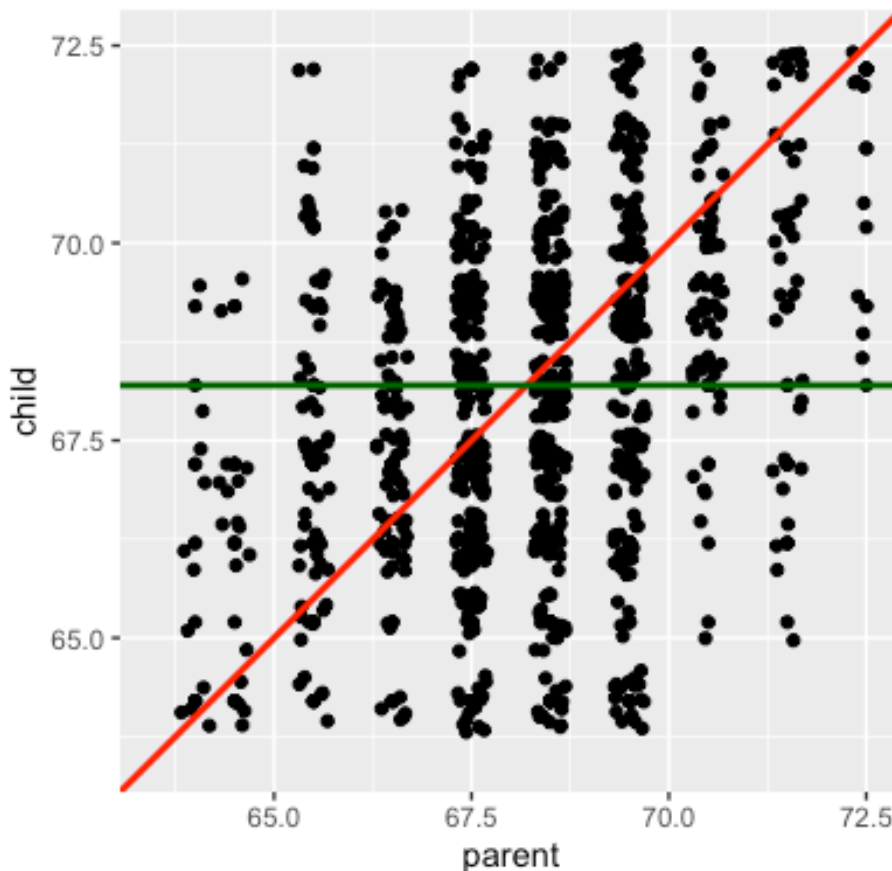
If they were to have the same height and any differences were just random noise with expectation 0, what would the values of  $\beta_0$  and  $\beta_1$  be?

Let's plot (a) the data in D as black dots, (b) your least squares line defined by  $b_0$  and  $b_1$  in blue, (c) the theoretical line  $\beta_0$  and  $\beta_1$  if the parent-child height equality held in red and (d) the mean height in green.

```
pacman::p_load(ggplot2)
ggplot(Galton, aes(x = parent, y = child)) +
  geom_point() +
  geom_jitter() +
  geom_abline(intercept = beta_0, slope = beta_1, color = "blue", size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = y_bar, slope = 0, color = "darkgreen", size = 1) +
  xlim(63.5, 72.5) +
  ylim(63.5, 72.5) +
  coord_equal(ratio = 1)

## Warning: Removed 76 rows containing missing values (geom_point).
## Warning: Removed 85 rows containing missing values (geom_point).
```





Fill in the following sentence:

TO-DO: Children of short parents became taller on average and children of tall parents became shorter on average.

Why did Galton call it “Regression towards mediocrity in hereditary stature” which was later shortened to “regression to the mean”?

TO DO: Precisely because of the fact above. Children of short parents become taller, children of tall parents become shorter, so on average, the heights of children regresses to the mean.

Why should this effect be real?

TO-DO: Because every child gets a mixture of genes from both parents including genes that may have been recessive in their parents.

You now have unlocked the mystery. Why is it that when modeling with  $y$  continuous, everyone calls it “regression”? Write a better, more descriptive and appropriate name for building predictive models with  $y$  continuous.

TO-DO: Everyone calls it regression because of Galton’s data and his findings which he called “regression to the mean.” OLS would be a better name because it actually minimizes the error in your model.

You can now clear the workspace.

```
rm(list = ls())
```

Create a dataset D which we call *Xy* such that the linear model has  $R^2$  about 50% and RMSE approximately 1.

```
x = 1:5
y = x^30
Xy = data.frame(x = x, y = y)
model = lm(x~y)
summary(model)$Rsqr

## NULL

summary(model)$sigma

## [1] 1.289795
```

Create a dataset D which we call *Xy* such that the linear model has  $R^2$  about 0% but *x*, *y* are clearly associated.

```
x = (-100:100)
y = x^2
Xy = data.frame(x = x, y = y)
model = lm(y ~ x)
summary(model)$Rsqr

## NULL
```

Extra credit but required for 650 students: create a dataset D and a model that can give you  $R^2$  arbitrarily close to 1 i.e. approximately  $1 - \epsilon$  but RMSE arbitrarily high i.e. approximately *M*.

```
epsilon = 0.01
M = 1000
#TO-DO
```

Write a function `my_ols` that takes in *X*, a matrix with *p* columns representing the feature measurements for each of the *n* units, a vector of *n* responses *y* and returns a list that contains the *b*, the *p*+1-sized column vector of OLS coefficients, *yhat* (the vector of *n* predictions), *e* (the vector of *n* residuals), *df* for degrees of freedom of the model, *SSE*, *SST*, *MSE*, *RMSE* and *Rsqr* (for the R-squared metric). Internally, you cannot use `lm` or any other package; it must be done manually. You should throw errors if the inputs are non-numeric or not the same length. Or if *X* is not otherwise suitable. You should also name the class of the return value `my_ols` by using the `class` function as a setter. No need to create R Oxygen documentation here.

```
my_ols = function(X, y){ #same as above but with matrices and matrix
multiplication
  n = nrow(X)
```

```

if(!is.numeric(X) && !is.integer(X)){
  stop("X is not numeric.")
}
p = ncol(X)
X = cbind(1, X)
Xt = t(X)
XtX = Xt %*% X
XtXinv = solve(XtX)
b = solve(XtX) %*% Xt %*% y
y_hat = X %*% b
e = y - y_hat
df = p + 1
if(n != nrow(X)){
  stop("X rows and length of y need to be the same length.")
}
if(class(y) != 'numeric' && class(y) != 'integer'){
  stop("y needs to be numeric.")
}
if(n <= ncol(X)+1){
  stop("n must be more than 2.")
}
SSE = sum(e^2)
SST = var(y) * (n-1)
MSE = SSE / n - df
RMSE = sqrt(MSE)
Rsqr = 1 - SSE/SST
ols = list()
ols$b=b
ols$y_hat = y_hat
ols$e = e
ols$df =df
ols$SSE = SSE
ols$SST = SST
ols$MSE = MSE
ols$RMSE = RMSE
ols$Rsqr = Rsqr
ols
}

```

Verify that the OLS coefficients for the Type of cars in the cars dataset gives you the same results as we did in class (i.e. the  $\bar{y}$ 's within group).

```

X = model.matrix(~Type, MASS::Cars93)[,-1] #takes away one of the linearly
dependent columns
y = MASS::Cars93$Price
my_ols(X,y)

## $b
##           [,1]
##          18.212500

```

```
## TypeLarge      6.087500
## TypeMidsize    9.005682
## TypeSmall      -8.045833
## TypeSporty     1.180357
## TypeVan        0.887500
##
## $y_hat
##      [,1]
## 1  10.16667
## 2  27.21818
## 3  18.21250
## 4  27.21818
## 5  27.21818
## 6  27.21818
## 7  24.30000
## 8  24.30000
## 9  27.21818
## 10 24.30000
## 11 27.21818
## 12 18.21250
## 13 18.21250
## 14 19.39286
## 15 27.21818
## 16 19.10000
## 17 19.10000
## 18 24.30000
## 19 19.39286
## 20 24.30000
## 21 18.21250
## 22 24.30000
## 23 10.16667
## 24 10.16667
## 25 18.21250
## 26 19.10000
## 27 27.21818
## 28 19.39286
## 29 10.16667
## 30 24.30000
## 31 10.16667
## 32 10.16667
## 33 18.21250
## 34 19.39286
## 35 19.39286
## 36 19.10000
## 37 27.21818
## 38 24.30000
## 39 10.16667
## 40 19.39286
## 41 19.39286
## 42 10.16667
```

## 43 18.21250  
## 44 10.16667  
## 45 10.16667  
## 46 19.39286  
## 47 27.21818  
## 48 27.21818  
## 49 27.21818  
## 50 27.21818  
## 51 27.21818  
## 52 24.30000  
## 53 10.16667  
## 54 10.16667  
## 55 18.21250  
## 56 19.10000  
## 57 19.39286  
## 58 18.21250  
## 59 27.21818  
## 60 19.39286  
## 61 27.21818  
## 62 10.16667  
## 63 27.21818  
## 64 10.16667  
## 65 18.21250  
## 66 19.10000  
## 67 27.21818  
## 68 18.21250  
## 69 27.21818  
## 70 19.10000  
## 71 24.30000  
## 72 19.39286  
## 73 10.16667  
## 74 18.21250  
## 75 19.39286  
## 76 27.21818  
## 77 24.30000  
## 78 18.21250  
## 79 10.16667  
## 80 10.16667  
## 81 10.16667  
## 82 18.21250  
## 83 10.16667  
## 84 10.16667  
## 85 19.39286  
## 86 27.21818  
## 87 19.10000  
## 88 10.16667  
## 89 19.10000  
## 90 18.21250  
## 91 19.39286  
## 92 18.21250

```
## 93 27.21818
##
## $e
##      [,1]
## 1  5.733333e+00
## 2  6.681818e+00
## 3  1.088750e+01
## 4  1.048182e+01
## 5  2.781818e+00
## 6 -1.151818e+01
## 7 -3.500000e+00
## 8 -6.000000e-01
## 9 -9.181818e-01
## 10 1.040000e+01
## 11 1.288182e+01
## 12 -4.812500e+00
## 13 -6.812500e+00
## 14 -4.292857e+00
## 15 -1.131818e+01
## 16 -2.800000e+00
## 17 -2.500000e+00
## 18 -5.500000e+00
## 19 1.860714e+01
## 20 -5.900000e+00
## 21 -2.412500e+00
## 22 5.200000e+00
## 23 -9.666667e-01
## 24 1.133333e+00
## 25 -4.912500e+00
## 26 -1.000000e-01
## 27 -1.161818e+01
## 28 6.407143e+00
## 29 2.033333e+00
## 30 -5.000000e+00
## 31 -2.766667e+00
## 32 -6.666667e-02
## 33 -6.912500e+00
## 34 -3.492857e+00
## 35 -5.392857e+00
## 36 8.000000e-01
## 37 -7.018182e+00
## 38 -3.400000e+00
## 39 -1.766667e+00
## 40 -6.892857e+00
## 41 4.071429e-01
## 42 1.933333e+00
## 43 -7.125000e-01
## 44 -2.166667e+00
## 45 -1.666667e-01
## 46 -9.392857e+00
```

```
## 47 -1.331818e+01
## 48 2.068182e+01
## 49 7.818182e-01
## 50 7.981818e+00
## 51 7.081818e+00
## 52 1.180000e+01
## 53 -1.866667e+00
## 54 1.433333e+00
## 55 -1.712500e+00
## 56 -3.552714e-15
## 57 1.310714e+01
## 58 1.368750e+01
## 59 3.468182e+01
## 60 -5.292857e+00
## 61 -1.231818e+01
## 62 1.333333e-01
## 63 -1.118182e+00
## 64 1.633333e+00
## 65 -2.512500e+00
## 66 -3.552714e-15
## 67 -5.718182e+00
## 68 -4.712500e+00
## 69 -1.091818e+01
## 70 4.000000e-01
## 71 -3.600000e+00
## 72 -4.992857e+00
## 73 -1.166667e+00
## 74 -7.112500e+00
## 75 -1.692857e+00
## 76 -8.718182e+00
## 77 1.000000e-01
## 78 1.048750e+01
## 79 9.333333e-01
## 80 -1.766667e+00
## 81 7.333333e-01
## 82 1.287500e+00
## 83 -1.566667e+00
## 84 -3.666667e-01
## 85 -9.928571e-01
## 86 -9.018182e+00
## 87 3.600000e+00
## 88 -1.066667e+00
## 89 6.000000e-01
## 90 1.787500e+00
## 91 3.907143e+00
## 92 4.487500e+00
## 93 -5.181818e-01
##
## $df
## [1] 6
```

```
##
## $SSE
## [1] 5162.586
##
## $SST
## [1] 8584.021
##
## $MSE
## [1] 49.51168
##
## $RMSE
## [1] 7.036454
##
## $Rsqr
## [1] 0.3985819
```

Create a prediction method `g` that takes in a vector `x_star` and the dataset `D` i.e. `X` and `y` and returns the OLS predictions. Let `X` be a matrix with with `p` columns representing the feature measurements for each of the `n` units. Do not use the “`my_ols`” function.

```
g = function(x_star, X, y){
  X = cbind(1, X)
  Xt = t(X)
  XtX = Xt %*% X
  XtXinv = solve(XtX)
  b = solve(XtX) %*% Xt %*% y
  cat(length(b), "/n")
  cat(length(x_star), "/n")
  c(1, x_star) %*% b
}
g(X[7, , drop = FALSE], X, y)

## 6 /n5 /n

##      [,1]
## [1,] 24.3
```