

Lab 3

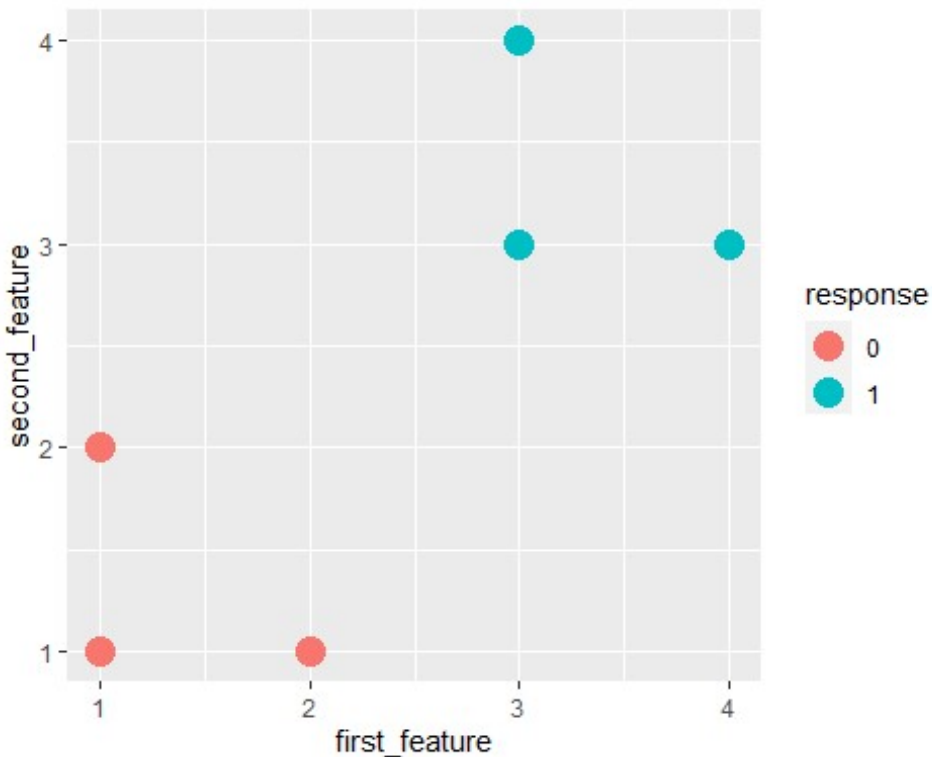
Enoch Kim

11:59PM March 4, 2021

Support Vector Machine vs. Perceptron

We recreate the data from the previous lab and visualize it:

```
pacman::p_load(ggplot2)
Xy_simple = data.frame(
  response = factor(c(0, 0, 0, 1, 1, 1)), #nominal
  first_feature = c(1, 1, 2, 3, 3, 4), #continuous
  second_feature = c(1, 2, 1, 3, 4, 3) #continuous
)
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature,
color = response)) +
  geom_point(size = 5)
simple_viz_obj
```

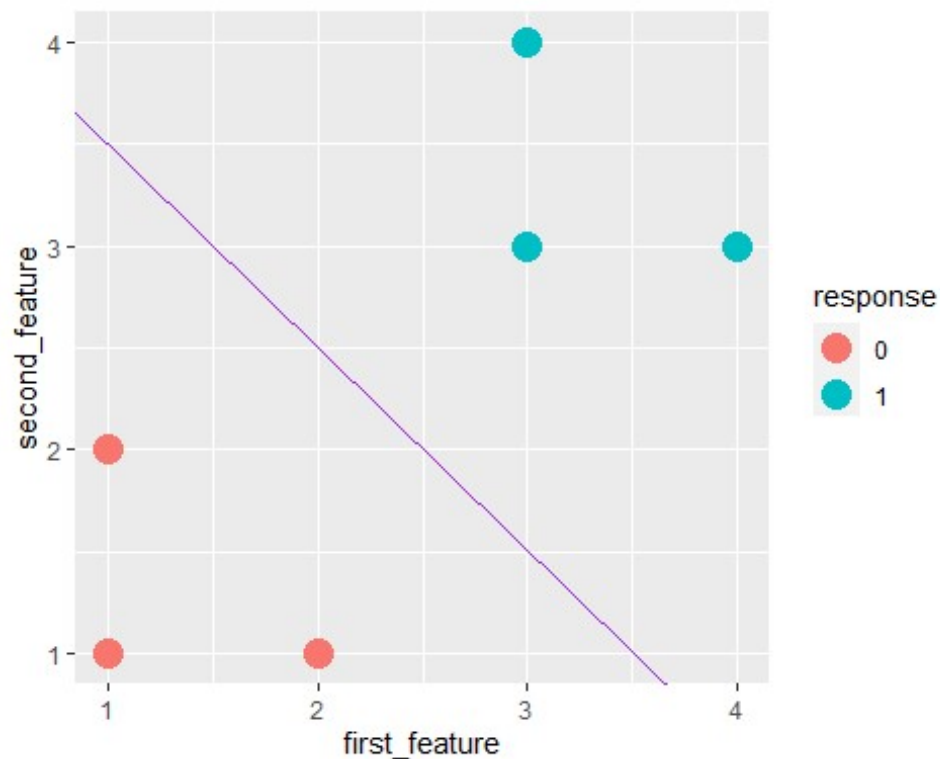


Use the `e1071` package to fit an SVM model to the simple data. Use a formula to create the model, pass in the data frame, set kernel to be `linear` for the linear SVM and don't scale the covariates. Call the model object `svm_model1`. Otherwise the remaining code won't work.

```
pacman::p_load(e1071)
svm_model = svm(
  formula = Xy_simple$response ~.,
  data = Xy_simple,
  kernel = "linear",
  scale = FALSE
)
```

and then use the following code to visualize the line in purple:

```
w_vec_simple_svm = c(
  svm_model$rho, #the b term
  -t(svm_model$coefs) %*% cbind(Xy_simple$first_feature,
Xy_simple$second_feature)[svm_model$index, ] # the other terms
)
simple_svm_line = geom_abline(
  intercept = -w_vec_simple_svm[1] / w_vec_simple_svm[3],
  slope = -w_vec_simple_svm[2] / w_vec_simple_svm[3],
  color = "purple")
simple_viz_obj + simple_svm_line
```



Source the `perceptron_learning_algorithm` function from lab 2. Then run the following to fit the perceptron and plot its line in orange with the SVM's line:

```
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w
= NULL){
```

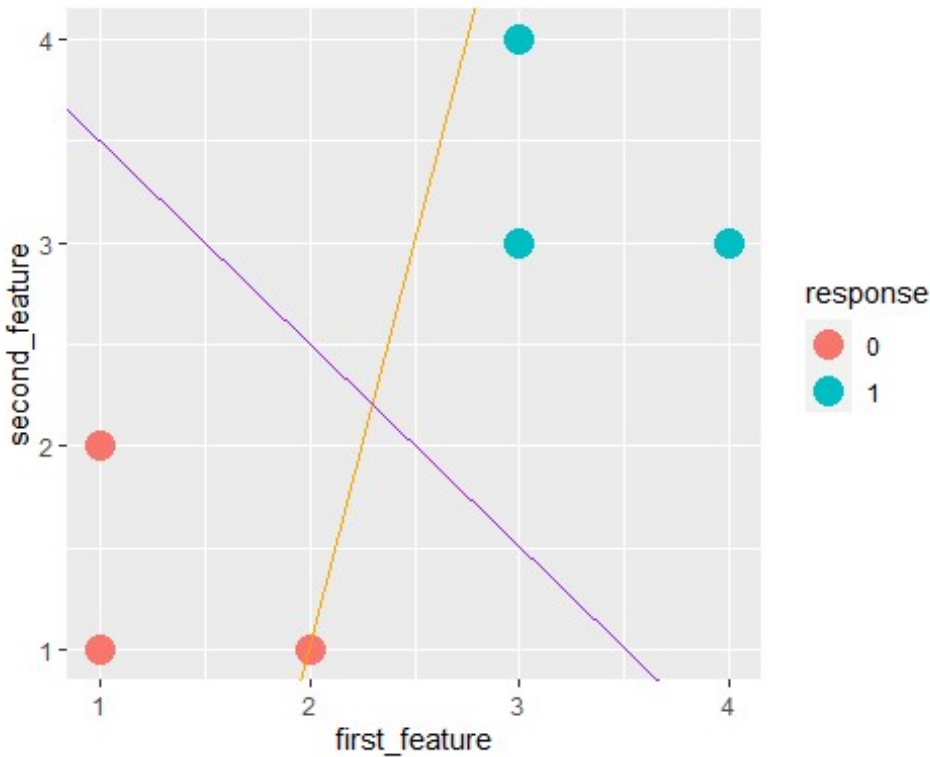
```

Xinput = as.matrix(cbind(1,Xinput))
p = ncol(Xinput)
w = rep(0, p)
r = nrow(Xinput)

for (iter in 1 : MAX_ITER){
  for (i in 1 : r) {
    x_i = Xinput[i, ]
    yhat_i = ifelse(sum(x_i * w) > 0, 1, 0)
    y_i = y_binary[i]
    for(j in 1:p){
      w[j] = w[j] + (y_i - yhat_i) * x_i[j]
    }
  }
}
w

w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1)
)
simple_perceptron_line = geom_abline(
  intercept = -w_vec_simple_per[1] / w_vec_simple_per[3],
  slope = -w_vec_simple_per[2] / w_vec_simple_per[3],
  color = "orange")
simple_viz_obj + simple_perceptron_line + simple_svm_line

```



Is this SVM line a better fit than the perceptron?

Yes, this SVM line is a better fit than the perceptron, it is marginally acceptable.

Now write pseudocode for your own implementation of the linear support vector machine algorithm using the Vapnik objective function we discussed.

Note there are differences between this spec and the perceptron learning algorithm spec in question #1. You should figure out a way to respect the MAX_ITER argument value.

```
## Support Vector Machine
#
## This function implements the hinge-loss + maximum margin linear support
vector machine algorithm of Vladimir Vapnik (1963).
#
## @param Xinput      The training data features as an n x p matrix.
## @param y_binary    The training data responses as a vector of length n
consisting of only 0's and 1's.
## @param MAX_ITER    The maximum number of iterations the algorithm
performs. Defaults to 5000.
## @param lambda      A scalar hyperparameter trading off margin of the
hyperplane versus average hinge loss.
##                    The default value is 1.
## @return            The computed final parameter (weight) as a vector of
length p + 1
linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000,
lambda = 0.1){
```

```

#Declaring the Sum of hinge error
# SHE = 0

# initialize a w vector
# w <- () (This something like this?)

#Iterate through the for-loop till the MAX_ITER
# for x in MAX_ITER{

#Iterate through each row
#   for i in nrow(Xinput){

#       #Adding the hinge error for each i to obtain total
#       SHE += max{0, (1/2) - (y_binary[i]-1/2)(w*Xinput[i]-b)}
#   }
# }
# To maximize the margin
# argmin{(SHE/n) + lambda(distance of w)^2}

# Returns w (the wedge/weights?)
# w
}

```

If you are enrolled in 342W the following is extra credit but if you're enrolled in 650, the following is required. Write the actual code. You may want to take a look at the `optimx` package. You can feel free to define another function (a "private" function) in this chunk if you wish. R has a way to create public and private functions, but I believe you need to create a package to do that (beyond the scope of this course).

```

#' This function implements the hinge-loss + maximum margin linear support
vector machine algorithm of Vladimir Vapnik (1963).
#'
#' @param Xinput      The training data features as an n x p matrix.
#' @param y_binary    The training data responses as a vector of length n
consisting of only 0's and 1's.
#' @param MAX_ITER    The maximum number of iterations the algorithm
performs. Defaults to 5000.
#' @param lambda      A scalar hyperparameter trading off margin of the
hyperplane versus average hinge loss.
#'                    The default value is 1.
#' @return            The computed final parameter (weight) as a vector of
length p + 1
#linear_svm_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 5000,
lambda = 0.1){
#}

```

If you wrote code (the extra credit), run your function using the defaults and plot it in brown vis-a-vis the previous model's line:

```
#svm_model_weights = linear_svm_learning_algorithm(Xinput, y_binary)
#my_svm_line = geom_abline(
#  intercept = svm_model_weights[1] / svm_model_weights[3], #NOTE: negative
#  slope = -svm_model_weights[2] / svm_model_weights[3],
#  color = "brown")
#simple_viz_obj + my_svm_line
```

Is this the same as what the e1071 implementation returned? Why or why not?

TO-DO

We now move on to simple linear modeling using the ordinary least squares algorithm.

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

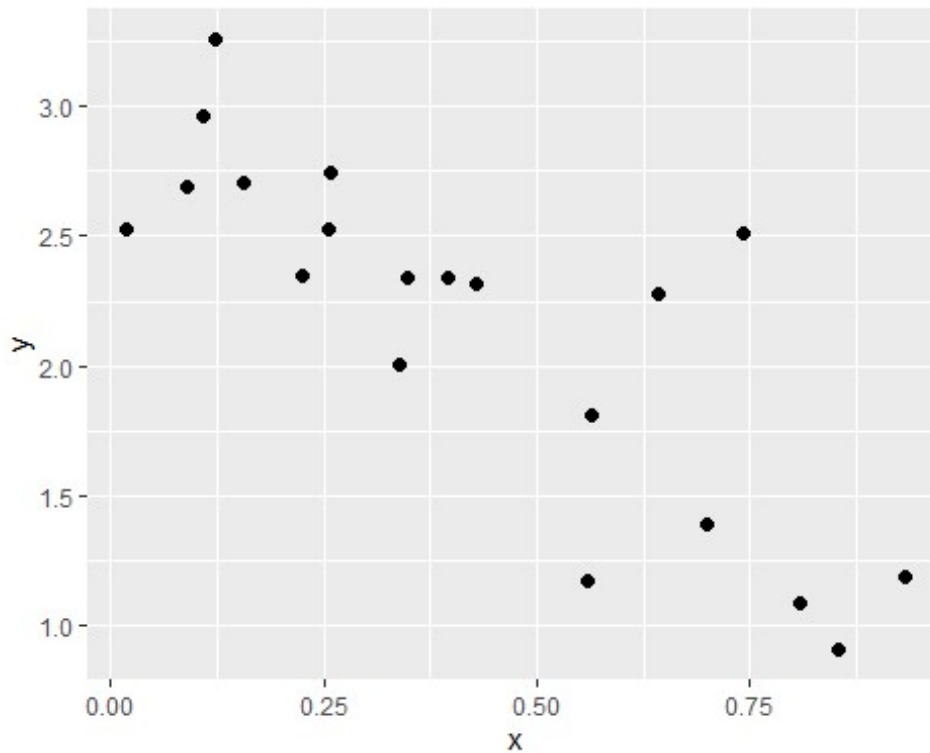
```
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

Compute $h^*(x)$ as `h_star_x`, then draw $\epsilon \sim N(0, 0.33^2)$ as `epsilon`, then compute y .

```
h_star_x = beta_0 + beta_1 * x
epsilon = rnorm(n, 0, 0.33)
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 β_0 and β_1 ?

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 ROxygen documentation here.

```
my_simple_ols = function(x, y){
  n = length(y)

  if(n != length(x)){
    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")
  }

  x_bar = sum(x) / n
  y_bar = sum(y) / n
  b_1 = (sum(x * y) - n * x_bar * y_bar) / (sum(x ^ 2) - n * x_bar ^ 2)
  b_0 = y_bar - b_1 * x_bar
  yhat = b_0 + b_1 * x
  e = y - yhat
  SSE = sum(e ^ 2)
  SST = sum((y - y_bar) ^ 2)
  MSE = SSE / (n - 2)
  RMSE = sqrt(MSE)
  Rsqrt = 1 - (SSE/SST)

  model = list(b_0 = b_0, b_1 = b_1, yhat = yhat, e = e, SSE = SSE, SST =
SST, MSE = MSE, RMSE = RMSE, Rsqrt = Rsqrt)

  class(model) = "my_simple_ols_obj"

  model
}

```

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$Rsqr, summary(lm_mod)$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$e)

## [1] 9.993041e-17

expect_equal(mean(my_simple_ols_mod$e), 0)

```

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
## 1             1 0.10924993
## 2             1 0.22476013
## 3             1 0.56309345
## 4             1 0.42804407
## 5             1 0.80875295
## 6             1 0.33911941
## 7             1 0.15550835
## 8             1 0.64240411
## 9             1 0.56008416
## 10            1 0.08981154
## 11            1 0.39491582
## 12            1 0.34775613
## 13            1 0.25622114
## 14            1 0.69927738
## 15            1 0.25742601
## 16            1 0.85305058
## 17            1 0.74138863
## 18            1 0.12251034
## 19            1 0.01860284
## 20            1 0.93226887
## attr(,"assign")
## [1] 0 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){
  y_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 β_0 and β_1 . How about $h = ||b - \beta||^2$ 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_betas = 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)
b = c(mod$b_0, mod$b_1)

errors_in_betas[i] = sum((beta - b)^2)
}

log(errors_in_betas, 10)

## [1] -1.295662 -2.290760 -2.793365 -5.560141 -5.290644 -6.943108 -7.651440
## [8] -7.613632

```

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	p25	p50	p75	p100	hist
parent	0	1	68.3	1.7	64.	67.	68.	69.	73.0	
			1	9	0	5	5	5		
child	0	1	68.0	2.5	61.	66.	68.	70.	73.7	
			9	2	7	2	2	2		

In Galton's data, I see the number of rows and columns within the data. Within the data, in the skim part. I see the parent and the child in the row, in the columns, I see the mean (parent is 68.3, child is 68.1), sd and see the percentile, 0, 25, 50, 75 and 100. n is the number of observations, p is the number of independent variable in this case it is height.

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

```
avg_height = mean(c(Galton$parent, Galton$child))
avg_height

## [1] 68.19833
```

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

```
n = nrow(Galton)
SST = sum((Galton$child - mean(Galton$child)) ^ 2)
sqrt(SST / (n - 1))

## [1] 2.517941
```

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 .

```
mod = lm(child ~ parent, Galton)
b_0 = coef(mod)[1]
b_1 = coef(mod)[2]
summary(mod)$sigma

## [1] 2.238547

summary(mod)$r.squared

## [1] 0.2104629
```

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

b_0 = If $x = 0$, then the parent has no height that means the parent does not exist (can't have zero height, in inches) b_1 = If there is a one inch increase by the parent's height, the child

average height will increase by the .64 inches. R^2 = This is about 21% variance explained which is considered low to certain standards. RMSE = This is Plus or minus 4.5 (2.23(2)) inches which makes it a 9 inches range 95% of the time

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

This model is pretty good because it only considers one feature which is height but 9 inches can be considered a large range because the confidence interval can be almost a foot(12 inches) off of the prediction.

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 is it reasonable to assume that parents and their children have the same height because children inherit both their parent's height genes which would have the similar height as their parents.

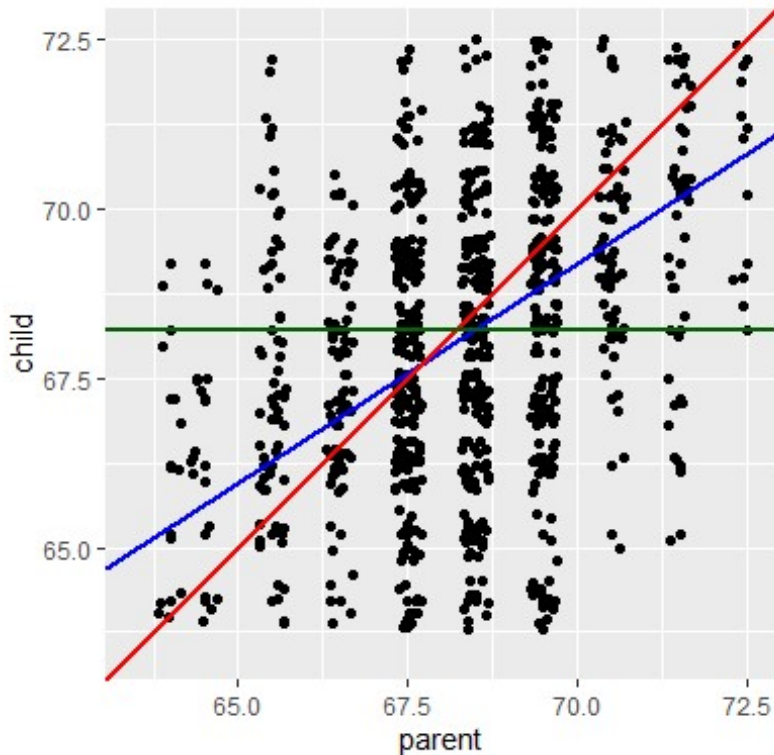
If they were to have the same height and any differences were just random noise with expectation 0, what would the values of β_0 and β_1 be?

The value of beta one is 1 and value of beta zero is 0.

Let's plot (a) the data in \mathbb{D} as black dots, (b) your least squares line defined by b_0 and b_1 in blue, (c) the theoretical line β_0 and β_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 = b_0, slope = b_1, color = "blue", size = 1) +
  geom_abline(intercept = 0, slope = 1, color = "red", size = 1) +
  geom_abline(intercept = avg_height, 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 86 rows containing missing values (geom_point).
```



Fill in the following sentence:

Children of short parents became tall on average and children of tall parents became short on average.

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

Galton called it because if the tall parents had short children and short parents had tall children, this led to a balance on height in the gene pool. The population by on height is regressing back to the mean.

Why should this effect be real?

This effect should be real because as time passes by, the height’s mean for the human population stays around the same point due to biology. If someone is taller or shorter than average, the population will regress to the mean.

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.

The reason why everyone calls it “regression” is because we expected the parent/child relationship to be around 1 to 1. However after running the test and viewing the results, the height average of the children seem to have regressed compared to the height average of parent. The children height average turned out to be lower than the parents height average.

You can now clear the workspace. Create a dataset \mathbb{D} which we call `Xy` such that the linear model as R^2 about 50% and RMSE approximately 1.

```
x = 1:5
y = x^25
Xy = data.frame(x = x, y = y)
model = lm(x~y)
summary(model)$r.squared

## [1] 0.5028341

summary(model)$sigma

## [1] 1.28733
```

Create a dataset \mathbb{D} which we call `Xy` such that the linear model as R^2 about 0% but `x, y` are clearly associated.

```
x = 1:400
y = x ^ 118
Xy = data.frame(x = x, y = y)
model = lm(x~y)
summary(model)$r.squared

## [1] 0.04978688
```

Extra credit: create a dataset \mathbb{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.

```
#From Office Hours

my_ols = function(X, y){

  n = length(y)

  if (!is.numeric(X) && !is.integer(X)) {
    stop("X is not numeric")
  }
}
```

```

}

X = cbind(rep(1, n), X)

p = ncol(X)

df = ncol(X)

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.")
}

y_bar = sum(y) / n

b = solve(t(X) %*% X) %*% t(X) %*% y

yhat = X %*% b

e = y - yhat

SSE = (t(e) %*% e)
SST = t(y - y_bar) %*% (y - y_bar)
MSE = SSE / (n - (p + 1))
RMSE = sqrt(MSE)
RSQ = 1 - (SSE / SST)

model = list(b = b, yhat = yhat, df = df, e = e, SSE = SSE, SST = SST, MSE
= MSE, RMSE = RMSE, RSQ = RSQ)

class(model) = "my_ols_obj"

model
}

```

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 ybar's within group).

```

cars = MASS::Cars93
mod = lm(Price~Type, data=cars)
my_ols(as.numeric(data.matrix(data.frame((cars$Type)))), cars$Price)

```

```
## $b
##      [,1]
## 22.871020
## X -1.001939
##
## $yhat
##      [,1]
## [1,] 18.86327
## [2,] 19.86520
## [3,] 21.86908
## [4,] 19.86520
## [5,] 19.86520
## [6,] 19.86520
## [7,] 20.86714
## [8,] 20.86714
## [9,] 19.86520
## [10,] 20.86714
## [11,] 19.86520
## [12,] 21.86908
## [13,] 21.86908
## [14,] 17.86133
## [15,] 19.86520
## [16,] 16.85939
## [17,] 16.85939
## [18,] 20.86714
## [19,] 17.86133
## [20,] 20.86714
## [21,] 21.86908
## [22,] 20.86714
## [23,] 18.86327
## [24,] 18.86327
## [25,] 21.86908
## [26,] 16.85939
## [27,] 19.86520
## [28,] 17.86133
## [29,] 18.86327
## [30,] 20.86714
## [31,] 18.86327
## [32,] 18.86327
## [33,] 21.86908
## [34,] 17.86133
## [35,] 17.86133
## [36,] 16.85939
## [37,] 19.86520
## [38,] 20.86714
## [39,] 18.86327
## [40,] 17.86133
## [41,] 17.86133
## [42,] 18.86327
## [43,] 21.86908
```


[44,] 18.86327
[45,] 18.86327
[46,] 17.86133
[47,] 19.86520
[48,] 19.86520
[49,] 19.86520
[50,] 19.86520
[51,] 19.86520
[52,] 20.86714
[53,] 18.86327
[54,] 18.86327
[55,] 21.86908
[56,] 16.85939
[57,] 17.86133
[58,] 21.86908
[59,] 19.86520
[60,] 17.86133
[61,] 19.86520
[62,] 18.86327
[63,] 19.86520
[64,] 18.86327
[65,] 21.86908
[66,] 16.85939
[67,] 19.86520
[68,] 21.86908
[69,] 19.86520
[70,] 16.85939
[71,] 20.86714
[72,] 17.86133
[73,] 18.86327
[74,] 21.86908
[75,] 17.86133
[76,] 19.86520
[77,] 20.86714
[78,] 21.86908
[79,] 18.86327
[80,] 18.86327
[81,] 18.86327
[82,] 21.86908
[83,] 18.86327
[84,] 18.86327
[85,] 17.86133
[86,] 19.86520
[87,] 16.85939
[88,] 18.86327
[89,] 16.85939
[90,] 21.86908
[91,] 17.86133
[92,] 21.86908
[93,] 19.86520

```
##
## $df
## [1] 2
##
## $e
##          [,1]
## [1,] -2.96326531
## [2,] 14.03479592
## [3,]  7.23091837
## [4,] 17.83479592
## [5,] 10.13479592
## [6,] -4.16520408
## [7,] -0.06714286
## [8,]  2.83285714
## [9,]  6.43479592
## [10,] 13.83285714
## [11,] 20.23479592
## [12,] -8.46908163
## [13,] -10.46908163
## [14,] -2.76132653
## [15,] -3.96520408
## [16,] -0.55938776
## [17,] -0.25938776
## [18,] -2.06714286
## [19,] 20.13867347
## [20,] -2.46714286
## [21,] -6.06908163
## [22,]  8.63285714
## [23,] -9.66326531
## [24,] -7.56326531
## [25,] -8.56908163
## [26,]  2.14061224
## [27,] -4.26520408
## [28,]  7.93867347
## [29,] -6.66326531
## [30,] -1.56714286
## [31,] -11.46326531
## [32,] -8.76326531
## [33,] -10.56908163
## [34,] -1.96132653
## [35,] -3.86132653
## [36,]  3.04061224
## [37,]  0.33479592
## [38,]  0.03285714
## [39,] -10.46326531
## [40,] -5.36132653
## [41,]  1.93867347
## [42,] -6.76326531
## [43,] -4.36908163
## [44,] -10.86326531
```

```
## [45,] -8.86326531
## [46,] -7.86132653
## [47,] -5.96520408
## [48,] 28.03479592
## [49,] 8.13479592
## [50,] 15.33479592
## [51,] 14.43479592
## [52,] 15.23285714
## [53,] -10.56326531
## [54,] -7.26326531
## [55,] -5.36908163
## [56,] 2.24061224
## [57,] 14.63867347
## [58,] 10.03091837
## [59,] 42.03479592
## [60,] -3.76132653
## [61,] -4.96520408
## [62,] -8.56326531
## [63,] 6.23479592
## [64,] -7.06326531
## [65,] -6.16908163
## [66,] 2.24061224
## [67,] 1.63479592
## [68,] -8.36908163
## [69,] -3.56520408
## [70,] 2.64061224
## [71,] -0.16714286
## [72,] -3.46132653
## [73,] -9.86326531
## [74,] -10.76908163
## [75,] -0.16132653
## [76,] -1.36520408
## [77,] 3.53285714
## [78,] 6.83091837
## [79,] -7.76326531
## [80,] -10.46326531
## [81,] -7.96326531
## [82,] -2.36908163
## [83,] -10.26326531
## [84,] -9.06326531
## [85,] 0.53867347
## [86,] -1.66520408
## [87,] 5.84061224
## [88,] -9.76326531
## [89,] 2.84061224
## [90,] -1.86908163
## [91,] 5.43867347
## [92,] 0.83091837
## [93,] 6.83479592
##
```

```
## $SSE
##          [,1]
## [1,] 8361.872
##
## $SST
##          [,1]
## [1,] 8584.021
##
## $MSE
##          [,1]
## [1,] 92.90969
##
## $RMSE
##          [,1]
## [1,] 9.638967
##
## $RSQ
##          [,1]
## [1,] 0.02587939
##
## attr(,"class")
## [1] "my_ols_obj"
```

cars

	Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city
## 1	Acura	Integra	Small	12.9	15.9	18.8	25
## 2	Acura	Legend	Midsize	29.2	33.9	38.7	18
## 3	Audi	90	Compact	25.9	29.1	32.3	20
## 4	Audi	100	Midsize	30.8	37.7	44.6	19
## 5	BMW	535i	Midsize	23.7	30.0	36.2	22
## 6	Buick	Century	Midsize	14.2	15.7	17.3	22
## 7	Buick	LeSabre	Large	19.9	20.8	21.7	19
## 8	Buick	Roadmaster	Large	22.6	23.7	24.9	16
## 9	Buick	Riviera	Midsize	26.3	26.3	26.3	19
## 10	Cadillac	DeVille	Large	33.0	34.7	36.3	16
## 11	Cadillac	Seville	Midsize	37.5	40.1	42.7	16
## 12	Chevrolet	Cavalier	Compact	8.5	13.4	18.3	25
## 13	Chevrolet	Corsica	Compact	11.4	11.4	11.4	25
## 14	Chevrolet	Camaro	Sporty	13.4	15.1	16.8	19
## 15	Chevrolet	Lumina	Midsize	13.4	15.9	18.4	21
## 16	Chevrolet	Lumina_APV	Van	14.7	16.3	18.0	18
## 17	Chevrolet	Astro	Van	14.7	16.6	18.6	15
## 18	Chevrolet	Caprice	Large	18.0	18.8	19.6	17
## 19	Chevrolet	Corvette	Sporty	34.6	38.0	41.5	17
## 20	Chrylser	Concorde	Large	18.4	18.4	18.4	20
## 21	Chrysler	LeBaron	Compact	14.5	15.8	17.1	23
## 22	Chrysler	Imperial	Large	29.5	29.5	29.5	20
## 23	Dodge	Colt	Small	7.9	9.2	10.6	29
## 24	Dodge	Shadow	Small	8.4	11.3	14.2	23

## 25	Dodge	Spirit	Compact	11.9	13.3	14.7	22
## 26	Dodge	Caravan	Van	13.6	19.0	24.4	17
## 27	Dodge	Dynasty	Midsize	14.8	15.6	16.4	21
## 28	Dodge	Stealth	Sporty	18.5	25.8	33.1	18
## 29	Eagle	Summit	Small	7.9	12.2	16.5	29
## 30	Eagle	Vision	Large	17.5	19.3	21.2	20
## 31	Ford	Festiva	Small	6.9	7.4	7.9	31
## 32	Ford	Escort	Small	8.4	10.1	11.9	23
## 33	Ford	Tempo	Compact	10.4	11.3	12.2	22
## 34	Ford	Mustang	Sporty	10.8	15.9	21.0	22
## 35	Ford	Probe	Sporty	12.8	14.0	15.2	24
## 36	Ford	Aerostar	Van	14.5	19.9	25.3	15
## 37	Ford	Taurus	Midsize	15.6	20.2	24.8	21
## 38	Ford	Crown_Victoria	Large	20.1	20.9	21.7	18
## 39	Geo	Metro	Small	6.7	8.4	10.0	46
## 40	Geo	Storm	Sporty	11.5	12.5	13.5	30
## 41	Honda	Prelude	Sporty	17.0	19.8	22.7	24
## 42	Honda	Civic	Small	8.4	12.1	15.8	42
## 43	Honda	Accord	Compact	13.8	17.5	21.2	24
## 44	Hyundai	Excel	Small	6.8	8.0	9.2	29
## 45	Hyundai	Elantra	Small	9.0	10.0	11.0	22
## 46	Hyundai	Scoupe	Sporty	9.1	10.0	11.0	26
## 47	Hyundai	Sonata	Midsize	12.4	13.9	15.3	20
## 48	Infiniti	Q45	Midsize	45.4	47.9	50.4	17
## 49	Lexus	ES300	Midsize	27.5	28.0	28.4	18
## 50	Lexus	SC300	Midsize	34.7	35.2	35.6	18
## 51	Lincoln	Continental	Midsize	33.3	34.3	35.3	17
## 52	Lincoln	Town_Car	Large	34.4	36.1	37.8	18
## 53	Mazda	323	Small	7.4	8.3	9.1	29
## 54	Mazda	Protege	Small	10.9	11.6	12.3	28
## 55	Mazda	626	Compact	14.3	16.5	18.7	26
## 56	Mazda	MPV	Van	16.6	19.1	21.7	18
## 57	Mazda	RX-7	Sporty	32.5	32.5	32.5	17
## 58	Mercedes-Benz	190E	Compact	29.0	31.9	34.9	20
## 59	Mercedes-Benz	300E	Midsize	43.8	61.9	80.0	19
## 60	Mercury	Capri	Sporty	13.3	14.1	15.0	23
## 61	Mercury	Cougar	Midsize	14.9	14.9	14.9	19
## 62	Mitsubishi	Mirage	Small	7.7	10.3	12.9	29
## 63	Mitsubishi	Diamante	Midsize	22.4	26.1	29.9	18
## 64	Nissan	Sentra	Small	8.7	11.8	14.9	29
## 65	Nissan	Altima	Compact	13.0	15.7	18.3	24
## 66	Nissan	Quest	Van	16.7	19.1	21.5	17
## 67	Nissan	Maxima	Midsize	21.0	21.5	22.0	21
## 68	Oldsmobile	Achieva	Compact	13.0	13.5	14.0	24
## 69	Oldsmobile	Cutlass_Ciera	Midsize	14.2	16.3	18.4	23
## 70	Oldsmobile	Silhouette	Van	19.5	19.5	19.5	18
## 71	Oldsmobile	Eighty-Eight	Large	19.5	20.7	21.9	19
## 72	Plymouth	Laser	Sporty	11.4	14.4	17.4	23
## 73	Pontiac	LeMans	Small	8.2	9.0	9.9	31
## 74	Pontiac	Sunbird	Compact	9.4	11.1	12.8	23

## 75	Pontiac	Firebird	Sporty	14.0	17.7	21.4	19
## 76	Pontiac	Grand_Prix	Midsize	15.4	18.5	21.6	19
## 77	Pontiac	Bonneville	Large	19.4	24.4	29.4	19
## 78	Saab	900	Compact	20.3	28.7	37.1	20
## 79	Saturn	SL	Small	9.2	11.1	12.9	28
## 80	Subaru	Justy	Small	7.3	8.4	9.5	33
## 81	Subaru	Loyale	Small	10.5	10.9	11.3	25
## 82	Subaru	Legacy	Compact	16.3	19.5	22.7	23
## 83	Suzuki	Swift	Small	7.3	8.6	10.0	39
## 84	Toyota	Tercel	Small	7.8	9.8	11.8	32
## 85	Toyota	Celica	Sporty	14.2	18.4	22.6	25
## 86	Toyota	Camry	Midsize	15.2	18.2	21.2	22
## 87	Toyota	Previa	Van	18.9	22.7	26.6	18
## 88	Volkswagen	Fox	Small	8.7	9.1	9.5	25
## 89	Volkswagen	Eurovan	Van	16.6	19.7	22.7	17
## 90	Volkswagen	Passat	Compact	17.6	20.0	22.4	21
## 91	Volkswagen	Corrado	Sporty	22.9	23.3	23.7	18
## 92	Volvo	240	Compact	21.8	22.7	23.5	21
## 93	Volvo	850	Midsize	24.8	26.7	28.5	20
##	MPG.highway	AirBags	DriveTrain	Cylinders	EngineSize		
## 1	31	None	Front	4	1.8		
140							
## 2	25	Driver & Passenger	Front	6	3.2		
200							
## 3	26	Driver only	Front	6	2.8		
172							
## 4	26	Driver & Passenger	Front	6	2.8		
172							
## 5	30	Driver only	Rear	4	3.5		
208							
## 6	31	Driver only	Front	4	2.2		
110							
## 7	28	Driver only	Front	6	3.8		
170							
## 8	25	Driver only	Rear	6	5.7		
180							
## 9	27	Driver only	Front	6	3.8		
170							
## 10	25	Driver only	Front	8	4.9		
200							
## 11	25	Driver & Passenger	Front	8	4.6		
295							
## 12	36	None	Front	4	2.2		
110							
## 13	34	Driver only	Front	4	2.2		
110							
## 14	28	Driver & Passenger	Rear	6	3.4		
160							
## 15	29	None	Front	4	2.2		

110					
## 16	23	None	Front	6	3.8
170					
## 17	20	None	4WD	6	4.3
165					
## 18	26	Driver only	Rear	8	5.0
170					
## 19	25	Driver only	Rear	8	5.7
300					
## 20	28	Driver & Passenger	Front	6	3.3
153					
## 21	28	Driver & Passenger	Front	4	3.0
141					
## 22	26	Driver only	Front	6	3.3
147					
## 23	33	None	Front	4	1.5
92					
## 24	29	Driver only	Front	4	2.2
93					
## 25	27	Driver only	Front	4	2.5
100					
## 26	21	Driver only	4WD	6	3.0
142					
## 27	27	Driver only	Front	4	2.5
100					
## 28	24	Driver only	4WD	6	3.0
300					
## 29	33	None	Front	4	1.5
92					
## 30	28	Driver & Passenger	Front	6	3.5
214					
## 31	33	None	Front	4	1.3
63					
## 32	30	None	Front	4	1.8
127					
## 33	27	None	Front	4	2.3
96					
## 34	29	Driver only	Rear	4	2.3
105					
## 35	30	Driver only	Front	4	2.0
115					
## 36	20	Driver only	4WD	6	3.0
145					
## 37	30	Driver only	Front	6	3.0
140					
## 38	26	Driver only	Rear	8	4.6
190					
## 39	50	None	Front	3	1.0
55					
## 40	36	Driver only	Front	4	1.6

90					
## 41	31	Driver & Passenger	Front	4	2.3
160					
## 42	46	Driver only	Front	4	1.5
102					
## 43	31	Driver & Passenger	Front	4	2.2
140					
## 44	33	None	Front	4	1.5
81					
## 45	29	None	Front	4	1.8
124					
## 46	34	None	Front	4	1.5
92					
## 47	27	None	Front	4	2.0
128					
## 48	22	Driver only	Rear	8	4.5
278					
## 49	24	Driver only	Front	6	3.0
185					
## 50	23	Driver & Passenger	Rear	6	3.0
225					
## 51	26	Driver & Passenger	Front	6	3.8
160					
## 52	26	Driver & Passenger	Rear	8	4.6
210					
## 53	37	None	Front	4	1.6
82					
## 54	36	None	Front	4	1.8
103					
## 55	34	Driver only	Front	4	2.5
164					
## 56	24	None	4WD	6	3.0
155					
## 57	25	Driver only	Rear	rotary	1.3
255					
## 58	29	Driver only	Rear	4	2.3
130					
## 59	25	Driver & Passenger	Rear	6	3.2
217					
## 60	26	Driver only	Front	4	1.6
100					
## 61	26	None	Rear	6	3.8
140					
## 62	33	None	Front	4	1.5
92					
## 63	24	Driver only	Front	6	3.0
202					
## 64	33	Driver only	Front	4	1.6
110					
## 65	30	Driver only	Front	4	2.4

150						
## 66	23	None	Front	6	3.0	
151						
## 67	26	Driver only	Front	6	3.0	
160						
## 68	31	None	Front	4	2.3	
155						
## 69	31	Driver only	Front	4	2.2	
110						
## 70	23	None	Front	6	3.8	
170						
## 71	28	Driver only	Front	6	3.8	
170						
## 72	30	None	4WD	4	1.8	
92						
## 73	41	None	Front	4	1.6	
74						
## 74	31	None	Front	4	2.0	
110						
## 75	28	Driver & Passenger	Rear	6	3.4	
160						
## 76	27	None	Front	6	3.4	
200						
## 77	28	Driver & Passenger	Front	6	3.8	
170						
## 78	26	Driver only	Front	4	2.1	
140						
## 79	38	Driver only	Front	4	1.9	
85						
## 80	37	None	4WD	3	1.2	
73						
## 81	30	None	4WD	4	1.8	
90						
## 82	30	Driver only	4WD	4	2.2	
130						
## 83	43	None	Front	3	1.3	
70						
## 84	37	Driver only	Front	4	1.5	
82						
## 85	32	Driver only	Front	4	2.2	
135						
## 86	29	Driver only	Front	4	2.2	
130						
## 87	22	Driver only	4WD	4	2.4	
138						
## 88	33	None	Front	4	1.8	
81						
## 89	21	None	Front	5	2.5	
109						
## 90	30	None	Front	4	2.0	

134						
## 91	25	None	Front	6	2.8	
178						
## 92	28	Driver only	Rear	4	2.3	
114						
## 93	28	Driver & Passenger	Front	5	2.4	
168						
##	RPM	Rev.per.mile	Man.trans.avail	Fuel.tank.capacity	Passengers	Length
## 1	6300	2890	Yes	13.2	5	177
## 2	5500	2335	Yes	18.0	5	195
## 3	5500	2280	Yes	16.9	5	180
## 4	5500	2535	Yes	21.1	6	193
## 5	5700	2545	Yes	21.1	4	186
## 6	5200	2565	No	16.4	6	189
## 7	4800	1570	No	18.0	6	200
## 8	4000	1320	No	23.0	6	216
## 9	4800	1690	No	18.8	5	198
## 10	4100	1510	No	18.0	6	206
## 11	6000	1985	No	20.0	5	204
## 12	5200	2380	Yes	15.2	5	182
## 13	5200	2665	Yes	15.6	5	184
## 14	4600	1805	Yes	15.5	4	193
## 15	5200	2595	No	16.5	6	198
## 16	4800	1690	No	20.0	7	178
## 17	4000	1790	No	27.0	8	194
## 18	4200	1350	No	23.0	6	214
## 19	5000	1450	Yes	20.0	2	179
## 20	5300	1990	No	18.0	6	203
## 21	5000	2090	No	16.0	6	183
## 22	4800	1785	No	16.0	6	203
## 23	6000	3285	Yes	13.2	5	174
## 24	4800	2595	Yes	14.0	5	172
## 25	4800	2535	Yes	16.0	6	181
## 26	5000	1970	No	20.0	7	175
## 27	4800	2465	No	16.0	6	192
## 28	6000	2120	Yes	19.8	4	180
## 29	6000	2505	Yes	13.2	5	174
## 30	5800	1980	No	18.0	6	202
## 31	5000	3150	Yes	10.0	4	141
## 32	6500	2410	Yes	13.2	5	171
## 33	4200	2805	Yes	15.9	5	177
## 34	4600	2285	Yes	15.4	4	180
## 35	5500	2340	Yes	15.5	4	179
## 36	4800	2080	Yes	21.0	7	176
## 37	4800	1885	No	16.0	5	192
## 38	4200	1415	No	20.0	6	212
## 39	5700	3755	Yes	10.6	4	151
## 40	5400	3250	Yes	12.4	4	164
## 41	5800	2855	Yes	15.9	4	175
## 42	5900	2650	Yes	11.9	4	173

## 43	5600	2610	Yes	17.0	4	185
## 44	5500	2710	Yes	11.9	5	168
## 45	6000	2745	Yes	13.7	5	172
## 46	5550	2540	Yes	11.9	4	166
## 47	6000	2335	Yes	17.2	5	184
## 48	6000	1955	No	22.5	5	200
## 49	5200	2325	Yes	18.5	5	188
## 50	6000	2510	Yes	20.6	4	191
## 51	4400	1835	No	18.4	6	205
## 52	4600	1840	No	20.0	6	219
## 53	5000	2370	Yes	13.2	4	164
## 54	5500	2220	Yes	14.5	5	172
## 55	5600	2505	Yes	15.5	5	184
## 56	5000	2240	No	19.6	7	190
## 57	6500	2325	Yes	20.0	2	169
## 58	5100	2425	Yes	14.5	5	175
## 59	5500	2220	No	18.5	5	187
## 60	5750	2475	Yes	11.1	4	166
## 61	3800	1730	No	18.0	5	199
## 62	6000	2505	Yes	13.2	5	172
## 63	6000	2210	No	19.0	5	190
## 64	6000	2435	Yes	13.2	5	170
## 65	5600	2130	Yes	15.9	5	181
## 66	4800	2065	No	20.0	7	190
## 67	5200	2045	No	18.5	5	188
## 68	6000	2380	No	15.2	5	188
## 69	5200	2565	No	16.5	5	190
## 70	4800	1690	No	20.0	7	194
## 71	4800	1570	No	18.0	6	201
## 72	5000	2360	Yes	15.9	4	173
## 73	5600	3130	Yes	13.2	4	177
## 74	5200	2665	Yes	15.2	5	181
## 75	4600	1805	Yes	15.5	4	196
## 76	5000	1890	Yes	16.5	5	195
## 77	4800	1565	No	18.0	6	177
## 78	6000	2910	Yes	18.0	5	184
## 79	5000	2145	Yes	12.8	5	176
## 80	5600	2875	Yes	9.2	4	146
## 81	5200	3375	Yes	15.9	5	175
## 82	5600	2330	Yes	15.9	5	179
## 83	6000	3360	Yes	10.6	4	161
## 84	5200	3505	Yes	11.9	5	162
## 85	5400	2405	Yes	15.9	4	174
## 86	5400	2340	Yes	18.5	5	188
## 87	5000	2515	Yes	19.8	7	187
## 88	5500	2550	Yes	12.4	4	163
## 89	4500	2915	Yes	21.1	7	187
## 90	5800	2685	Yes	18.5	5	180
## 91	5800	2385	Yes	18.5	4	159
## 92	5400	2215	Yes	15.8	5	190

##	93	6200	2310	Yes	19.3	5	184	
##		Wheelbase	Width	Turn.circle	Rear.seat.room	Luggage.room	Weight	Origin
##	1	102	68	37	26.5	11	2705	non-USA
##	2	115	71	38	30.0	15	3560	non-USA
##	3	102	67	37	28.0	14	3375	non-USA
##	4	106	70	37	31.0	17	3405	non-USA
##	5	109	69	39	27.0	13	3640	non-USA
##	6	105	69	41	28.0	16	2880	USA
##	7	111	74	42	30.5	17	3470	USA
##	8	116	78	45	30.5	21	4105	USA
##	9	108	73	41	26.5	14	3495	USA
##	10	114	73	43	35.0	18	3620	USA
##	11	111	74	44	31.0	14	3935	USA
##	12	101	66	38	25.0	13	2490	USA
##	13	103	68	39	26.0	14	2785	USA
##	14	101	74	43	25.0	13	3240	USA
##	15	108	71	40	28.5	16	3195	USA
##	16	110	74	44	30.5	NA	3715	USA
##	17	111	78	42	33.5	NA	4025	USA
##	18	116	77	42	29.5	20	3910	USA
##	19	96	74	43	NA	NA	3380	USA
##	20	113	74	40	31.0	15	3515	USA
##	21	104	68	41	30.5	14	3085	USA
##	22	110	69	44	36.0	17	3570	USA
##	23	98	66	32	26.5	11	2270	USA
##	24	97	67	38	26.5	13	2670	USA
##	25	104	68	39	30.5	14	2970	USA
##	26	112	72	42	26.5	NA	3705	USA
##	27	105	69	42	30.5	16	3080	USA
##	28	97	72	40	20.0	11	3805	USA
##	29	98	66	36	26.5	11	2295	USA
##	30	113	74	40	30.0	15	3490	USA
##	31	90	63	33	26.0	12	1845	USA
##	32	98	67	36	28.0	12	2530	USA
##	33	100	68	39	27.5	13	2690	USA
##	34	101	68	40	24.0	12	2850	USA
##	35	103	70	38	23.0	18	2710	USA
##	36	119	72	45	30.0	NA	3735	USA
##	37	106	71	40	27.5	18	3325	USA
##	38	114	78	43	30.0	21	3950	USA
##	39	93	63	34	27.5	10	1695	non-USA
##	40	97	67	37	24.5	11	2475	non-USA
##	41	100	70	39	23.5	8	2865	non-USA
##	42	103	67	36	28.0	12	2350	non-USA
##	43	107	67	41	28.0	14	3040	non-USA
##	44	94	63	35	26.0	11	2345	non-USA
##	45	98	66	36	28.0	12	2620	non-USA
##	46	94	64	34	23.5	9	2285	non-USA
##	47	104	69	41	31.0	14	2885	non-USA
##	48	113	72	42	29.0	15	4000	non-USA

## 49	103	70	40	27.5	14	3510	non-USA
## 50	106	71	39	25.0	9	3515	non-USA
## 51	109	73	42	30.0	19	3695	USA
## 52	117	77	45	31.5	22	4055	USA
## 53	97	66	34	27.0	16	2325	non-USA
## 54	98	66	36	26.5	13	2440	non-USA
## 55	103	69	40	29.5	14	2970	non-USA
## 56	110	72	39	27.5	NA	3735	non-USA
## 57	96	69	37	NA	NA	2895	non-USA
## 58	105	67	34	26.0	12	2920	non-USA
## 59	110	69	37	27.0	15	3525	non-USA
## 60	95	65	36	19.0	6	2450	USA
## 61	113	73	38	28.0	15	3610	USA
## 62	98	67	36	26.0	11	2295	non-USA
## 63	107	70	43	27.5	14	3730	non-USA
## 64	96	66	33	26.0	12	2545	non-USA
## 65	103	67	40	28.5	14	3050	non-USA
## 66	112	74	41	27.0	NA	4100	non-USA
## 67	104	69	41	28.5	14	3200	non-USA
## 68	103	67	39	28.0	14	2910	USA
## 69	105	70	42	28.0	16	2890	USA
## 70	110	74	44	30.5	NA	3715	USA
## 71	111	74	42	31.5	17	3470	USA
## 72	97	67	39	24.5	8	2640	USA
## 73	99	66	35	25.5	17	2350	USA
## 74	101	66	39	25.0	13	2575	USA
## 75	101	75	43	25.0	13	3240	USA
## 76	108	72	41	28.5	16	3450	USA
## 77	111	74	43	30.5	18	3495	USA
## 78	99	67	37	26.5	14	2775	non-USA
## 79	102	68	40	26.5	12	2495	USA
## 80	90	60	32	23.5	10	2045	non-USA
## 81	97	65	35	27.5	15	2490	non-USA
## 82	102	67	37	27.0	14	3085	non-USA
## 83	93	63	34	27.5	10	1965	non-USA
## 84	94	65	36	24.0	11	2055	non-USA
## 85	99	69	39	23.0	13	2950	non-USA
## 86	103	70	38	28.5	15	3030	non-USA
## 87	113	71	41	35.0	NA	3785	non-USA
## 88	93	63	34	26.0	10	2240	non-USA
## 89	115	72	38	34.0	NA	3960	non-USA
## 90	103	67	35	31.5	14	2985	non-USA
## 91	97	66	36	26.0	15	2810	non-USA
## 92	104	67	37	29.5	14	2985	non-USA
## 93	105	69	38	30.0	15	3245	non-USA
##			Make				
## 1			Acura Integra				
## 2			Acura Legend				
## 3			Audi 90				
## 4			Audi 100				

## 5	BMW 535i
## 6	Buick Century
## 7	Buick LeSabre
## 8	Buick Roadmaster
## 9	Buick Riviera
## 10	Cadillac DeVille
## 11	Cadillac Seville
## 12	Chevrolet Cavalier
## 13	Chevrolet Corsica
## 14	Chevrolet Camaro
## 15	Chevrolet Lumina
## 16	Chevrolet Lumina_APV
## 17	Chevrolet Astro
## 18	Chevrolet Caprice
## 19	Chevrolet Corvette
## 20	Chrylser Concorde
## 21	Chrysler LeBaron
## 22	Chrysler Imperial
## 23	Dodge Colt
## 24	Dodge Shadow
## 25	Dodge Spirit
## 26	Dodge Caravan
## 27	Dodge Dynasty
## 28	Dodge Stealth
## 29	Eagle Summit
## 30	Eagle Vision
## 31	Ford Festiva
## 32	Ford Escort
## 33	Ford Tempo
## 34	Ford Mustang
## 35	Ford Probe
## 36	Ford Aerostar
## 37	Ford Taurus
## 38	Ford Crown_Victoria
## 39	Geo Metro
## 40	Geo Storm
## 41	Honda Prelude
## 42	Honda Civic
## 43	Honda Accord
## 44	Hyundai Excel
## 45	Hyundai Elantra
## 46	Hyundai Scoupe
## 47	Hyundai Sonata
## 48	Infiniti Q45
## 49	Lexus ES300
## 50	Lexus SC300
## 51	Lincoln Continental
## 52	Lincoln Town_Car
## 53	Mazda 323
## 54	Mazda Protege

```

## 55          Mazda 626
## 56          Mazda MPV
## 57          Mazda RX-7
## 58    Mercedes-Benz 190E
## 59    Mercedes-Benz 300E
## 60          Mercury Capri
## 61          Mercury Cougar
## 62          Mitsubishi Mirage
## 63    Mitsubishi Diamante
## 64          Nissan Sentra
## 65          Nissan Altima
## 66          Nissan Quest
## 67          Nissan Maxima
## 68    Oldsmobile Achieva
## 69 Oldsmobile Cutlass_Ciera
## 70    Oldsmobile Silhouette
## 71 Oldsmobile Eighty-Eight
## 72          Plymouth Laser
## 73          Pontiac LeMans
## 74          Pontiac Sunbird
## 75          Pontiac Firebird
## 76    Pontiac Grand_Prix
## 77    Pontiac Bonneville
## 78          Saab 900
## 79          Saturn SL
## 80          Subaru Justy
## 81          Subaru Loyale
## 82          Subaru Legacy
## 83          Suzuki Swift
## 84          Toyota Tercel
## 85          Toyota Celica
## 86          Toyota Camry
## 87          Toyota Previa
## 88          Volkswagen Fox
## 89    Volkswagen Eurovan
## 90    Volkswagen Passat
## 91    Volkswagen Corrado
## 92          Volvo 240
## 93          Volvo 850

```

Create a prediction method g that takes in a vector x_star and the dataset \mathbb{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

#From Office Hours

```

g = function(x_star, X, y){
  b = my_ols(X, y)$b
  x_star = c(1,x_star)
  x_star %*% b
}

```

```

}

X = model.matrix(~Type,cars)[, 2:6]
head(X)

##   TypeLarge TypeMidsize TypeSmall TypeSporty TypeVan
## 1         0           0         1           0         0
## 2         0           1         0           0         0
## 3         0           0         0           0         0
## 4         0           1         0           0         0
## 5         0           1         0           0         0
## 6         0           1         0           0         0

g(X[1,], X, cars$Price)

##           [,1]
## [1,] 10.16667

t(c(1,X[1,])) %*% my_ols(X, cars$Price)$b

##           [,1]
## [1,] 10.16667

X[1,]

##   TypeLarge TypeMidsize TypeSmall TypeSporty TypeVan
##         0           0         1           0         0

predict(mod, cars[1,])

##           1
## 10.16667

```