Lab 3

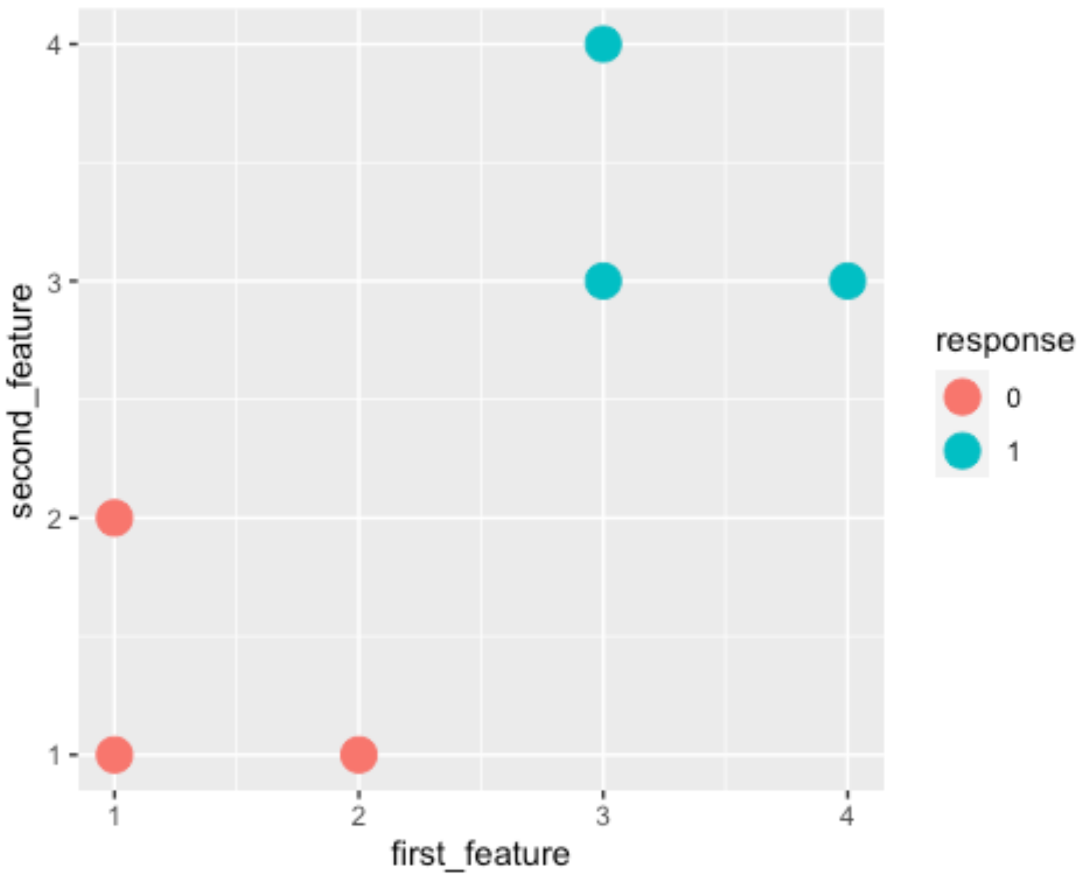
Sara Jedwab

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
```



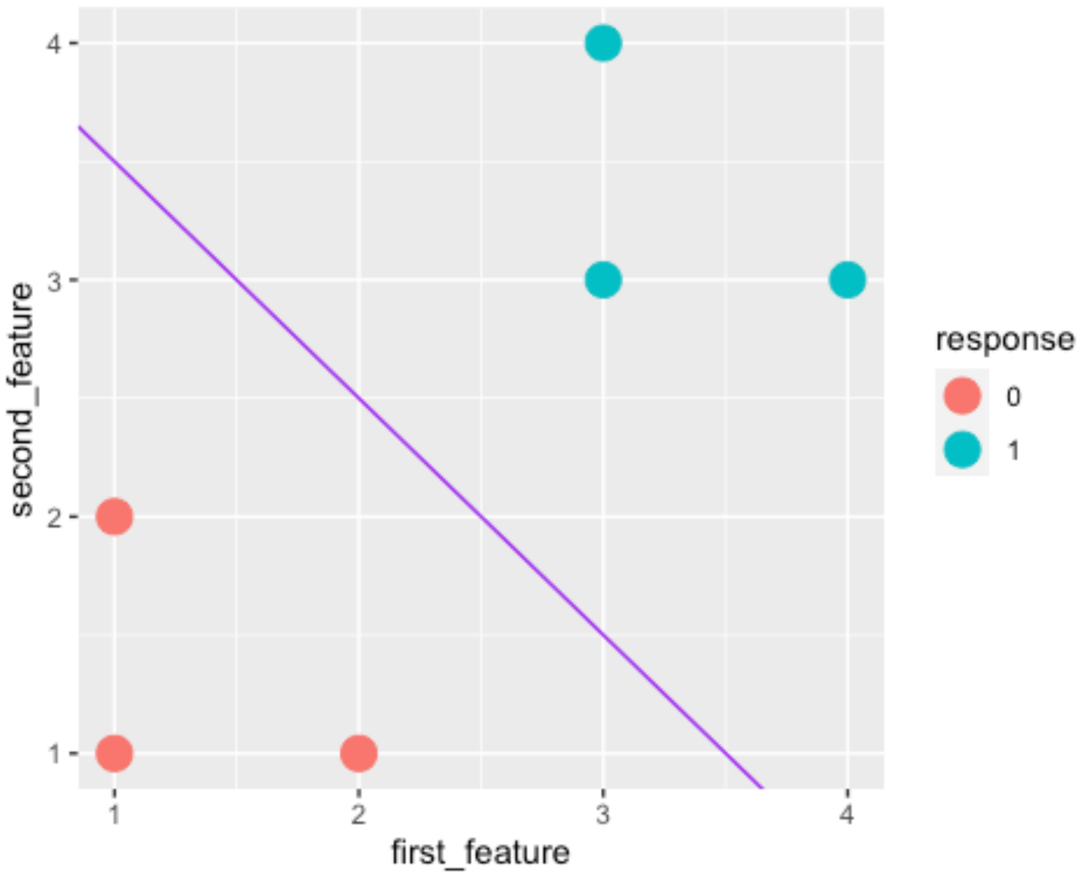
Use the `e1071` package to fit an SVM model to the simple data. Use a formula to create the model, pass in the data frame, set kernel to be `linear` for the linear SVM and don't scale the covariates. Call the model object `svm_model`. Otherwise the remaining code won't work.

```r
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:

```r
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
```



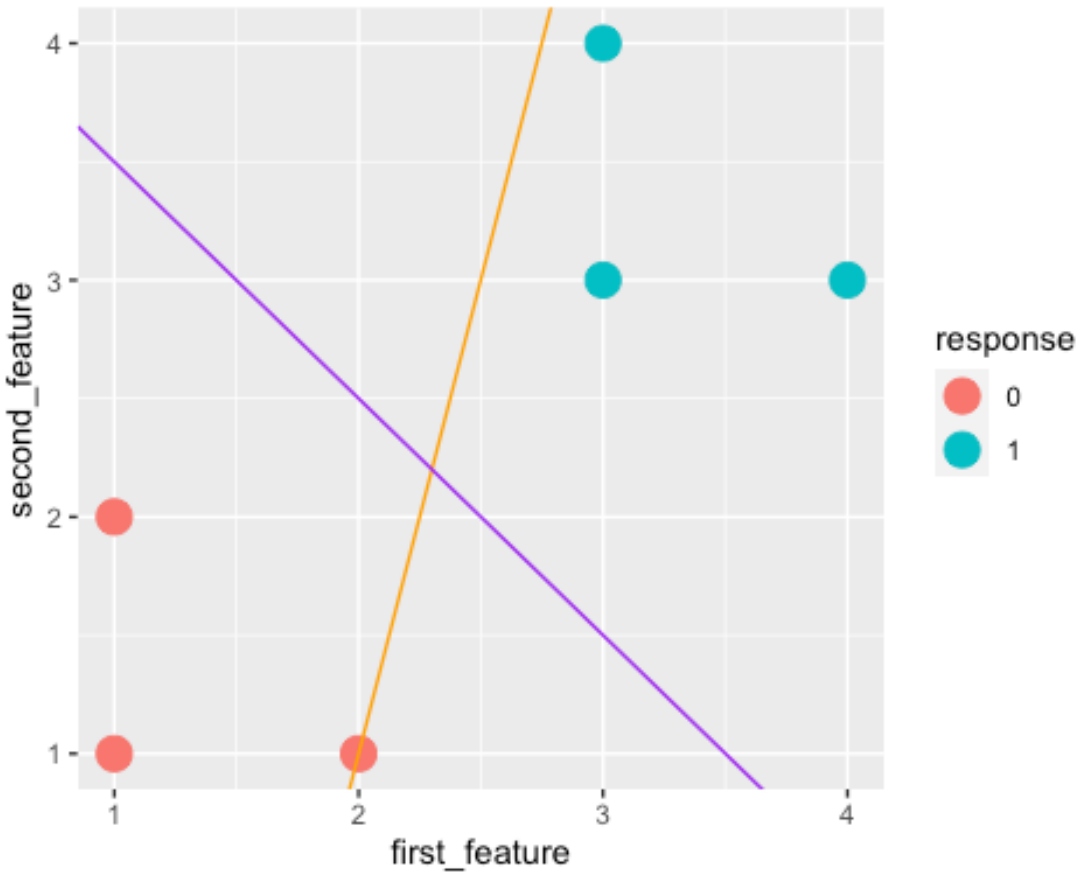
Source the `perceptron_learning_algorithm` function from lab 2. Then run the following to fit the perceptron and plot its line in orange with the SVM's line:

```
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)

for (iter in 1 : MAX_ITER){
  for (i in 1 : nrow(Xinput)) {
    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, it is much better! It's much more satisfying because it evenly separately the data down the center in a cleaner way.

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
```

```r
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){
  SHE = 0
  initialize a w vector
  for x in MAX_ITER{
    for i in nrow(Xinput){
        SHE += max{0, (1/2) - (y_binary[i]-1/2)(w*Xinput[i]-b)}
    }
  }
  argmin{(SHE/n) + lambda(distance of w)^2}
  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).

```r
#' 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:

```
y_binary = as.matrix(Xy_simple$response)
X_simple_feature_matrix = as.matrix(Xy_simple)
X_simple_feature_matrix$response = NULL

svm_model_weights = linear_svm_learning_algorithm(X_simple_feature_matrix,
y_binary)
my_svm_line = geom_abline(
    intercept = svm_model_weights[1] / svm_model_weights[3],#NOTE: negative
sign removed from intercept argument here
    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:

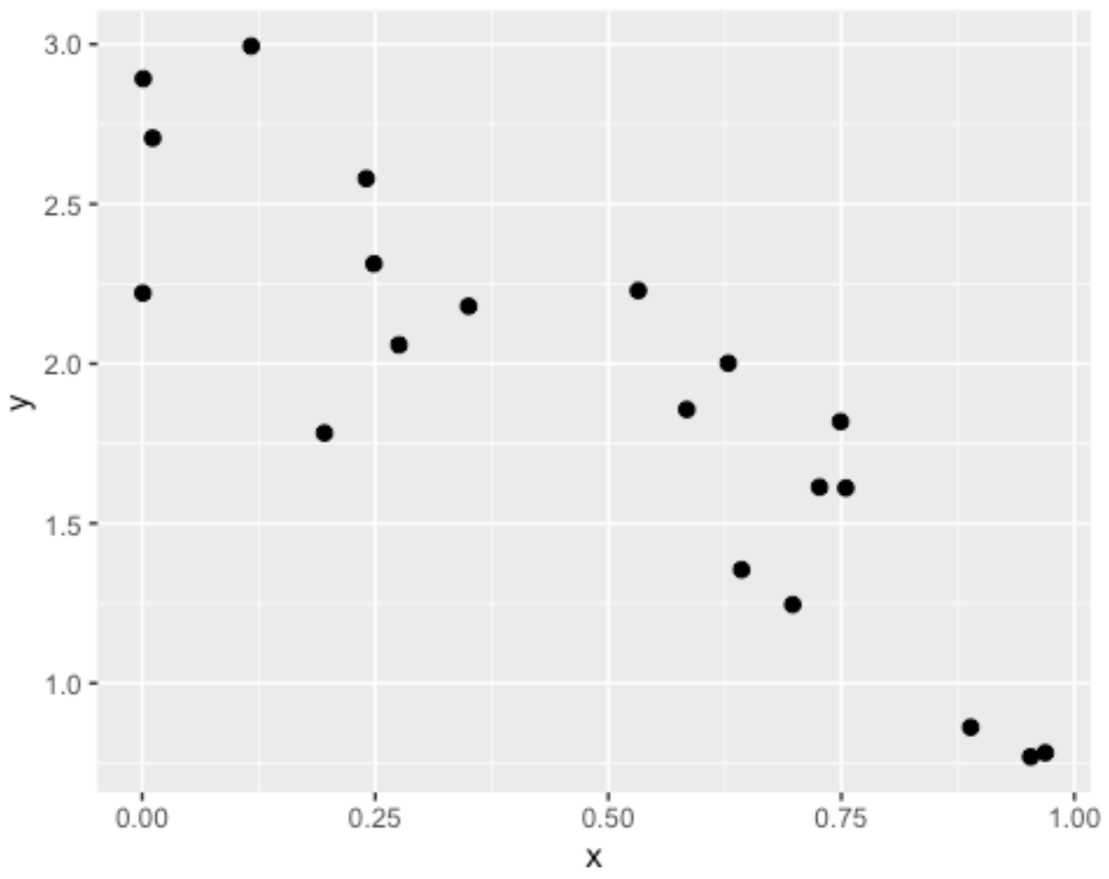
```
n = 20
x = runif(n)
beta_0 = 3
beta_1 = -2
```

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

```
h_star_x = beta_0 + beta_1*x
epsilon = rnorm(n, mean = 0, sd = 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 $beta_0$ and $beta_1$?

Yes it does because beta_0 is the y-intercept and equal to 3, and beta_1 is the slope and equal to -2, and the graph above more or less surrounds a line with those values (y=3-2x). (It's not exactly on the line because of the epsion values.)

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 `Rsq` (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)
  Rsq = 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, Rsq = Rsq)
  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$Rsq, summary(lm_mod)$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$e)

## [1] -5.551115e-18

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)
X
```

```
##                    x
##  [1,] 1 0.9687357054
##  [2,] 1 0.5841753292
##  [3,] 1 0.2404282063
##  [4,] 1 0.0007064049
##  [5,] 1 0.5321656407
##  [6,] 1 0.6429440738
##  [7,] 1 0.9530694261
##  [8,] 1 0.0112380881
##  [9,] 1 0.7265754181
## [10,] 1 0.6287019639
## [11,] 1 0.1955606039
## [12,] 1 0.0011337111
## [13,] 1 0.7547773405
## [14,] 1 0.2755071190
## [15,] 1 0.2485597406
## [16,] 1 0.6977965727
## [17,] 1 0.7490801935
## [18,] 1 0.1169883334
## [19,] 1 0.3501116438
## [20,] 1 0.8886933182
```

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 0.9687357054
## 2            1 0.5841753292
## 3            1 0.2404282063
## 4            1 0.0007064049
## 5            1 0.5321656407
## 6            1 0.6429440738
## 7            1 0.9530694261
## 8            1 0.0112380881
## 9            1 0.7265754181
## 10           1 0.6287019639
## 11           1 0.1955606039
## 12           1 0.0011337111
## 13           1 0.7547773405
## 14           1 0.2755071190
## 15           1 0.2485597406
## 16           1 0.6977965727
## 17           1 0.7490801935
## 18           1 0.1169883334
## 19           1 0.3501116438
## 20           1 0.8886933182
## 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){
  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_b = 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_b[i] = sum((beta - b)^2)
}
log(errors_in_b,10)

## [1] -2.179163 -2.605445 -3.461640 -3.447384 -5.186187 -5.483148 -6.744943
## [8] -8.708581
```

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

```r
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	68.31	1.79	64.00	67.55	68.55	69.55	73.0	
child	0	1	68.09	2.52	61.77	66.22	68.22	70.22	73.7	

p is 1 = height measurements, and n is 928 the number of observations
This data frame depicts the data of 928 children and their parents, comparing their heights. The mean height for the parents was 68.3 in and 68.1 in for the children, which makes sense since there's no reason why the average height should change in one generation.

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

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

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

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

## [1] 2.309372
```

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_0

## (Intercept)
##    23.94153

b_1 = coef(mod)[2]
b_1

##    parent
## 0.6462906

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: Obviously, you'd never be predicting for x=0 in this case,since that would mean that the parent has no height (0 inches) and therefore wouldn't exist, but this value gives the line that we care about because it needs a y intercept. b_1: For every in increase in the parents' height, the child's height increase by the value of b_1 = 0.6462906 inches. RMSE: +-2.23(2) = +-4.5 inch from the mean => 9 inches range 95% time R^2: Only 21% variance explained which is pretty low, but truly interpreting this will depend on what your standards are for your model.

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

It's a pretty good model considering its only based on one feature (height), however it still has a pretty big range of 9 inches (95% of the time) which means the prediction can be almost a foot off within that confidence interval.

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.

This is definitely a reasonable assumption that children, whose DNA are a combination of their parents', would inherit their "height genes" and have around the same 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?

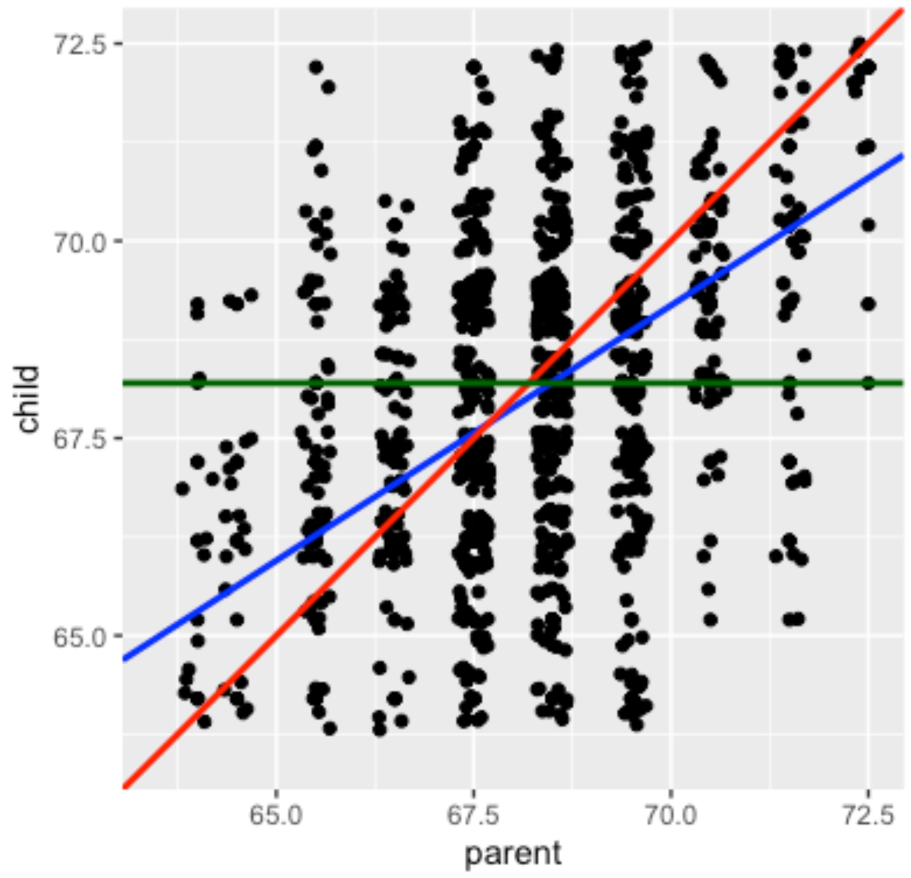
In that case, beta_0 would be 0, and beta_1 would be 1.

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 84 rows containing missing values (geom_point).
```



Fill in the following sentence:

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"?

Galton called it "Regression towards mediocrity in hereditary stature" because he observed that, compared to their parents, the children shifted/regressed to the mean height. When parents were taller than average, then the child was shorter and therefore closer to the mean, and vice versa.

Why should this effect be real?

This effect should be real because over time the mean height of the human population stays the same due to biological restrictions, so whenever there is a deviation from the mean, it makes sense that there will be some type of compensation of sorts with their children.

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.

Everyone calls y continuous "regression" when modeling continuous dependent variables, as a result of Galton's innovative way of modeling and the results that he observed. It's a bit of misnomer because preforming a regression analysis won't always result in a regression to the mean; it's just a way of measuring the mean change in a dependent variable given a one-unit change in each independent variable. A better name for building predictive models with y continuous would be the linear least squares model since that's actually what we're measuring.

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.

```
y = 1:5
x = y^60
Xy = data.frame(x = x, y = y)
model = lm(y ~ x)
summary(model)$r.squared

## [1] 0.5000011

summary(model)$sigma

## [1] 1.290993
```

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 = 1:200
y = x^67
Xy = data.frame(x = x, y = y)
model = lm(y ~ x)
summary(model)$r.squared

## [1] 0
```

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 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 `Rsq` (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 ROxygen documentation here.

```
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, p=p)
```

```
  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)
x = my_ols(as.numeric(data.matrix(data.frame((cars$Type)))), cars$Price)
x

## $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
##
## $p
## [1] 2
##
## attr(,"class")
## [1] "my_ols_obj"
```

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

```
g = function(x_star, X, y){
  b = my_ols(X,y)$b
  x_star = c(1,x_star)
  x_star %*% b
}
X = model.matrix( ~Type,cars)[, 2:6]
g(X[1,], X, cars$Price)
```

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

```
predict(mod, cars[1,])
```

```
##        1
## 10.16667
```