HW02p

Greg Maghakian

March 6, 2018

Welcome to HW02p where the "p" stands for "practice" meaning you will use R to solve practical problems. This homework is due 11:59 PM Tuesday 3/6/18.

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Some of this will be a pure programming assignment. Sometimes you will have to also write English.

The tools for the solutions to these problems can be found in the class practice lectures. I want you to use the methods I taught you, not for you to google and come up with whatever works. You won't learn that way.

To "hand in" the homework, you should compile or publish this file into a PDF that includes output of your code. To do so, use the knit menu in RStudio. You will need LaTeX installed on your computer. See the email announcement I sent out about this. Once it's done, push the PDF file to your github class repository by the deadline. You can choose to make this respository private.

For this homework, you will need the `testthat` libray.

```
pacman::p_load(testthat)
```

1. Source the simple dataset from lecture 6p:

```
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
)
X_simple_feature_matrix = as.matrix(Xy_simple[, 2 : 3])
y_binary = as.numeric(Xy_simple$response == 1)
```

Try your best to write a general perceptron learning algorithm to the following `Roxygen` spec. For inspiration, see the one I wrote in lecture 6.

```
#' This function implements the "perceptron learning algorithm" of Frank Rosenblatt (1957).
#'
#' @param Xinput      The training data features as an n x (p + 1) matrix where the first column is all
#' @param y_binary    The training data responses as a vector of length n consisting of only 0's and 1'
#' @param MAX_ITER    The maximum number of iterations the perceptron algorithm performs. Defaults to 1
#' @param w           A vector of length p + 1 specifying the parameter (weight) starting point. Defaul
#'                    \code{NULL} which means the function employs random standard uniform values.
#' @return            The computed final parameter (weight) as a vector of length p + 1
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w = NULL){
  if (is.null(w)){
    w = runif(ncol(Xinput)) #intialize a p+1-dim vector with random values
  }
  for (iter in 1 : MAX_ITER){
    for (i in 1 : nrow(Xinput)){
      x_i = Xinput[i, ]
      yhat_i = ifelse(x_i %*% w > 0, 1, 0)
      w = w + as.numeric(y_binary[i] - yhat_i) * x_i
    }
  }
```

```
  w
}
```

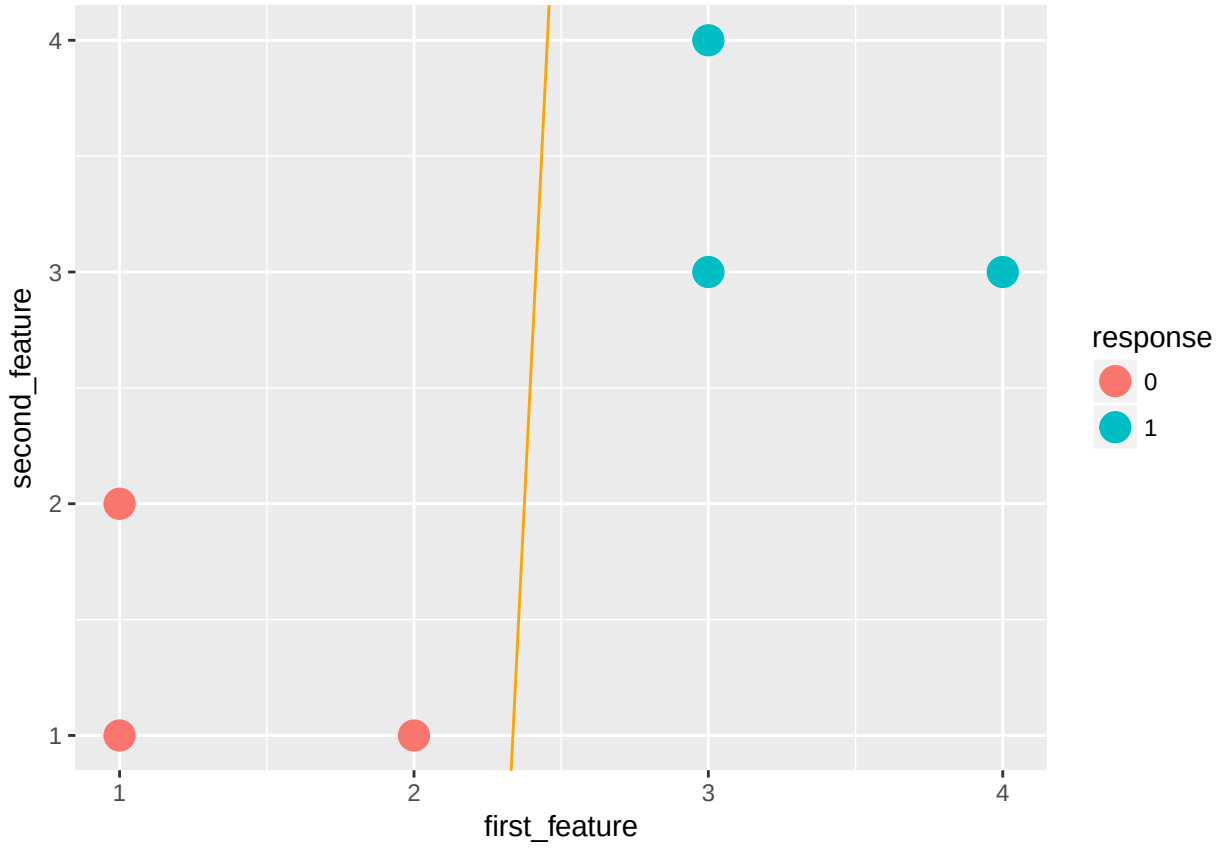
Run the code on the simple dataset above via:

```
w_vec_simple_per = perceptron_learning_algorithm(
  cbind(1, Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1))
w_vec_simple_per
```

```
## [1] -2.97216406  1.29397397 -0.05051048
```

Use the ggplot code to plot the data and the perceptron's g function.

```
pacman::p_load(ggplot2)
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature, color = response)) +
  geom_point(size = 5)
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
```



Why is this line of separation not "satisfying" to you?

It could be separated better maybe cut across in the diagonal more efficiently.

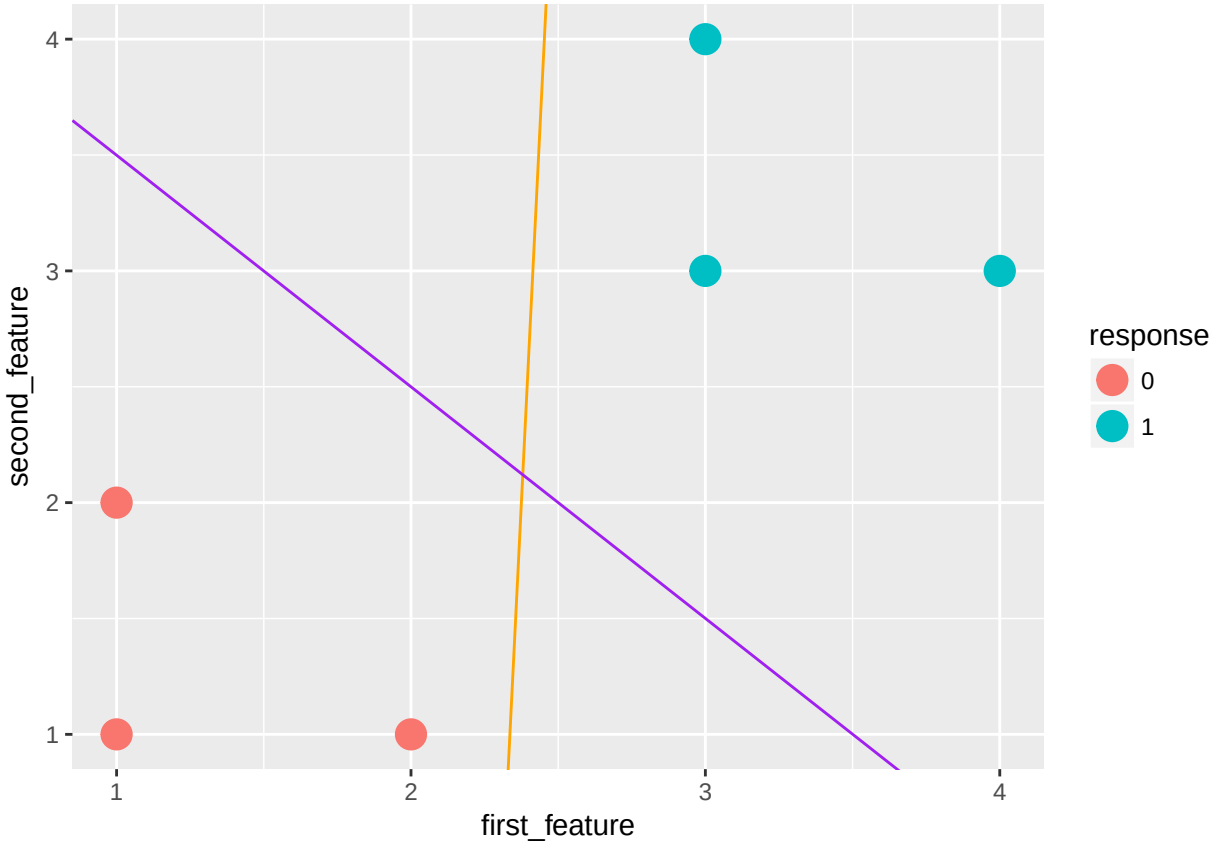
2. Use the `e1071` package to fit an SVM model to `y_binary` using the predictors found in `X_simple_feature_matrix`. Do not specify lambda

```
pacman::p_load(e1071)


n = nrow(X_simple_feature_matrix)
svm_model = svm(X_simple_feature_matrix, Xy_simple$response, 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) %*% X_simple_feature_matrix[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_perceptron_line + simple_svm_line
```



Is this SVM line a better fit than the perceptron?

It seems to be a better fit based on how the svm separated it more naturally but again, the data we have is linerally separable. The svm finds the best fit based on lambda with wedge size and least number of errors–average hinge-loss. However, since it's linearlly separable, we get the greatest distance between the two closest different features.

TO-DO

3. Now write your own implementation of the linear support vector machine algorithm respecting the following spec making use of the nelder mead `optim` function from lecture 5p. It turns out you do not

need to load the package `neldermead` to use this function. You can feel free to define a function within this function if you wish.

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.

```
#' This function implements the hinge-loss linear support vector machine algorithm of Vladimir Vapnik (
#'
#' @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'
#' @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
#'                    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 = 1){
  #TO-DO
 }
```

How to do the SVM:

1)

We need to create a loop that will run through all observations and find the two points that have the minimum distance between them i.e one 0 and one 1 point that is the minimum distance. we divide that by 2 and that creates our w vector. we then take the norm of that. Now that we have our w vector, we will plug that into our cost function which basically is something similar to taking the sum of the average errors.

wvecˆ2 and lambda+ a for loop to find the hinge loss and divide that by 2.

we then initialize our starting point at a randomized point, or something like the origin, and pass it through the optim function which will minimize the hingeloss function+lambda*wˆ2

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(X_simple_feature_matrix, y_binary) my_svm_line = geom_abline(intercept = -svm_model_weights[1] / svm_model_weights[3], slope = -svm_model_weights[2] / svm_model_weights[3], color = "brown") simple_viz_obj + simple_svm_line + my_svm_line

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

4. Write a $k = 1$ nearest neighbor algorithm using the Euclidean distance function. Respect the spec below:

```
#' This function implements the nearest neighbor algorithm.
#'
#' @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'
#' @param Xtest       The test data that the algorithm will predict on as a n* x p matrix.
#' @return            The predictions as a n* length vector.



nn_algorithm_predict = function(Xinput, y_binary, Xtest){
  vec=rep(NA,nrow(Xtest))
  count=1
  best_sqd_distance = Inf
  i_star = NA
```

```
  p=ncol(Xinput)

    for (i in 1 : nrow(Xtest) ){

      best_sqd_distance=Inf

      for(index in 1:nrow(Xinput)){
         dsqd = sum((Xinput[index,] - Xtest[i,])^2)
         if (dsqd < best_sqd_distance){
        best_sqd_distance = dsqd
        i_star = index
         vec[count]=y_binary[i_star]
         }



      }

    count=count+1
    }



  vec
}
```

Write a few tests to ensure it actually works:

```
t=nn_algorithm_predict(X_simple_feature_matrix, y_binary , X_simple_feature_matrix)

expect_equal(t,y_binary)


test=matrix(c(rep(1,12)),nrow=6,ncol=2)
y_binary=c(1,2,3,4,5,6)
q=nn_algorithm_predict(X_simple_feature_matrix, y_binary , test)


expect_equal(q,rep(1,nrow(test)))
```

For extra credit, add an argument `k` to the `nn_algorithm_predict` function and update the implementation so it performs KNN. In the case of a tie, choose $\hat{y}$ randomly. Set the default `k` to be the square root of the size of $\mathcal{D}$ which is an empirical rule-of-thumb popularized by the "Pattern Classification" book by Duda, Hart and Stork (2007). Also, alter the documentation in the appropriate places.

```
#not required TO-DO --- only for extra credit
```

For extra credit, in addition to the argument `k`, add an argument `d` representing any legal distance function to the `nn_algorithm_predict` function. Update the implementation so it performs KNN using that distance function. Set the default function to be the Euclidean distance in the original function. Also, alter the documentation in the appropriate places.

```
#not required TO-DO --- only for extra credit
```

5. We 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
y = beta_0 + beta_1 * x + rnorm(n, mean = 0, sd = 0.33)
```

Solve for the least squares line by computing b_0 and b_1 *without* using the functions `cor`, `cov`, `var`, `sd` but instead computing it from the x and y quantities manually. See the class notes.

```
y_bar=sum(y)/n
x_bar=sum(x)/n
sumXy=sum(x*y)
nyx=n*y_bar*x_bar
Xsqr=sum(x*x)
Nx=n*(x_bar^2)
b_1=(sumXy-nyx)/(Xsqr-Nx)
b_0=y_bar-b_1*x_bar
```

Verify your computations are correct using the `lm` function in R:

```
lm_mod = lm(y~x)
b_vec = coef(lm_mod)
expect_equal(b_0, as.numeric(b_vec[1]), tol = 1e-4)
expect_equal(b_1, as.numeric(b_vec[2]), tol = 1e-4)
```

6. 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 using the `data` command:

```
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`.

Summary: n which is the number of observations is 928 and p, the number of characteristics is two. For each observation we observe one column for the height of the mid-parent and another column for the height of the child. The data is measured in intervals of width 1 inch. The female children's height were weighted for sex differences. The average parent height was 68.31 inches and the average child height was 68.09 inches. The range is 64 inches to 73 inches for parents and 61.70 inches to 73.70 inches for children

```
summary(Galton)
```

```
##      parent          child      
##  Min.   :64.00   Min.   :61.70  
##  1st Qu.:67.50   1st Qu.:66.20  
##  Median :68.50   Median :68.20  
##  Mean   :68.31   Mean   :68.09  
##  3rd Qu.:69.50   3rd Qu.:70.20  
##  Max.   :73.00   Max.   :73.70
```

```
length(Galton)
```

```
## [1] 2
```

TO-DO

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
```

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. Use the correct units to report these quantities.

```
y=Galton$child
x=Galton$parent

linmod=lm(y~x)
names(linmod)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
r = cor(x, y)
s_x = sd(x)
s_y = sd(y)
ybar = mean(y)
xbar = mean(x)

b_1 = r * s_y / s_x
b_0 = ybar - b_1 * xbar
b_0
```

```
## [1] 23.94153
```

```
b_1
```

```
## [1] 0.6462906
```

```
y_hat=b_0+b_1*x
residual=y-y_hat
sse = sum(residual^2)
mse = sse / length(y)
rmse = sqrt(mse)
rmse
```

```
## [1] 2.236134
```

```
rsq = (var(y) - var(residual)) / var(y)
rsq
```

```
## [1] 0.2104629
```

```
Rsqr=(s_y^2-2.519301)/s_y^2
```

Interpret all four quantities: b_0, b_1, RMSE and R^2.

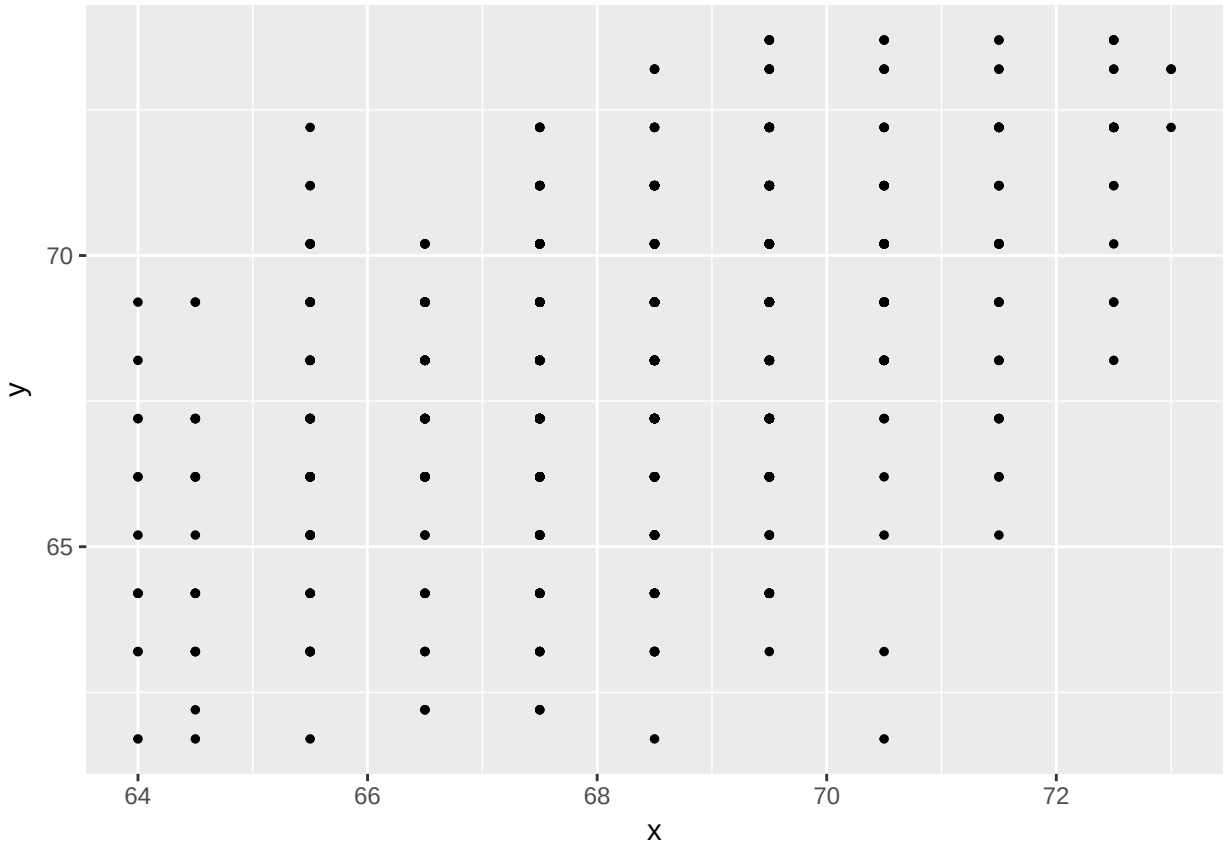
b_0 is the Intercept of the line reported in inches b_1 is the slope of the line child/parent RMSE is interpreted as "95% of the predictive errors lie between plus/minus 4.5 inches" R^2 has no units. It is the total variance

explained in the child's height from the parent's height. i.e. 60% of the variance in the child's height is explained by the parent's height.

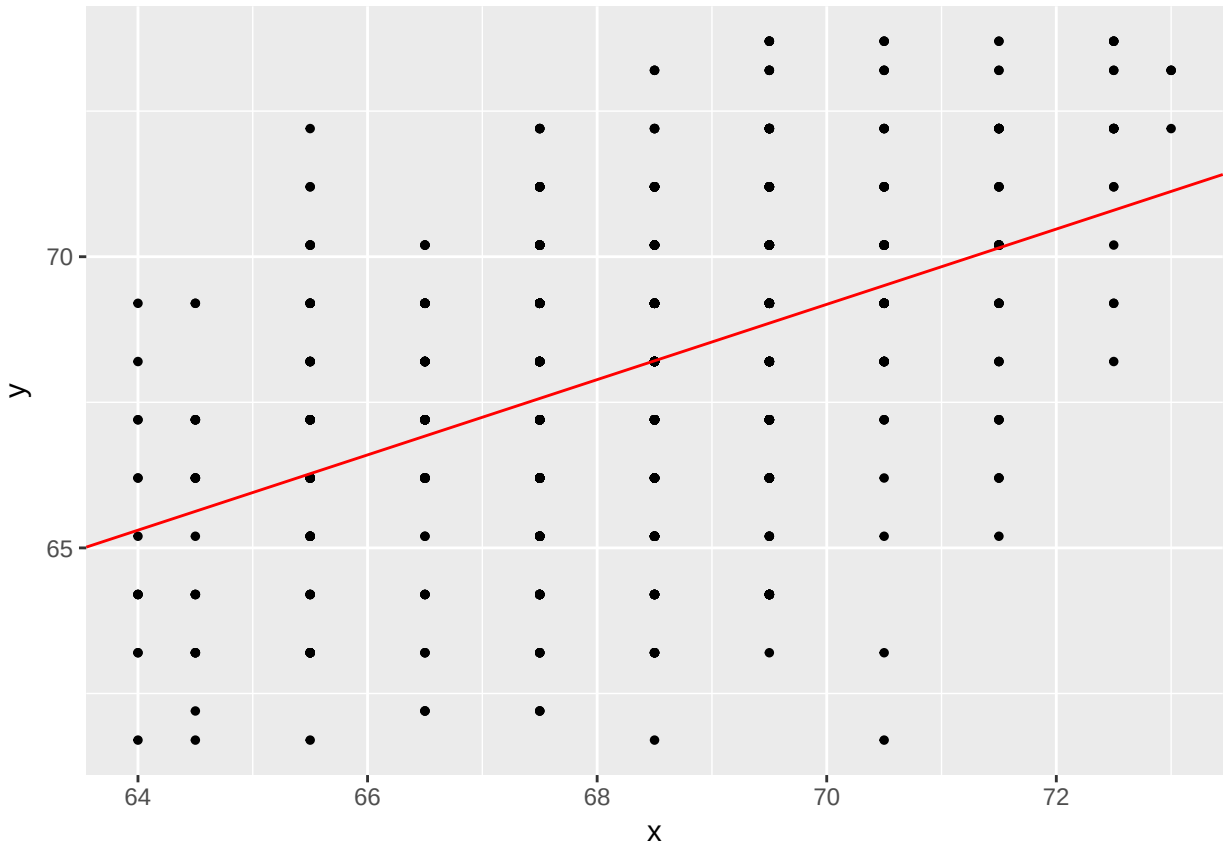
How good is this model? How well does it predict? Discuss. We can discuss how well this model performs using two metrics–R^2 and the RMSE. Based off of the R^2, our model explains about 60% of the variance in the child's heights by the parent's height. This isn't the best, as reducing the variance in the residuals would prove us better. Also, the RMSE I believe isn't good as 95% of our errors to the mean lie between plus/minus 4.5 inches meaning that we are off by a lot relative to our dataset. Meaning, our model isn't good at predicting height.

Now use the code from practice lecture 8 to plot the data and a best fit line using package `ggplot2`. Don't forget to load the library.

```
pacman::p_load(ggplot2)
simple_df = data.frame(x = x, y = y)
dataviz = ggplot(simple_df, aes(x, y)) +
  geom_point(size = 1)
dataviz
```




```
bestfit = geom_abline(intercept = b_0, slope = b_1, color = "red")
dataviz + bestfit
```



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

Yes this is reasonable to assume as genetics usually dictates that we inherit the same physical attributes like height from our 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?

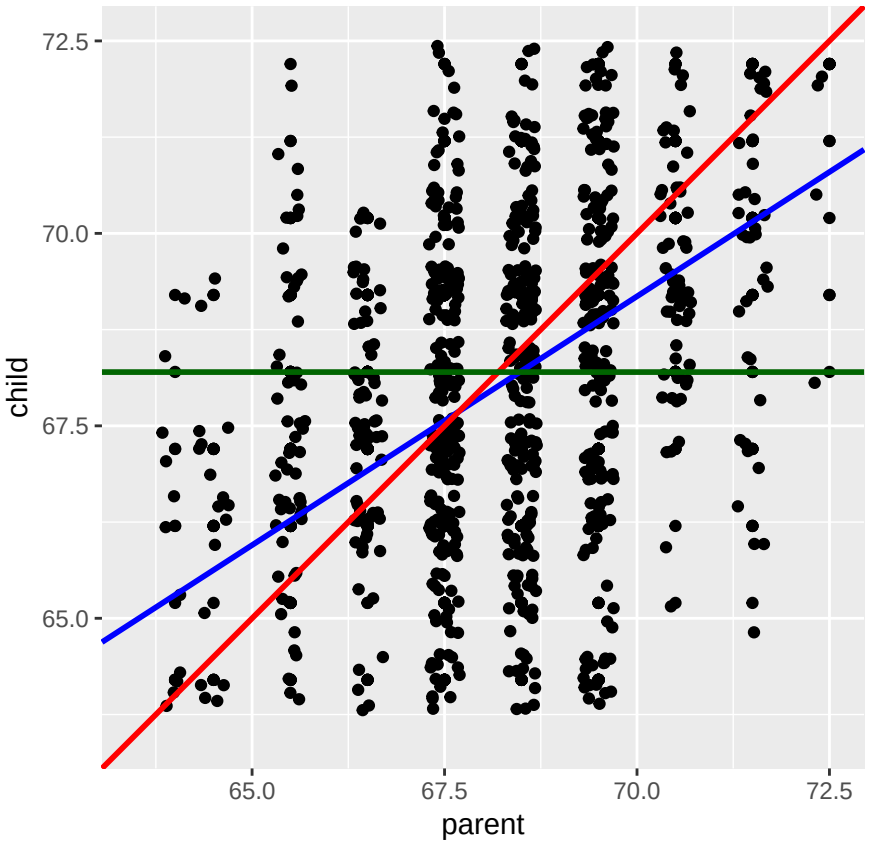
β_0 should be 0 and β_1 should be 1

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.

```
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 89 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"?

Most people will have a height that will center around the mean instead of being an outlier or a height that is drastically different from this.

Why should this effect be real?

We expect that on average, tall parents will have shorter children as genetics wise, the average is in the middle of the tallest and shortest parents. So, on average, children will either be shorter than their tall parents or taller than their short 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.

When modeling with y continuous, we call it a regression because we expect the data to clump or regress around the mean.

We can rename this something like "mean modeling" or "tendency to move towards the mean" as our modeling will move towards the mean or tend to centralize around it with y continuous.