---

title: "Lab 3"

author: "Asher Katz"

output: pdf_document

date: 03/06/2022

Regression via OLS with one feature

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

```{r}
set.seed(1984)

num = 20

x = runif(num)

beta_0 = 3

beta_1 = -2

print(x)
```

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.

```{r}
h_star_x = beta_0 + beta_1*x

epsilon = rnorm(20, 0, 0.33^2)

y = h_star_x + epsilon
```

Graph the data by running the following chunk:

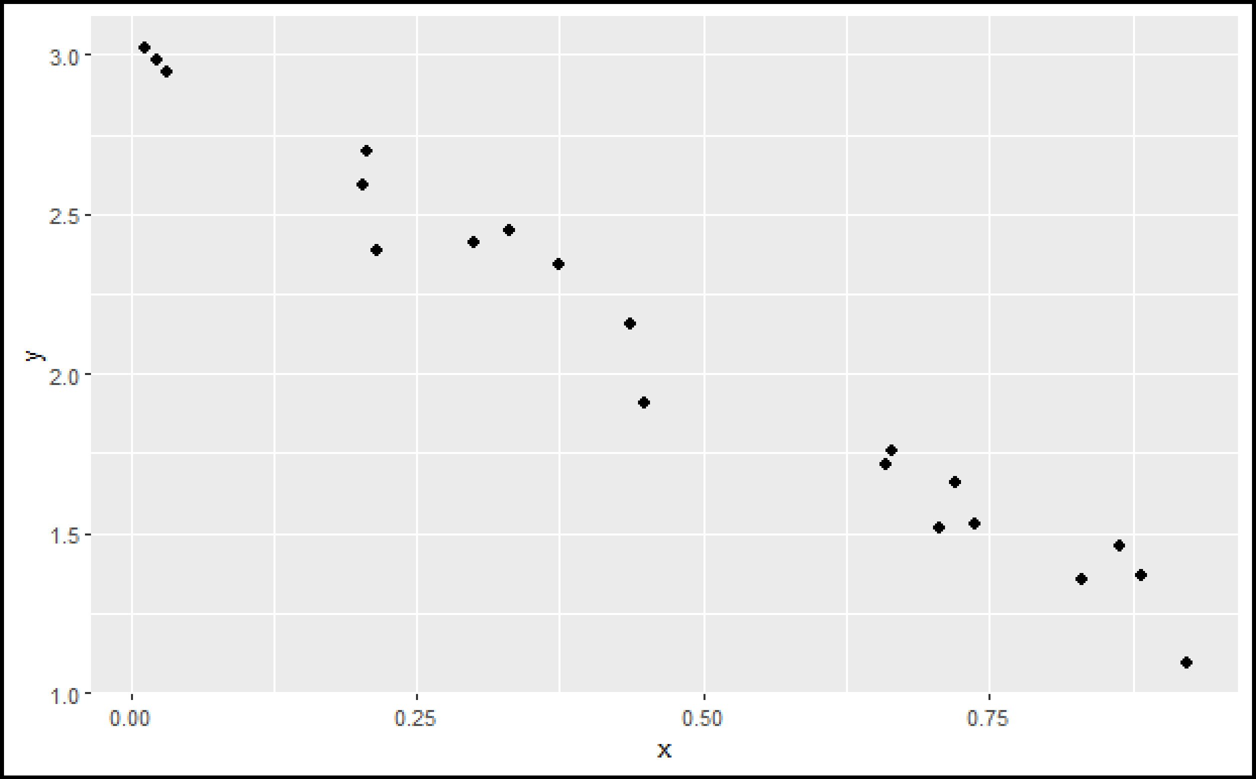
```{r}
pacman::p_load(ggplot2)

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, we can see a y-axis intercept of 3 and the line descends at a rate of -2/1

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 `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.

```
```{r}
my_simple_ols = function(x, y){
  if (is.numeric(x) == FALSE || is.numeric(y) == FALSE){
    return("ERROR: inputs are non-numeric ")
  }
  if (length(x)!= length(y)){
    return("ERROR: length of first arguement is not the same as length of second arguement")
  }
  ols_obj = list()
  n = length(x)
  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
  yhat = b_0 + b_1*x
  e = y - yhat
  SSE = sum(e^2)
  SST = (n-1)*(s_y^2)
  MSE = SSE/(n-2)
  RMSE = sqrt(MSE)
  Rsq = 1- (SSE)/SST
#
#   ols_obj$x = x
#   ols_obj$y =  y
  ols_obj$b_0 = b_0
  ols_obj$b_1 =  b_1
  ols_obj$yhat = yhat
  ols_obj$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
}
View(my_simple_ols(x,y))
```

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

```{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.

```{r}
expect_equal(sum(my_simple_ols_mod$e), sum(lm_mod$residuals), tol = 1e-4)

```

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

```{r}
x

X = as.matrix(cbind(1,x))

```

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

```{r}
modMat = model.matrix(~x)

```

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`.

```{r}
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.

```{r}
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.

```{r}
beta_0 = 3
beta_1 = -2
beta = c(beta_0, beta_1)
ns = (10^(1:7))# 7 increasingly big numbers
errors_in_beta = array(NA, length(ns))

for (i in 1 : length(ns)) {
  n = ns[i]
  x = runif(n) # 7 increasingly large datasets
  h_star_x = beta_0 + beta_1 * x # predictions to go with those datasets
  epsilon = rnorm(n, mean = 0, sd = 0.33) # n random errors within +-33%
  y = h_star_x + epsilon # error-corrected prediction set
  mod = my_simple_ols(x,y)
  b = c(mod$b_0, mod$b_1)

  errors_in_beta[i] = (b[1] - beta[1])^2 + (b[2] - beta[2])^2
  cat("when n = ", n,", (b_0 - beta_0)^2 + (b_1 - beta_1)^2 =", (b[1] - beta[1])^2 + (b[2] - beta[2])^2, "\n")

}
```

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`.

```{r}
pacman::p_load(HistData)
```

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

```{r}
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)

```

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

```{r}

y = c(Galton$parent, Galton$child)

avg_height = 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?

```{r}

SSE_0 = sum( (y - avg_height)^2)

RMSE_0 = sqrt(SSE_0/(2*nrow(Galton)-2))

RMSE_0

```

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.

```{r}

summary(Galton)

childHeight = lm(child ~ parent, Galton)

summary(childHeight)

b_0 = childHeight$coefficients[1]

b_1 = childHeight$coefficients[2]

RMSE = summary(childHeight)$sigma

Rsq = summary(childHeight)$r.squared
```

```
cat("b_0 = ", b_0,", b_1 = ", b_1, ", RMSE = ", RMSE,", Rsq = ", Rsq)
```

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

Galtons original data was collected in with a minimum increment of 0.1 inches so I will relate our predicted values using that increment as well.

* Simply, b_0 is the "child intercept", the predicted height of a child whose average parent height is negligible. It was valued at 23.9 inches tall.

* b_1 is the ratio of child height to average parent height. It was valued at 0.6 inches, which is to say that a child grows 0.6inches for each inch of their average parent height.

* The RMSE is valued at 2.2 inches, which is to say that a child's height is predicted to be within 2.2 inches of their average parent height.

* The Rsq is valued at 21.05%, which is to say that our predicted outcomes were 21.05% better than if we had predicted every value using the average outcome.

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

I think this is an okay model because it doesn't predict very well but it is a bit better than the null model. One might think that an RMSE of 2.2 inches is a good sign but one needs to realize that the range of average parent height is 9 inches, so 2.2 inches is a significant distance in the dataset.

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.

It is reasonable to assume, especially in the big picture, that parents and children will on average be the same height. This is because they come from the same gene pool so, even if they have a short kid, maybe that kid will have tall kids and it will seem weird but really it runs in the family. Even within what we think is not the best predictive model, our errors balanced out exceptionally well at -0.00000000002197464 inches, which tells us that despite our inability to predict children heights, we can say that they not getting taller as a whole.

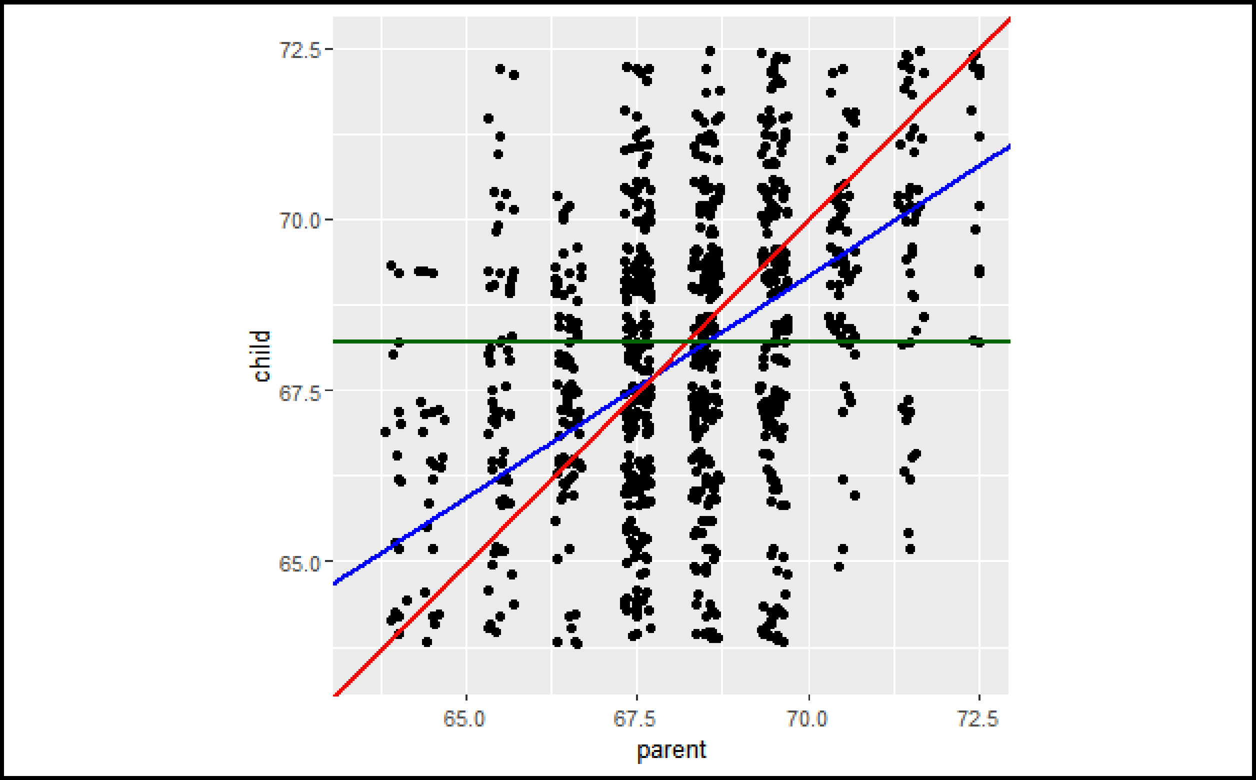
If they were to have the same height and any differences were just random noise with expectation 0, what would the values of beta_0 and beta_1 be?

they would both be zero in long run.

TO-DO

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.

```{r}
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)
```



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"?

because children heights tended towards the mean

Why should this effect be real?

Height abnormalities tend to not carry on and instead regress to the mean height. In terms of genetics, height is dependent on many genes and just becuase a child is tall doesn't mean he will carry all of that on and his generations may be shorter then him

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.

because height regressed towards the mean and "mean oriented" could be a better name.

You can now clear the workspace.

```{r}
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.

```{r}
x = c(1, 1, 2, 2, 3, 3, 4)
y = c(3, 2, 4 ,4, 6, 3, 5)
Xy = data.frame(x = x, y = y)
model = lm(y ~ x)
summary(model)$r.squared
summary(model)$sigma

```

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

```{r}
x = seq(-100,100)
y = x^2
Xy = data.frame(x = x, y = y)
Rsq0 = lm(y~x)
summary(Rsq0)$r.squared
```

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.

```{r}
```

```
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.

```{r}

my_ols = function(X, y){

n = nrow(X)

p = ncol(X)

X = cbind(1, X)

Xt = t(X)

XtX = Xt%*%X

XtXinv = solve(XtX)

b = XtXinv%*%Xt*y

y_hat = X%*%b

e = y - y_hat

df = p+1

SSE = sum(e^2)

SST = var(y)*(n-1)

MSE = SSE/(n-df)

RMSE = sqrt(MSE)

Rsq = (1-SSE)/SST

ols = list()

  ols$b = b

  ols$y_hat = y_hat

  ols$e = e

  ols$df = df

  ols$SSE = SSE

  ols$SST = SSE

  ols$MSE = MSE
```

```
  ols$RMSE = RMSE
  ols$Rsq = Rsq

return(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 ybar's within group).

```{r}
X = model.matrix(~Type,MASS::Cars93)[,-1]#everything but 1
y = MASS::Cars93$Price
typrce = lm(y~X)
typrce$coefficients
# Call:
# lm(formula = Price ~ Type, data = cars)
#
# Coefficients:
# (Intercept)    TypeLarge  TypeMidsize    TypeSmall   TypeSporty      TypeVan
#     18.2125       6.0875       9.0057      -8.0458       1.1804       0.8875



```

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

```{r}
g = function(x_star, X, y){
  X = cbind(1, X)
  Xt = t(X)
  XtX = Xt%*%X
  XtXinv = solve(XtX)
  b = XtXinv%*%Xt*y
  c(1,x_star)%*%b #need to append
```