

Lecture 7: Simple linear regression

Stats 32: Introduction to R for Undergraduates

Harrison Li

April 23, 2024

Agenda

- 1 Dependent and independent variables
- 2 Ordinary least squares
- 3 `lm()`
- 4 Binary predictors

Reading: Chapter 5 (except 5.1.3 and 5.2.3)

Dependent and independent variables

Dependent and independent variables

Today, we will begin our final unit of the course on **data analysis**. In particular, we will focus on simple linear regression, a standard and widely useful technique for quantifying the relationship between two variables.

In simple linear regression, we have one **dependent variable** (or outcome variable), typically labeled y , and one **independent variable** (or explanatory/predictor variable), typically labeled x . We want to understand how changes in the independent variable (x) correspond to changes in the dependent variable (y).

Correlation

To understand the idea of a “relationship” between two variables, we introduce the mathematical concept of **correlation**.

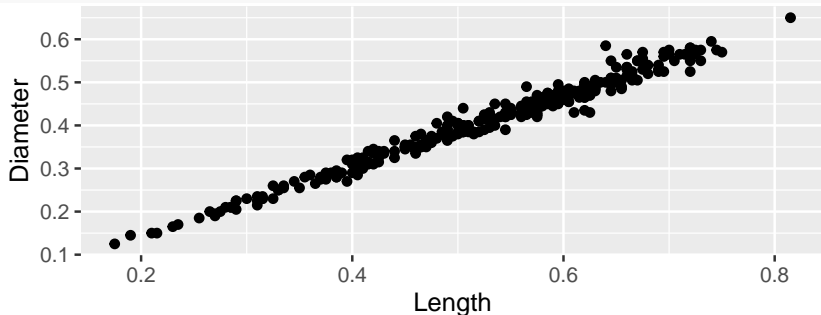
We say two variables x and y are **positively correlated** if an increase in one is associated with an increase in the other. This can be determined visually by a scatterplot of x versus y that is concentrated around an upward sloping line.

By contrast, x and y are **negatively correlated** if an increase in one is associated with a *decrease* in the other. This corresponds to a scatterplot concentrated around a downward sloping line.

Correlation

Is Length positively correlated, negatively correlated, or uncorrelated with Diameter in the abalone tibble?

```
library(tidyverse)
abalone <- read_csv("abalone.csv")
abalone %>%
  ggplot(aes(x=Length, y=Diameter)) +
  geom_point()
```



Correlation coefficient

The **correlation coefficient** is a numerical measure of correlation between two (numeric) data variables. It takes on values between -1 and 1.

- A coefficient of 1 indicates a perfect positive correlation, i.e. data lie perfectly along a straight line with positive slope
- A coefficient of -1 indicates a perfect negative correlation, i.e. data lie perfectly along a straight line with negative slope
- A coefficient of 0 indicates no correlation

The correlation coefficient can be computed with `cor()`. We see Length and Diameter are very highly (positively) correlated:

```
cor(abalone$Length, abalone$Diameter)
```

```
## [1] 0.9881948
```

Confounding variables

Correlation does not imply causation!

A strongly positive or negative correlation does NOT necessarily mean changes in x *cause* changes in y , or vice versa.

For example, the number of shark attacks in each month is positively correlated with monthly ice cream consumption in the United States. But it would be silly to conclude that eating ice cream causes you to be more likely to get eaten by a shark!

The key issue here is that both are associated with hot weather (more people going to the beach, and wanting to eat ice cream to cool off). In this case, the outdoor temperature is called a **confounding variable**.

Ordinary least squares

Ordinary least squares

All models are wrong but some models are useful – George Box

Simple linear regression is the most basic way to *model* the relationship between y and x . Specifically, it models y to be approximately a linear function of x .

Mathematically we can write this as

$$y = \beta_0 + \beta_1 * x + e$$

where β_0 is the intercept, β_1 is the slope of the line, and e is an error term that represents “random noise” centered around 0, which is not explicitly modeled.

Note that this linear approximation may be poor in some settings, but is often still useful.

Estimating the slope and intercept

Once we've decided to model y as a linear function of x , it remains to figure out how we can find values β_0 and β_1 to “fit” the data. By far the most common method of doing so is **ordinary least squares** (OLS).

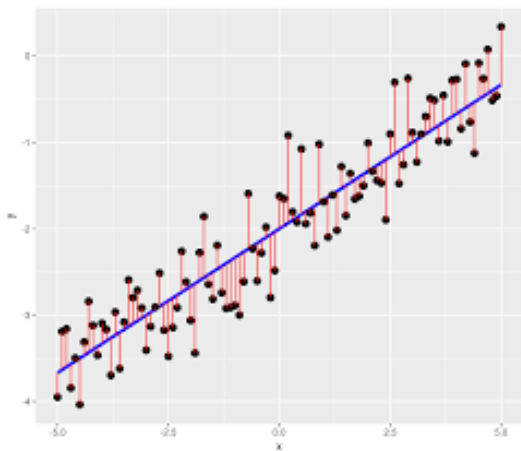
As the name suggests, OLS finds β_0 and β_1 to minimize the *sum of squared deviations between y and the regression line*.

Mathematically, β_0 and β_1 are selected to minimize $\sum_{i=1}^n (y_i - \beta_0 - \beta_1 * x_i)^2$.

Here $(x_1, y_1), \dots, (x_n, y_n)$ are the n pairs of independent and dependent variables in our tibble.

A picture of OLS

OLS considers all possible blue lines and picks the one that minimizes the sum of the squared lengths of the red lines:



$\ln()$

lm()

While you can find algebraic expressions for β_0 and β_1 in any introductory statistics textbook, R can easily compute them for you using the built-in `lm()` function.

Let's regress Diameter (dependent variable) on Length (independent variable) in the `abalone` tibble, corresponding to the scatterplot we saw earlier. We input the result of the `lm()` call into `summary()` to get some important information about the model:

lm()

```
library(moderndiver)
summary(lm(formula=Diameter~Length, data=abalone))

##
## Call:
## lm(formula = Diameter ~ Length, data = abalone)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.059475 -0.009019 -0.000988  0.008421  0.083292
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.020211   0.003967  -5.094 6.22e-07 ***
## Length      0.815498   0.007324 111.348 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01579 on 298 degrees of freedom
## Multiple R-squared:  0.9765, Adjusted R-squared:  0.9765
## F-statistic: 1.24e+04 on 1 and 298 DF, p-value: < 2.2e-16
```

We get $\beta_1 = 0.815$ and $\beta_0 = -0.020$ (only look at the estimate column under Coefficients for now).

Formula and data

Our call to `lm()` looked like this: `lm(formula=Diameter ~ Length, data=abalone)`

The first argument to `lm()` is called a **formula**. The column name corresponding to the dependent variable `y` goes first, followed by a tilde `~`, followed by the column name corresponding to the independent variable `x`.

The second argument, `data`, is simply the name of the tibble where the named columns can be found. Think of it like the first argument in a `dplyr` verb (though in `lm()`, it is no longer the first argument)

Use the data argument!

Note that the data argument to `lm()` is optional; we could have simply written

```
lm(formula=abalone$Diameter~abalone$Length) %>%  
  get_regression_table()
```

```
## # A tibble: 2 x 7  
##   term                estimate std_error statistic p_value lower_ci upper_ci  
##   <chr>              <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>  
## 1 intercept         -0.02     0.004    -5.09     0    -0.028  -0.012  
## 2 abalone$Length      0.815     0.007    111.      0     0.801   0.83
```

but this will lead to annoying errors in later sections. It's best practice to have both variables in the same tibble (as appropriately named columns), and to use the data argument. It's kind of similar in spirit to the `.data` argument in `dplyr` verbs, though `lm()` long predates the tidyverse.

Plotting

We can use `geom_abline()` to add a line onto a plot created using `ggplot()`. This lets us easily visualize the regression line on top of the scatterplot.

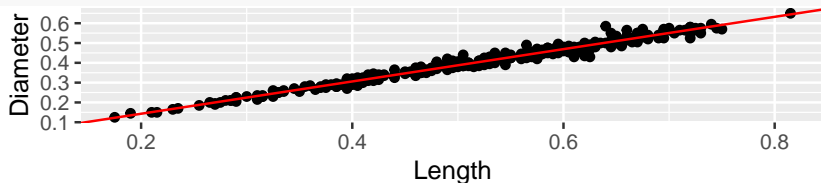
We need to pass in the slope and intercept to `geom_abline()`. The output of `lm()` is a complex named list; one of its entries is named `coefficients`, so we can extract it with the `$` notation:

Plotting

```
model <- lm(formula=Diameter ~ Length, data=abalone)
coefs <- model$coefficients
coefs

## (Intercept)      Length
## -0.02021087  0.81549812

intercept <- coefs[1]
slope <- coefs[2]
abalone %>%
  ggplot(aes(x=Length, y=Diameter)) +
  geom_point() +
  geom_abline(slope=slope, intercept=intercept, colour="red")
```



Indeed, the regression line seems to fit the data well!

Prediction

Once we have the slope and intercept of the regression line (β_0 and β_1), we can use them to predict y given x . For instance, how would we predict the diameter of an abalone with length 0.65?

Answer: we simply find the y value on our regression line corresponding to the x value 0.65:

$$\beta_0 + \beta_1 * x = -0.020 + 0.815 * 0.65 = 0.51$$

Prediction

We can let R automatically do the prediction for us using the `predict()` function. We pass it two arguments: the `lm()` object returned by the call to `lm()` (which we stored in the variable `model`), and the `newdata` argument, which must be a *one-column tibble* (or data frame) of new predictor values *with the same column name as when we fit the model*:

```
predict(object=model, newdata=tibble(Length=0.65))
```

```
##           1  
## 0.5098629
```

Prediction

Note if you don't specify `newdata`, R will return the predictions for all the `x` values in the original data:

```
head(predict(object=model))
```

```
##           1           2           3           4           5           6  
## 0.4894755 0.4853980 0.4364681 0.2815234 0.4650105 0.4201581
```

You can get predictions for multiple new data points simultaneously, if the tibble passed into the `newdata` argument has multiple rows:

```
predict(object=model, newdata=tibble(length=c(0.3, 0.4, 0.7)))
```

```
##           1           2           3  
## 0.2244386 0.3059884 0.5506378
```

Interpretation

How do we interpret our values for β_0 and β_1 ?

From the regression equation $y \approx \beta_0 + \beta_1 * x$, we see that β_0 is the prediction when $x = 0$.

β_1 is the slope of the regression line, and hence corresponds to the predicted change in y for each 1 unit increase in x . In our abalone example, we can say that we estimate each unit increase in diameter is associated with a 0.815 unit increase in length.

Binary predictors

Binary predictors

So far, we've used simple linear regression under the assumption that both the outcome variable and the predictor are quantitative.

However, linear regression also works well when the predictor is categorical. Today, we will consider the case of a **binary predictor**, that is, a categorical predictor that can only take on one of 2 possible values.

Binary predictors

Typically, binary predictors are encoded as 0 or 1. We must specify which possibility corresponds to 0, and which corresponds to 1.

From the regression line equation $y = \beta_0 + \beta_1 * x$, we see that if x is binary, the regression prediction is β_0 when $x = 0$ and $\beta_0 + \beta_1$ when $x = 1$.

Thus, β_1 can be interpreted as: based on my regression, how much greater do I expect y to be when $x = 1$, compared to when $x = 0$?

Binary predictors

Let's define a `dep_state` variable in `flights` for the state of the origin airport for each flight (either NY or NJ), and regress the arrival delay `arr_delay` against `dep_state`.

Since `flights$dep_state` will be a character vector taking on 2 possible values, `lm()` will know to treat this as a binary predictor (in fact it will convert it to a factor under the hood).

Binary predictors

```
library(nycflights13)
flights_reg <- flights %>%
  mutate(dep_state=if_else(origin=="EWR", "NJ", "NY"))
summary(lm(formula=arr_delay~dep_state, data=flights_reg))

##
## Call:
## lm(formula = arr_delay ~ dep_state, data = flights_reg)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -95.11  -23.66  -11.66    7.34  1266.34
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.1071    0.1303   69.88  <2e-16 ***
## dep_stateNY  -3.4440    0.1626  -21.18  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 44.6 on 327344 degrees of freedom
## (9430 observations deleted due to missingness)
## Multiple R-squared:  0.001368, Adjusted R-squared:  0.001365
## F-statistic: 448.4 on 1 and 327344 DF, p-value: < 2.2e-16
```

Binary predictors

We see a row in the coefficients table labeled `dep_stateNY`. This indicates that $x=1$ corresponds to `dep_state="NY"` while $x=0$ corresponds to `dep_state="NJ"`.

By default, R sets the reference level (the level corresponding to $x=0$ when used as a predictor in `lm()`) for a factor variable to be the first alphabetically. This can be changed using `fct_relevel()` (recall Lab 2).

We conclude, based on the coefficient estimate of -3.444 , that flights leaving from the NY airports (JFK or LGA) tend to be delayed an average of 3.444 minutes less than flights leaving from EWR (in NJ).