Fitting Linear Models with `lm()`

Gaston Sanchez

Stat 154, Spring 2018

Classic Linear Regression

In R, the function that allows you to fit a regression model via Least Squares is `lm()`, which stands for *linear model*. I should say that this function is a general function that works for various types of linear models such as regression, single stratum analysis of variance, and analysis of covariance.

The main arguments to `lm()` are:

```
lm(formula, data, subset, na.action)
```

where:

- `formula` is the *model formula* (the only required argument)
- `data` is an optional data frame
- `subset` is an index vector specifying a subset of the data to be used (by default all items are used)
- `na.action` is a function specifying how missing values are to be handled (by default missing values are omitted)

Let's see an example of how to use `lm()`. For illustration purposes we'll use a couple of variables from the data `mtcars`. Say we want to fit a simple linear model in which `mpg` is regressed on `disp`, which would correspond to:

$$\texttt{mpg} = \beta_0 + \beta_1 \texttt{disp} + \varepsilon$$

How do you specify the above model with `lm()`? When the predictor(s) and the response variable are all in a single data frame, you can use `lm()` as follows:

```
# simple linear regression
reg <- lm(mpg ~ disp, data = mtcars)
```

The first argument of `lm()` consists of an R formula: `mpg ~ disp`. The tilde, `~`, is the formula operator used to indicate that `mpg` *is predicted* or *described* by `disp`.

The second argument, `data = mtcars`, is used to indicate the name of the data frame that contains the variables `mpg` and `disp`, which in this case is the object `mtcars`. Working with data frames and using this argument is strongly recommended.

The example above is a simple linear regression (i.e. only one predictor). To fit a multiple linear model, just include more predictors:

```
# multiple linear regression
reg <- lm(mpg ~ disp + hp, data = mtcars)
```

Output of `lm()`

Let's go back to the simple regression model in which `mpg` is regressed on `disp`:

```
reg <- lm(mpg ~ disp, data = mtcars)
```

We are storing the output of `lm()` in the object `reg`. Technically, `reg` is an object of class `"lm"`. When you print `reg` R displays the following information:

```
# default output
reg
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
```

Notice that the output contains two parts: `Call:` and `Coefficients:`.

The first part of the output, `Call:`, simply tells you the command used to run the analysis, in this case: `lm(formula = mpg ~ disp, data = mtcars)`.

The second part of the output, `Coefficients:`, shows information about the regression coefficients. The intercept is 29.6, and the other coeffcient is -0.0412. Observe the names used by R to display the intercept b_0. While the intercept has the same name `(Intercept)`, the non-intercept term is displayed with the name of the associated variable `disp`.

The printed output of `reg` is very minimalist. However, `reg` contains more information. To see a list of the different components in `reg`, use the function `names()`:

```
# what's in an "lm" object?
names(reg)
```

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

As you can tell, `reg` contains many more things than just the `coefficients`. In fact, the output of `lm()` is heavily focused on statistical inference, designed to provide results that you can use to form confidence intervals and perform significance tests.

Here's a short description of each of the output elements:

- `coefficients`: a named vector of coefficients.
- `residuals`: the residuals, that is, response minus fitted values.
- `fitted.values`: the fitted mean values.
- `rank`: the numeric rank of the fitted linear model.
- `df.residual`: the residual degrees of freedom.
- `call`: the matched call.
- `terms`: the terms object used.
- `model`: if requested (the default), the model frame used.

To inspect what's in each returned component, type the name of the regression object, `reg`, followed by the `$` dollar operator, followed by the name of the desired component. For example, to inspect the `coefficients` run this:

```
# regression coefficients
reg$coefficients
```

```
## (Intercept)        disp
## 29.59985476 -0.04121512
```

Likewise, to take a peek at the fitted values use `$fitted.values`

```
# fitted values
head(reg$fitted.values, n = 8)
```

```
##         Mazda RX4     Mazda RX4 Wag        Datsun 710    Hornet 4 Drive
##          23.00544          23.00544          25.14862          18.96635
## Hornet Sportabout           Valiant        Duster 360         Merc 240D
##          14.76241          20.32645          14.76241          23.55360
```

Alternatively, `lm()`—and other similar model fitting functions—have generic helper functions to extract the output elements such as:

- `coef()` to extract the coefficients
- `fitted()` to extract the fitted values
- `residuals()` to extract the residuals

About Model Formulae

The `formula` declaration in `lm()` was originally introduced as a way to specify linear models, but have since been adopted for so many other purposes. An R formula has the general form:

$$\texttt{response} \sim \texttt{expression}$$

where the left-hand side, `response`, may in some uses be absent and the right-hand side, `expression`, is a collection of terms joined by operators usually resembling an arithmetical expression. The meaning of the right-hand side is context dependent.

The `formula` is interpreted in the context of the argument `data` which must be a list, usually a data frame; the objects named on either side of the formula are looked for first in `data`. If no data frame is provided, then R will search for the specified objects (i.e. variables) in the global environment. So, the following calls to `lm()` are equivalent:

```
# with argument 'data'
lm(mpg ~ disp + hp, data = mtcars)

# without argument 'data'
lm(mtcars$mpg ~ mtcars$disp + mtcars$hp)
```

Notice that in these cases the `+` indicates *inclusion*, not addition. You can also use `-` which indicates *exclusion*.

As mentioned above, the formula expression `mpg ~ disp` corresponds to the linear model:

$$\texttt{mpg} = \beta_0 + \beta_1 \texttt{disp} + \varepsilon$$

In vector-matrix notation, we would have a model expression like this:

$$\underset{n\times 1}{\mathbf{y}} = \underset{n\times 2}{\mathbf{X}} \times \underset{2\times 1}{\boldsymbol{\beta}} + \underset{n\times 1}{\boldsymbol{\varepsilon}}$$

When calling `lm(mpg ~ disp + hp, data = mtcars)`, R will create a **model matrix X** that would conceptually correspond to:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} \\ 1 & x_{21} \\ 1 & x_{31} \\ \vdots & \vdots \\ 1 & x_{n1} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

In other words, R will include a column of ones in $\mathbf{X}$ for the intercept term.

Another useful syntax when working with formulas is the dot `"."` character. Sometimes you will find the dot `.` as part of a formula declaration, for example:

```
# fit model with all available predictors
reg_all <- lm(mpg ~ ., data = mtcars)
```

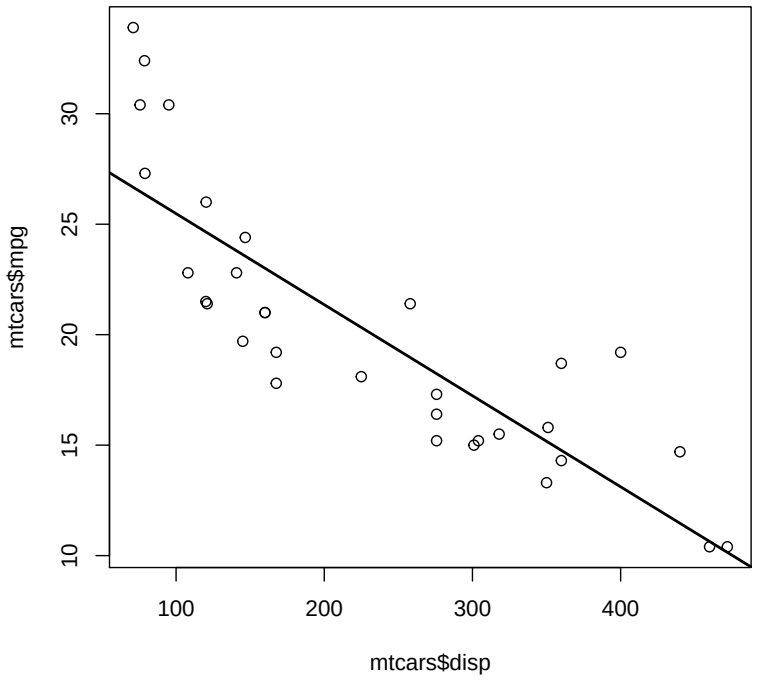
The dot `"."` has a special meaning; in `lm()` it means *all the other variables* available in the object `data`. This is very convenient when your model has several variables, and typing all of them becomes tedious. The previous call is equivalent to:

```
# verbose: fit model with all available predictors
reg_all <- lm(mpg ~ cyl + disp + hp + drat + wt + qsec +
              vs + am + gear + carb, data = mtcars)
```

Plotting the Regression Line

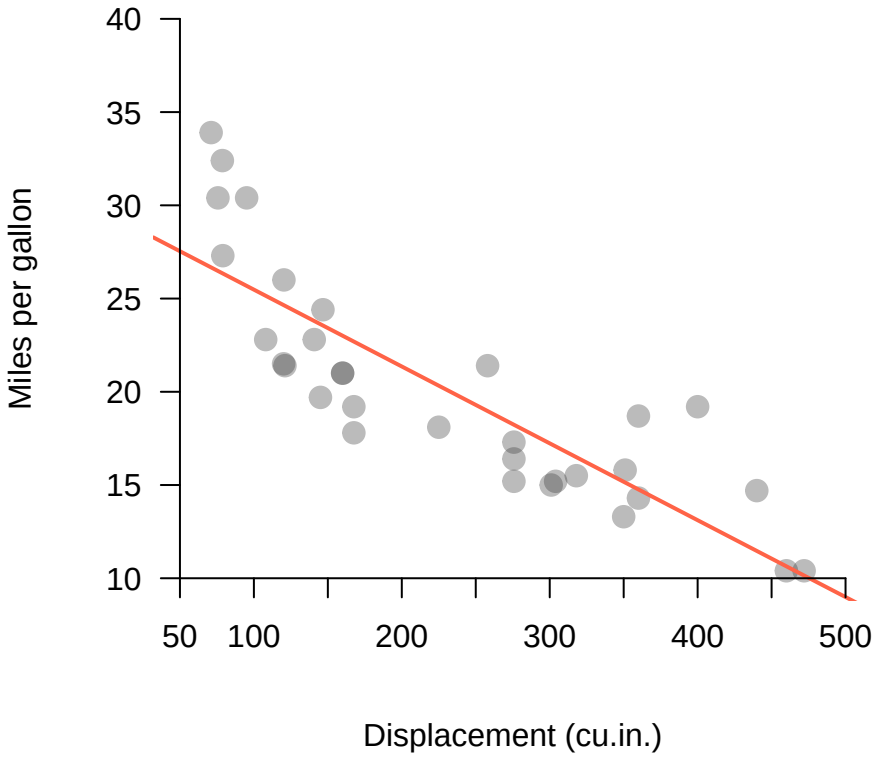
With a simple linear regression model (i.e. one predictor), once you obtained the `"lm"` object `reg`, you can use it to get a scatterplot with the regression line on it. The simplest way to achieve this visualization is to first create a scatter diagram with `plot()`, and then add the regression line with the function `abline()`; here's the code in R:

```
# scatterplot with regression line
plot(mtcars$disp, mtcars$mpg)
abline(reg, lwd = 2)
```



The function `abline()` allows you to add lines to a `plot()`. The good news is that `abline()` recognizes objects of class `"lm"`, and when invoked after a call to `plot()`, it will add the regression line to the plotted chart.

Here's how to get a nicer plot using low-level plotting functions:




```
# scatterplot with regression line
plot.new()
plot.window(xlim = c(50, 500), ylim = c(10, 40))
title(xlab = 'Displacement (cu.in.)', ylab = 'Miles per gallon')
points(mtcars$disp, mtcars$mpg, pch = 19, cex = 1.5, col = "#33333355")
abline(reg, col = "tomato", lwd = 2)   # regression line
axis(side = 1, pos = 10, at = seq(50, 500, 50))
axis(side = 2, las = 1, pos = 50, at = seq(10, 40, 5))
```

Your turn

- Use `lm()` to regress `mpg` on `disp`. Store the output in an object called `reg1`.
- Now, mean-center the `mtcars` data.
- Re-run `lm()` regressing `mpg` on `disp` using the centered data. Store the output in an object called `reg2`.
- The intercept term that you obtained in `reg2` should be zero, since the used variables are mean-centered. How would you recover the intercept term?
- Now, standardize the `mtcars` data (mean = 0, var = 1).
- Re-run `lm()` regressing `mpg` on `disp` using the standardized data. Store the output in an object called `reg3`.

- From `reg3`, how would you recover the un-standardized $\hat{\beta}_0$ and $\hat{\beta}_1$—like those in `reg1`?
- Find out how to use `lm()` in order to exclude the intercept term β_0, that is, without a coefficient β_0

$$\text{mpg} = \beta_1 \text{disp} + \varepsilon$$

- `mtcars` has a column `am` for *automatic transmission*: 0 = automatic, 1 = manual. Find out how to use the argument `subset` of `lm()` to regress `mpg` on `disp`, with just those cars having automatic transmission.

Summary method of an object `"lm"`

As with many objects in R, you can apply the function `summary()` to an object of class `"lm"`. This will provide, among other things, an extended display of the fitted model. Here's what the output of `summary()` looks like with our object `reg`:

```
# summary of an "lm" object
reg <- lm(mpg ~ disp + hp, data = mtcars)
reg_sum <- summary(reg)
reg_sum
```

```
##
## Call:
## lm(formula = mpg ~ disp + hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7945 -2.3036 -0.8246  1.8582  6.9363
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.735904   1.331566  23.083  < 2e-16 ***
## disp        -0.030346   0.007405  -4.098 0.000306 ***
## hp          -0.024840   0.013385  -1.856 0.073679 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.127 on 29 degrees of freedom
## Multiple R-squared:  0.7482, Adjusted R-squared:  0.7309
## F-statistic: 43.09 on 2 and 29 DF,  p-value: 2.062e-09
```

There's a lot going on in the output of `summary()`. So let's examine all the returned pieces.

1) The first part of the output, `Call:`, corresponds to the command that we used to fit the model with `lm()`, in this case: `lm(formula = mpg ~ disp, data = mtcars)`.

2) The second part, `Residuals:`, has the 5-number summary of the computed residuals: minimum, 1st quartile, median, 3rd quartile, and maximum. In case you wonder, you can visualize this numeric summaries with a boxplot

```
# boxplot of residuals
boxplot(residuals(reg), horizontal = TRUE, ylim = c(-6, 8), axes = FALSE)
axis(side = 1)
```

3) The 3rd part, `Coefficients`, corresponds to a table with five columns, and as many rows as coefficient estimates.

   - The first column has the names of the coefficients: `(Intercept)` and `disp`.
   - The second column contains the estimated values.
   - The third column has the standard error of the estimates.
   - The fourth column has the t-statistic values.
   - The fifth column corresponds to the p-values associated to t.

Notice all those asterisks next to the p-values. Both estimates are marked with three stars, indicating a p-value of less than 0.001.

These p-values correspond to tests of the (null) hypothesis:

$$H_0 : \beta_i = 0$$

under the assumptions of the classic linear model (i.e. linearity, normality, homoscedasticity, independence). We'll say more things about the table of coefficients shortly.

4) The 4th and last part is comprised by the last three lines of text. Here there is also a lot going on. We have the following elements:

   - Residual Standard Error (RSE)
   - Coefficient of determination R^2
   - Adjusted R^2
   - F-statistic
   - And p-value associated to the F-statistic.

Confidence Intervals for β_j's

Confidence intervals provide an alternative way of expressing the uncertainty in the estimates of $\beta_0, \beta_1, \ldots, \beta_p$. The confidence intervals (CIs) take the form:

$$\hat{\beta}_j \pm t_{n-p-1}^{(\alpha/2)} \, \text{se}(\hat{\beta}_j)$$

We can construct individual 95% CIs for $\beta_{\texttt{disp}}$ for which we need the 2.5% and 97.5% percentiles of the t-distribution with 32 - 1 degrees of freedom:

```r
qt(0.975, n - p - 1)
```

```
## [1] 2.04523
```

If the interval contains zero, this indicates that the null hypothesis $H_0 : \beta_{\texttt{disp}} = 0$ would not be rejected at the 5% level.

Very conveniently, R provides the function `confint()` that computes all the univariate intervals of an `"lm"` output:

```r
# confidence intervals of each coefficient estimate
confint(reg)
```

```
##                   2.5 %       97.5 %
## (Intercept) 28.01254573 33.459262767
## disp        -0.04549091 -0.015201645
## hp          -0.05221650  0.002536338
```

Common Hypothesis Tests

Two types of test of significance can be performed using the output of `summary.lm()`:

1. Whether a given coefficient β_j is zero
2. Whether all the predictors are zero.

Test of all the predictors

Are any of the predictors useful in predicting the response? This corresponds to the null hypothesis:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$$

To test this hypothesis, we use an F-statistic given by:

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n - p - 1)}$$

Large values of F would indicate rejection of the null hypothesis. Traditionally, the information in the above test is presented in an Analysis of Variance (ANOVA) table:

Source	Deg. Freedom	Sum of Squares	Mean Square
Regression	p	ESS	ESS/p
Residual	$n - p - 1$	RSS	RSS/$(n - p - 1)$
Total	$n - 1$	TSS	

Besides the `summary.lm()` output fo the `F-statistic`, R also provides the `anova()` function to obtain the above *Analysis of variance table*

```
reg_anova <- anova(reg)
reg_anova
```

```
## Analysis of Variance Table
##
## Response: mpg
##           Df Sum Sq Mean Sq F value    Pr(>F)
## disp       1 808.89  808.89 82.7454 5.406e-10 ***
## hp         1  33.67   33.67  3.4438   0.07368 .
## Residuals 29 283.49    9.78
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This table breaks down the sum of squares about the mean, for the response variable, in two parts: a part that is accounted for by the deterministic component of the model, and a part attributed to the noise component or residual.

The total sum of squares (about the mean) for the 32 observations is 1126.0471875. Including the variable `disp` rediced this by 808.888, giving a residual sum of squares equal to 33.665.

This table has the information needed for calculating R^2, also known as the *coefficient of determination* and adjusted R^2. The R^2 statistic is the square of the correlation coefficient, and is the sum of squares due to `disp` divided by the total sum of squares: $R^2 = 0.7183$.

Your turn

- What class of object is `reg_sum`?
- What does `reg_sum` contain?
- Find out how to turn-off the displayed stars (i.e. asterisks) referred to as *significance codes* (these are just interpretative features). Tip: read the documentation of `options()`.

Auxiliary plots

Model assumptions should be checked as far as is possible. The common checks are:

- **a plot of residuals versus fitted values**: for example there may be a pattern in the residuals that suggests that we should be fitting a curve rather than a line;
- **a normal probability**: if residuals are from a normal distribution points should lie, to within statistical error, close to a line.

Your turn: How do you get such plots? The easiest way is to apply `plot()` to the object `"lm"` object, and specify the argument `which`:

- residuals vs fitted model: `plot(reg, which = 1)`
- normal probability plot: `plot(reg, which = 2)`

Interpretation of these plots may require a certain amount of practice. This is specially the case for normal probability plots. Keep in mind that these diagnostic plots are not definitive. Rather, they draw attention to points that require further investigation.

Predictions

In addition to the `summary.lm()` and `print.lm()` functions, `"lm"` objects also have an associated `predict.lm()` function.

Suppose we wished to predict *future* consumption of miles per gallon `mpg`. The first step is to create a new data frame with a variables `disp` and `hp` containing the new values, for example:

```
# data frame with new data
new_data <- data.frame(
  disp = 200,
  hp = 150,
  row.names = 'new car')

new_data
```

```
##         disp  hp
## new car  200 150
```

To obtain the predictions we call `predict()` and pass the `"lm"` object and the data frame for the argument `newdata`:

```
predict(reg, newdata = new_data)
```

```
##  new car
## 20.94064
```

The `predict()` method for `"lm"` objects works by attaching the estimated coefficients to a new model matrix that it constructs using the formula and the new data.

Now let's get predictions with more new observations:

```
# data frame with more new observations
new_data <- data.frame(
  disp = seq(100, 200, by = 25),
  hp = seq(80, 120, by = 10),
  row.names = paste0('car_', letters[1:5]))

new_data
```

```
##       disp  hp
## car_a  100  80
## car_b  125  90
## car_c  150 100
## car_d  175 110
## car_e  200 120
```

and obtain the predicted `mpg`'s:

```
predict(reg, newdata = new_data)
```

```
##    car_a    car_b    car_c    car_d    car_e
## 25.71407 24.70701 23.69995 22.69290 21.68584
```