Fitting Regression Lines with `lm()`

Stat 131A, Fall 2018, Prof. Sanchez

Learning Objectives

- Getting to know the function `lm()`
- How to add the regression line to a scatter diagram

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:

```r
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:

```r
# 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 (although not mandatory).

Output of `lm()`

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

```r
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:

```r
# 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()`:

```r
# 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
```

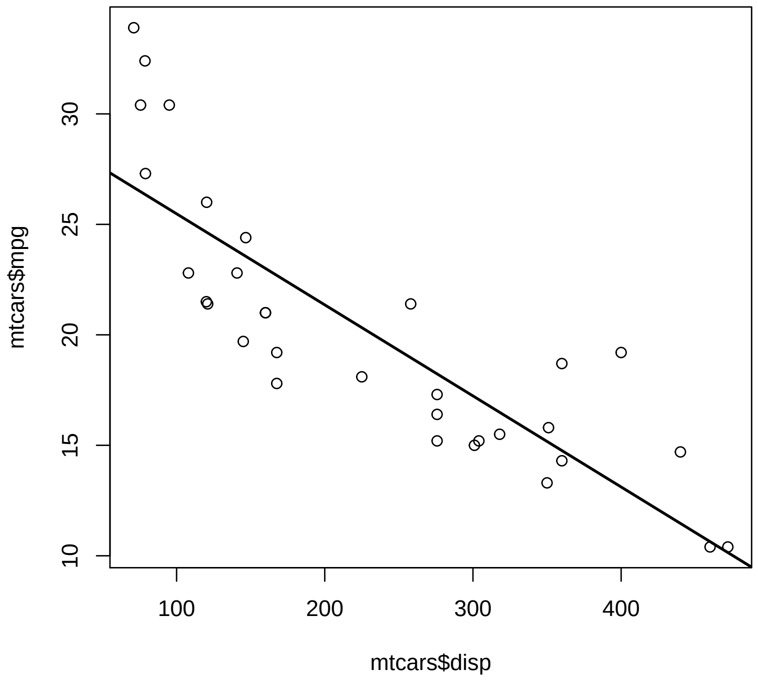
Alternatively, `lm()`—and other similar model fitting functions—have generic helper functions to extract the output elements such as:

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

Plotting the Regression Line

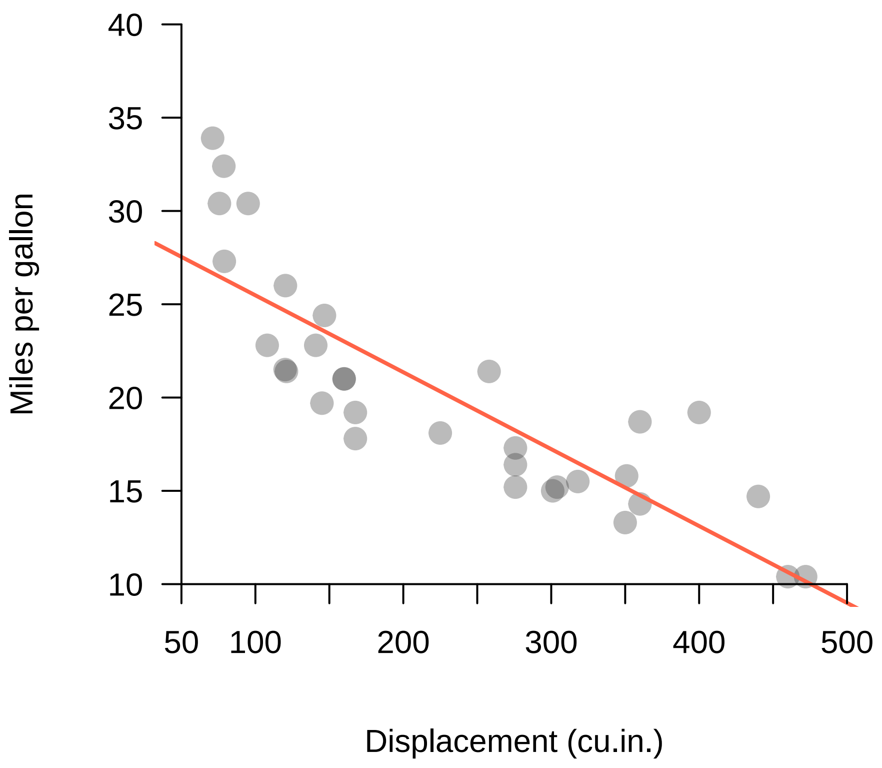
With a simple linear regression model (i.e. one predictor), once you obtained the `"lm"` object `reg`, you can use it to get a scatterplot with the regression line on it. The simplest way to achieve this visualization is to first create a scatter diagram with `plot()`, and then add the regression line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



```r
# 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))
```

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`:

```r
# 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

```r
# 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.

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.