Linear models, assumptions and outputs

UA Summer R Workshop: Week 2

Nicholas M. Caruso
Christina L. Staudhammer

7 June 2016

Create data

Before we start, we'll create a dataframe that includes a continuous response and a continuous explanatory variable using a function(). Functions are helpful to create in situations where you might otherwise be re-writing the same code multiple times. Within a function, we can establish the arguments (inside the parentheses), the argument defaults (not necessary, however, you must supply values for all arguments that do not have defaults), the function statements (what it does, within brackets {}), and the function values (what the function returns). We can then store the results of our function into a name. Alternatively, we would read in our dataframe using read.csv().

```
library(tidyr)
library(dplyr)

lmdat.fn <- function(n, alpha=1.5576, beta.var=-2.6513,
                     sd.noise=4.5){
# n: Number of individuals
# alpha, beta.var: coefficients (intercept and slope)

# Generate explanatory variable
variable <- rnorm(n, mean=0, sd=1)

# Signal: Build up systematic part of the LM
expected.response <- alpha + beta.var*variable

# Noise: generate error (normally distributed) around expected response
response <- rnorm(n, mean=expected.response, sd=sd.noise)
response2 <- expected.response + rnorm(n, mean=0, sd=sd.noise)

# Return dataframe
return(data_frame(variable=variable, response=response,
                  expect.response=expected.response,
                  response2=response2))
}

set.seed(8675309)
dat <- lmdat.fn(n=100, alpha=1.5576, beta.var=-1.5, sd.noise=3)

# write.csv(dat, 'dat_2june2016.csv')
# dat <- read.csv('dat_2june2016.csv')

library(ggplot2)

ggplot(dat, aes(variable, response)) +
  geom_point()
```

Linear model

We use the lm() function to call a linear model. The main arguments in this function are the formula, which is written as the response followed by a tilde (~), then the explanatory variable(s), and then the data that we are using (typically a data frame). This is typically stored into a name which can be used to inspect the model further.

```
mod.lm <- lm(response~variable, data=dat) # store the model
```

Linear model assumptions

1) The model is correctly specified (linearity)
2) Each $y_i|x_i$ is normally distributed (given is important!)
3) The y_i are homoscedastic
4) Any observation of y_i is independent of all other y_i
5) For a given x_i, the observations of y_i are randomly selected
6) The values of x_i are fixed and/or measured without error

Assumptions 1-3 can be assessed using R, while assumptions 4-6 should be addressed in the study design or by using an alternative model; thus we will focus on the first three assumptions. These can be assessed graphically in R using the plot() function. This produces 4 figures, if you just use plot(), R will cycle through the 4 figures, asking you to hit return to scroll through them. Alternatively you can adjust the mfrow graphic parameter par() so that we create a plot with 4 panels (2 rows and 2 columns) to view all of them at once, but we will go over each plot individually.

```
par(mfrow=c(2,2))
plot(mod.lm)
```

Plot 1: Linearity and homoscedasticity

The first plot of the residuals (error, y-axis) vs the fitted values ($\hat{y}$, x-axis) assess both the linearity and homoscedasticity assumptions. If our data are linear, the red line will be relatively flat. We can also use this plot to assess if the residual standard error is constant (Homoscedasticity), in which the data points would be scattered with no obvious pattern. Our model meets both of the assumptions based on this figure.

```
plot(mod.lm, 1)
```

Violations in plot 1

Here is an example of two responses that do not meet these assumptions. For the first response, I created a non linear response, in which the response is a polynomial function (3rd order) of our explanatory variable with added normally distributed error. The second response was created with a trend in the normally distributed error, in which size and standard deviation of the error are positively related to the explanatory variable. The top graphics show the data (A) and a violation of the assumption of linearity (B). The bottom graphics show the data (C) and a violation of the assumption of homoscedasticity (D).

```
dat <- dat %>%
  mutate(response.nonlinear=-3.453*variable + -2.345*(variable^2) + -1.97*(variable^3) +
           rnorm(length(variable), 0, 3),
         response.nonconstant=-3.453*variable + rnorm(length(variable),
                                                        rank(variable), rank(variable)))

par(mfrow=c(2,2))
plot(response.nonlinear~variable, dat)
```

```r
mtext('A', side=3, line=1, adj=0, font=2)
plot(lm(response.nonlinear~variable, dat), 1) # non linear, but constant variance
mtext('B', side=3, line=1, adj=0, font=2)
plot(response.nonconstant~variable, dat)
mtext('C', side=3, line=1, adj=0, font=2)
plot(lm(response.nonconstant~variable, dat), 1) # variance is not constant, but is linear
mtext('D', side=3, line=1, adj=0, font=2)
```

Plot 2: Errors are normally distributed

The second plot assesses whether the errors are normally distributed. This can also be tested using a Shapiro Wilks test (null hypothesis is that the errors are normally distributed), a common mistake is to test the normality of the response rather than of the errors (residuals). We can call the residuals of the model using the residuals() function. Additionally, most models are robust to moderate violations of the normality assumption, often this test would have to show extreme deviations from normality (i.e., $p << 0.05$) to preclude use of a linear model. Alternatively, we can use a qqplot to graphically inspect our model. For the graph, the data should follow the dotted line.

```r
plot(mod.lm, 2)

shapiro.test(residuals(mod.lm))
```

Violations in plot 2

Here is an example of a response that violates the normality assumption.

```r
dat <- dat %>%
  mutate(response.notnormal=exp(response))

shapiro.test(residuals(lm(response.notnormal~variable, dat)))

par(mfrow=c(1,2))
plot(response.notnormal~variable, dat)
plot(lm(response.notnormal~variable, dat), 2)
```

Shapiro Wilks test vs. graphic diagnostics

A good reason to inspect the model using plots rather than Shapiro Wilks test is that the Shapior Wilks test test is sensitive to large sample sizes and will often show non-normally distributed errors with larger datasets (Type I error). Obviously, our data was created using normally distributed errors, and the diagnostic plot shows no deviations from normality in the errors.

```r
set.seed(7375)
dat.large <- lmdat.fn(n=5000, alpha=1.5576, beta.var=-1.5, sd.noise=1)

shapiro.test(residuals(lm(response~variable, dat.large)))

par(mfrow=c(1,2))
plot(response~variable, dat.large)
plot(lm(response~variable, dat.large), 2)
```

Plot 3: Linearity and homoscedasticity (standardized residuals)

The third diagnostic plot can also assess non-linearity and homoscedasticity. The third plot is very similar to the first, except the scale of the residuals is standardized such that they are all positive (i.e., large negative residuals become large positive residuals).

```
plot(mod.lm, 3)
```

Plot 4: Leverage and Cook's distance

The last plot (actually 5th of the diagnostic plots, there's six total), shows the standadized residuals against leverage. Leverage is a measure of how much impact that a given y_i has on $\hat{y}_i$, such that the further a given x_i is from the $\bar{x}$, the more influence it has on the estimate of regression coefficients (higher leverage). For this plot, the red smoothed line should stay close to the gray dotted line and no points should have large Cook's distance (all should be < 0.5 and inside the dotted red lines). Cook's distance is a measure of the influence of a data point, typically these high influence data points are those with large residuals and/or high leverage.

```
plot(mod.lm, 5)
```

Next, we will show examples where data show violations in plots 3 and 4. We will revisit our nonlinear and non normal data that we created before.

```
# a non linear relationship violates this diagnostic plot
plot(lm(response.nonlinear~variable, dat), 3)
```

```
# a non linear relationship violates this diagnostic plot
plot(lm(response.nonlinear~variable, dat), 5)
```

```
# a non normal relationship has points with large Cook's distance
plot(lm(response.notnormal~variable, dat), 5)
```

Summarise model results

Ok now that we know our data meet the assumptions of the model, we can look at the results. We will use summary and anova to extract the results of this model.

summary vs. anova

summary() and anova() test two different hypotheses for linear models, summary() (using *t*-test statistics) tests if a given model coefficient $\beta = 0$ while anova() (using ***F***-test) tests if a given variable reduces the residual sum of squares significantly. For a linear model with one variable, these two outputs yield identical results (p values), but this won't be true for more complicated models.

```
mod.lm # calling just the object shows model formula and coefficients
summary(mod.lm)
anova(mod.lm)
```

lm vs. aov

Alternatively, we could have used the aov() function to fit our model. This function, however, would only allow us to test the hypothesis that the added variable reduces the residual sum of squares (***F***-test, analysis of variance), regardless of using summary() or anova().

```
mod.aov <- aov(response~variable, dat)
summary(mod.aov)
anova(mod.lm)
```

broom package

We can further inspect our model using the *broom* package, which summarise model objects into a dataframe, which are easier to use for manipulating and plotting. This package has 3 main functions: glance(), which shows model statistics in one row, tidy(), which shows the model coefficients in a tidy summary, and augment(), which provides additional information on the fitted values and residuals. glance() and tidy() are also useful for creating model summary tables. Note that glance provides the ***F***-test statistics, while tidy provides the t-test statistics. However, we can produce a tidy sum of squares table using tidy() with anova().

```
library(broom)
library(knitr)
glance(mod.lm)
kable(tidy(mod.lm), digits=5)

kable(tidy(anova(mod.lm)), digits=5)
head(augment(mod.lm))
```