Lesson 7: Modelling the data

Randall Julian, Adam Zabell

1/2/2018

Data Modelling

After enough measurements have been made to consider your data a dataset, the next step is exploring that dataset to find trends and outliers. Most often, this includes a graph as well as a computational model which fits the data to a trendline. Modelling reduces the complexity of individual observations, which is good. However, it can only operate within the limits of the model. The linear model shown below doesn't fit as well as the quadratic model, but deciding whether the quadratic model makes physical sense is not something the model can demonstrate.

```
# data taken from http://www.theanalysisfactor.com/r-tutorial-4/
y <- c(126.6, 101.8, 71.6, 101.6, 68.1, 62.9, 45.5, 41.9, 46.3, 34.1, 38.2, 41.7, 24.7, 41.5, 36.6, 19.
x <- c(0, 1, 2, 4, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30)
x2 <- x ** 2
raw <- tibble(x, y, x2)
linearModel <- lm(y ~ x, data = raw)
quadraticModel <- lm(y ~ x + x2, data = raw)
plot(x, y)
sequenceX <- seq(0, 30, by = 0.1)
modeldata <- data.frame(x = sequenceX, x2 = sequenceX ** 2)
sequenceYL <- predict(linearModel, modeldata)
sequenceYQ <- predict(quadraticModel, modeldata)
lines(sequenceX, sequenceYL, col = "blue", lwd = 2)
lines(sequenceX, sequenceYQ, col = "darkgreen", lwd = 2)
```

> Exercise 1: Rebuild this plot using ggplot
> Partially complete commands are commented out in the following code chunk. Start by looking at the data.frame `modeldata` before and after the *mutate* command, to see what changed. Then, use these new columns as values for the two geoms.

```
modeldata <- modeldata %>%
  mutate(linear = sequenceYL, quadratic = sequenceYQ)

g <- ggplot(data = raw, aes(x = x, y = y)) + geom_point()
# g +
#  geom_point(data= ,aes(x=x,y= ),color= ) +
#  geom_point( , , )
```

Parsing dates and times with *lubridate*

The context for most of the data we'll encounter involve a time series, where each batch is collected at a specific date and that order of collection is a necessary component of the analysis. Dates within **R** are interpreted using the IEEE "POSIX" format, but reading data from an exterior source such as a CSV means this information arrives as a string. Functions in the **lubridate** package like `ymd` help to convert strings to POSIX, or combine information from multiple columns of a data.frame into a usable date format.

```
stringOfDates <- c("2017-12-15", "2017-Dec-16", "2017-12-17", "2017-12-18", "2017-12-19", "2017-12-20",
stringOfDates + 7
```

```r
oneWeek <- ymd(stringOfDates)
oneWeek + 7
tableOfDates <- tibble(
  year = rep(2017, 8),
  month = c(rep(4, 3), rep(5, 5)),
  day = seq(1, 8),
  hour = rep(4, 8),
  minute = c(15, 29, 40, 45, 0, 58, 9, 11)
)
tableOfDates %$% make_date(year, month, day) # uses the magrittr 'not a pipe' operator
tableOfDates %$% make_datetime(year, month, day, hour, minute)
```

It helps to remember R uses "yyyy-mm-dd" as the default order for dates when viewing or exporting.

Symbolic formula notation

The difficulty in using R for scientific modelling is that the language was built for statistical modelling. This is most obvious in the Wilkinson-Rogers notation for formula design. It is a versitile and powerful way to quickly describe the interactions between variables, but lacks an intuitive way to express basic transformations (quadratic, logrithmic, etc)

Formula Notation	Common Understanding
y ~ x	$y = a0 + a1x$
y ~ x - 1	$y = a1x$
y ~ x + z	$y = a0 + a1x + a2z$
y ~ x * z	$y = a0 + a1x + a2z + a3xz$
y ~ x + I(x^2)	$y = a0 + a1x + a2x^2$
y ~ Gauss(amplitude,mu,sigma,x)	fit the data to a formula, defined elsewhere

Wilkinson-Rogers notation comes into it's own when looking for generalized interactions (and their relative importance) between the dependent variable and a collection of independent ones: the formula $y \sim Height * Weight * Age$ would solve for eight coefficients, the linear combination for each of the three terms and the intercept. By comparison, the formula $y \sim Height + Weight + Age$ would solve four coefficients, ignoring any interaction between Height, Weight, and Age. We'll draw a plot in the next section which will hopefully help this to make more sense.

Linear modelling

Most of time a linear model, or transforming the data into a linear model, is sufficient to establish the key relationships between variables. Let's start by expanding the `raw` data.frame with additional observations and two more descriptors for the data.

```r
newraw <- raw %>%
  mutate(y = raw$y * rnorm(nrow(raw), mean = 1, sd = 0.3)) %>%
  rbind(raw) %>%
  arrange(x)
newraw$instrument <- c(rep("Coarse", 12), rep("Fine", 40))
newraw$date <- c("07-08-2017", "10-08-2017") %>%
  rep(times = 26) %>%
  dmy()
```

```
# this could also have been done on a single line without pipes:
# newraw$date <- dmy( rep(c("07-08-2017","10-08-2017"), times=26) )
```

Now we can explore the simple linear model in context of the instrument or the date of collection. Using the `summary` command on each linear model will provide basic statistical results assocaited with that model.

```
fullModel <- lm(y ~ x * date * instrument, data = newraw)
interactingTerms <- lm(y ~ x * instrument, data = newraw)
noninteractingTerms <- lm(y ~ x + instrument, data = newraw)
summary(interactingTerms)
```

We can now bulid the `grid` data.frame for plotting both the data and the models. Breaking down the pipe sequence of commands, we begin with our `newraw` data set and restructure it to work as an evenly spaced grid of points using *data_grid*, and then use *gather_predictions* to attach the prediction (y-value) at each obsevation point (x-value) from two of our linear models.

```
grid <- newraw %>%
  data_grid(x, date, instrument) %>%
  gather_predictions(interactingTerms, noninteractingTerms)
ggplot(newraw, aes(x, y, shape = factor(date))) +
  geom_point() +
  geom_line(data = grid, aes(y = pred, color = model)) +
  coord_cartesian(ylim = range(y)) +
  facet_wrap(~ instrument)
```

> Exercise 2: Modelling with categorical variables
> Revise `newraw` so that the last 40 observations include a third instrument called "Ultrafine," and rerun the models. How does this affect the regression and your interpretation?

```
newraw$instrument[13:52] <- rep(c(rep("Fine", 2), rep("Ultrafine", 2)), 10)
```

Nonlinear modelling

Although not common when looking for relationships between variables, nonlinear models are a concern for other aspects of data evaluation (e.g. fitting a chromatogram to a Gaussian curve) or assay validation (e.g. instrument response as a function of concentration). Unlike linear modelling, which will find a single solution starting from the data itself, a nonlinear model must iterate from a starting guess for each variable. A well formed guess should converge on the correct value, but there is no general method for finding that guess.

Below are three nonlinear fits to noisy Gaussian data, each starting from a different initial guess of the parameters. Because the noise is randomly applied, your graph shouldn't be an exact match to the plot made by anybody else. Not all the fits are going to look great!

```
Gauss <- function(amplitude, mu, sigma, x) {
  amplitude * exp(-0.5 * ((x - mu) / sigma) ^ 2)
}
x <- seq(1, 5, by = 0.1)
yExact <- Gauss(10000, 3, 0.4, x)
yNoisy <- yExact * rnorm(length(yExact), mean = 1, sd = 0.125)
noisyCurve <- tibble(x = x, y = yNoisy)
perfectGuess <- nls(
  y ~ Gauss(a, m, s, x),
  start = list(a = 10000, m = 3, s = 0.4),
  data = noisyCurve
```

```
)
okayGuess <- nls(
  y ~ Gauss(a, m, s, x),
  start = list(a = max(yNoisy), m = x[yNoisy == max(yNoisy)], s = 1),
  data = noisyCurve
)
badGuess <- nls(
  y ~ Gauss(a, m, s, x),
  start = list(a = 1, m = 1, s = 1),
  data = noisyCurve
)
forPlot <- noisyCurve %>%
  rename(yNoisy=y) %>%
  cbind(yExact) %>%
  gather(yType, y, -x)
grid <- forPlot %>%
  data_grid(x = seq_range(x, 100), y) %>%
  gather_predictions(perfectGuess, okayGuess)
ggplot(forPlot, aes(x, y)) +
  geom_point(aes(shape = yType)) +
  scale_shape_manual(values = c(3, 20), name = "data source") +
  geom_line(data = grid, aes(y = pred, color = model, linetype = model), size = 1)
summary(perfectGuess)
```

Understanding the reasoning for nonlinear model selection is a critical step before attempting that fit. The risk of non-convergence means the nonlinear model should be avoided, if possible.

Summary

- **lubridate** functions will convert a text string to a date
- statistial modelling with Wilkinson-Rogers notation builds possible interactions into the analysis
- scientific modelling with Wilkinson-Rogers notation requires careful understanding of terms
- linear models are strongly recommended whenever possible
  - this may require a transformation of the data
  - 'linear' doesn't mean 'straight line' when you draw the plot
- nonlinear models are dependant on a starting guess