Formulas and ANOVA

Nicholas A. Del Grosso

July 24, 2018

Formulas

So far, our strategy for analysis has been:

1. Load the Dataset as a dataframe.
2. Transform the data, calculating new columns as we work.
3. Extract vectors from the dataframe for each subset of the data we want to analyze.
4. Give the vectors as arguments to statistics functions.

This has worked out just fine, but it produces quite a bit of code in the process. As our analysis gets more complex and we want to explore our data further, we can become bogged down in the amount of steps required. Luckily, R has some built-in approaches for describing common operations in the data analysis pipeline, including describing the statistical model you'd like to test. To describe a statistical model, we'll use R's **formula** syntax.

Formulas have a left side and a right side; the left side is the output (e.g. the dependent variable of an experiment), and the right side is the input to the model (e.g. the dependent variables of an experiment). Whereas mathematical notation would use the equal sign, R uses the tilda (~) symbol to seperate the left and right sides in order to specifically indicate that the line is a formula.

For our PlantGrowth dataset, for example, the experiment can be described with the following formula:

```
weight ~ group
```

This formula can be assigned to its own variable, in order to be used again later:

```
model <- weight ~ group
```

By default, R includes an additional intercept value into the model, thereby making the above formula equivalent to the linear model

$$y = mX + b$$

This intercept can be made explicit by adding "+ 1" to the R formula. Writing "+ 0" removes the intercept, in case you don't want it:

```
weight ~ group + 1
```

More variablees can be included as well. For example, if there was also an "age" variable, a 2-factor model describing independent contributions of age and group on the weight would written as:

```
weight ~ group + age
```

Including interactions between the two variables can be written in multiple ways:

```
weight ~ group * age
weight ~ group + age + group:age
```

This is equivalent to the following linear model:

$$y = m_1 x_1 + m_2 x_2 + m_3 (x_1 * x_2) + b$$

Using Formulas in Analysis

Many statistical functions can take formulas as arguments and apply it to the data. These functions all have a common interface:

```
result <- stats_function(formula = ____, data = ____)
```

You can additionally perform subsetting in these functions, saving you additional steps of selecting your data:

```
result <- stats_function(formula = ____, data = ____, subset = _____)
```

We'll look now at how these are used for t-tests (*t.test()*), linear models (*lm()*), and ANOVA (*aov()*) analyses.

Formulas in t-tests

To look at how group decides the weight of the plants in our PlantGrowth dataset, we select the 'trt2' and 'ctrl' groups and pass the corresponding model and dataset to the t.test() function, all in one step:

```
t.test(formula = weight ~ group, data=PlantGrowth, subset=group %in% c('trt2', 'ctrl'))
```

```
##
##  Welch Two Sample t-test
##
## data:  weight by group
## t = -2.134, df = 16.786, p-value = 0.0479
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.98287213 -0.00512787
## sample estimates:
## mean in group ctrl mean in group trt2
##              5.032              5.526
```

Formulas in ANOVA

The same approach works for ANOVA analysis. This time, we'll look at all 3 groups at once, removing the need to subset at all:

```
aov(formula = weight ~ group, data=PlantGrowth)
```

```
## Call:
##    aov(formula = weight ~ group, data = PlantGrowth)
##
## Terms:
##                     group Residuals
## Sum of Squares   3.76634  10.49209
## Deg. of Freedom        2        27
##
## Residual standard error: 0.6233746
## Estimated effects may be unbalanced
```

Notice that **aov()** is not returning the full table, nor the p-value. This happens with several models in R; the solution is to pass your results to the **summary()** function, which reveals everything:

```
model <- aov(formula = weight ~ group, data=PlantGrowth)
summary(model)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## group        2  3.766  1.8832   4.846 0.0159 *
## Residuals   27 10.492  0.3886
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If you'd like to extract values from this table, the **unlist()** function unwraps the table into a vector, which you can index as usual:

```
results <- unlist(summary(model))
results
```

```
##         Df1         Df2     Sum Sq1     Sum Sq2    Mean Sq1    Mean Sq2
##  2.00000000 27.00000000  3.76634000 10.49209000  1.88317000  0.38859593
##    F value1    F value2     Pr(>F)1     Pr(>F)2
##  4.84608786          NA  0.01590996          NA
```

```
results[['Pr(>F)1']]
```

```
## [1] 0.01590996
```

Linear Models and General Linear Models

Linear models have the same formula-data-subset interface as aov() and can be used with the **lm()** and **glm()** functions.

Of special note is the glm() function's **family** optional argument, which lets you specify what kind of distribution is being fit to the data. Logistic, poisson, and many other distributions can be set and modified. For example, logistic regression can be done as so:

```
PlantGrowth$is.heavy <- PlantGrowth$weight > 5
model <- glm(formula = is.heavy ~ group, data = PlantGrowth, family = binomial(link = 'logit'))
summary(model)
```

```
##
## Call:
## glm(formula = is.heavy ~ group, family = binomial(link = "logit"),
##     data = PlantGrowth)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1460  -0.6681   0.4590   0.8728   1.7941
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.4055     0.6455   0.628   0.5299
## grouptrt1    -1.7918     1.0206  -1.756   0.0792 .
## grouptrt2     1.7918     1.2360   1.450   0.1471
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 41.054  on 29  degrees of freedom
## Residual deviance: 29.970  on 27  degrees of freedom
## AIC: 35.97
##
```

```
## Number of Fisher Scoring iterations: 4
```

Model Exploration

Once a model has been calculated, you can pass that model as an argument to several functions:

- **fitted(model)**: get the model's predictions to the data it was originally fit to.
- **predict(model, new_data = _____)**: Give the same model more data and see how well it predicts those values.
- **resid(model)** Get the model's error (i.e. the model's "residuals"") for each row–useful for checking the core assumptions of the model.

Further Reading

- I found Will Lowe's blog article on formulas very helpful: http://conjugateprior.org/2013/01/formulae-in-r-anova/
- More detailed models and model exploration, including use of the *lattice* package, here: http://garrettgman.github.io/model-fitting/