Fitting Models in R Statistics 506

Statistical Models

One downside of R's user-driven development via the package system is that there is no enforced uniformity in terms of implementation. This is especially true of statistical models, as different packages can implement different models (or even the same models) in different ways. While the estimated parameters are usually the same (up to numerical precision), implementation details can differ widely, including:

- How the model fitting code looks
- What the output of the model reports
- How the software returns the model fitting artifacts

That said, for a lot of the most common models, there is some uniformity across these, so we'll cover that here.

Formulas

A `formula` is an R object that stores an equation:

```
(a <- 3 ~ 5 - 2)
```

```
3 ~ 5 - 2
```

The `~` is used in place of an `=`.

```
class(a)
```

```
[1] "formula"
```

```r
typeof(a)
```

```
[1] "language"
```

Often objects in R that aren't lists or vectors have type `language`.

More commonly, formulas are used to store a equation involving variables.

```r
form <- Fertility ~ Education + Catholic + Infant.Mortality
form
```

```
Fertility ~ Education + Catholic + Infant.Mortality
```

```r
data(swiss)
names(swiss)
```

```
[1] "Fertility"        "Agriculture"      "Examination"      "Education"
[5] "Catholic"         "Infant.Mortality"
```

Note that I loaded `swiss` *after* defining the formulas - the "variables" in a formula need not exist or be "real" until the point at which the formula evaluated to access the data. This is a variation of lazy loading.

When used in this fashion, the left hand side of the formula indicates the response (outcome/dependent) variable(s) in the model, whereas the right hand side of the formula indicates the predictor (covariate/independent) variable(s) in the model. So in `form` above, "Fertility" is the outcome and "Education", "Catholic" and "Infant.Mortality" are the predictors.

Interactions can be included by separating variables by `:` or `*` instead of `+`. `:` includes only the interaction, `*` also includes all lower-order terms. These two formulas would yield the same model in most cases:

```r
f1 <- a ~ b*c
f2 <- a ~ b + c + b:c
```

(We will discuss including polynomial terms after discussing fitting a model, below).

Terms can be removed with `-`

```r
y ~ x*z - x
y ~ z + x:z # Equivalent formulas
```

Adding 0 or subtracting 1 suppresses an intercept:

```
y ~ x + 0
y ~ x - 1
```

Fitting a linear regression model.

The `lm` function takes in, at a minimum, a formula and a data set.

```
mod1 <- lm(form, data = swiss)
mod1
```

```

Call:
lm(formula = form, data = swiss)

Coefficients:
    (Intercept)          Education           Catholic  Infant.Mortality
       48.67707           -0.75925            0.09607           1.29615
```

```
mod2 <- lm(Fertility ~ Education + Catholic*Infant.Mortality, data = swiss)
mod2
```

```

Call:
lm(formula = Fertility ~ Education + Catholic * Infant.Mortality,
    data = swiss)

Coefficients:
              (Intercept)                  Education
               48.9995699                 -0.7594599
                 Catholic           Infant.Mortality
                0.0890711                  1.2797901
Catholic:Infant.Mortality
                0.0003493
```

Passing a model output into the `summary` function typically produces far more useful information.

```
summary(mod1)
```

```
Call:
lm(formula = form, data = swiss)

Residuals:
     Min       1Q   Median       3Q      Max 
-14.4781  -5.4403  -0.5143   4.1568  15.1187 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)      48.67707    7.91908   6.147 2.24e-07 ***
Education        -0.75925    0.11680  -6.501 6.83e-08 ***
Catholic          0.09607    0.02722   3.530  0.00101 ** 
Infant.Mortality  1.29615    0.38699   3.349  0.00169 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.505 on 43 degrees of freedom
Multiple R-squared:  0.6625,	Adjusted R-squared:  0.639 
F-statistic: 28.14 on 3 and 43 DF,  p-value: 3.15e-10
```

```r
summary(mod2)
```

```
Call:
lm(formula = Fertility ~ Education + Catholic * Infant.Mortality, 
    data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max 
-14.464  -5.446  -0.467   4.152  15.193 

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)    
(Intercept)               48.9995699 11.4043460   4.297 0.000101 ***
Education                 -0.7594599  0.1183005  -6.420 9.88e-08 ***
Catholic                   0.0890711  0.1781610   0.500 0.619722    
Infant.Mortality           1.2797901  0.5681168   2.253 0.029563 *  
Catholic:Infant.Mortality  0.0003493  0.0087891   0.040 0.968489    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 7.594 on 42 degrees of freedom
Multiple R-squared:  0.6626,	Adjusted R-squared:  0.6304
F-statistic: 20.62 on 4 and 42 DF,  p-value: 1.844e-09
```

Refer to any introductory modeling notes for a discussion of the interpretation of the various parts of the output.

Models as R objects

We will dive much deeper into R's class system (S3 and S4) at a later point, but for now it is sufficient to understand that most model objects in R are lists with special `print` functions that make the output clear. (I'm not going to demonstrate in this notes to save space, but trying fitting a model (`mod <- lm(...)`), then changing the class to `list` (`class(mod) <- "list"`) before printing it (`mod`). We can look at the pieces of the list as well:

```
typeof(mod1)
```

```
[1] "list"
```

```
class(mod1)
```

```
[1] "lm"
```

```
names(mod1)
```

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

```
mod1$coefficients
```

```
    (Intercept)       Education        Catholic Infant.Mortality
    48.67707330     -0.75924577      0.09606607       1.29614813
```

```
head(mod1$residuals)
```

```
  Courtelary     Delemont Franches-Mnt      Moutier   Neuveville    Porrentruy 
   10.902569     4.331405    12.464392    12.881689    12.415261    -10.440597 
```

The object produced by `summary` is similar:

```r
smod1 <- summary(mod1)
typeof(smod1)
```

```
[1] "list"
```

```r
class(smod1)
```

```
[1] "summary.lm"
```

```r
names(smod1)
```

```
 [1] "call"          "terms"         "residuals"     "coefficients" 
 [5] "aliased"       "sigma"         "df"            "r.squared"    
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled" 
```

```r
smod1$r.squared
```

```
[1] 0.6625438
```

```r
smod1$coefficients
```

```
                   Estimate Std. Error   t value     Pr(>|t|)
(Intercept)      48.67707330 7.91908348  6.146806 2.235983e-07
Education        -0.75924577 0.11679763 -6.500524 6.833658e-08
Catholic          0.09606607 0.02721795  3.529511 1.006201e-03
Infant.Mortality  1.29614813 0.38698777  3.349326 1.693753e-03
```

```r
smod1$cov.unscaled
```

```
                  (Intercept)     Education      Catholic Infant.Mortality
(Intercept)       1.113269e+00 -4.171717e-03 -1.974047e-05    -5.241958e-02
Education        -4.171717e-03  2.421689e-04  7.859415e-06     5.965361e-05
Catholic         -1.974047e-05  7.859415e-06  1.315107e-05    -3.046909e-05
Infant.Mortality -5.241958e-02  5.965361e-05 -3.046909e-05     2.658550e-03
```

Polynomial terms

Including polynomial terms in R models is slightly non-trivial (compared to how trivial it is in Stata, which we'll see in the future). There are (at least) 3 different ways to do it, each with their own pros and cons.

Manually including polynomial terms

```
swiss$Infant.Mortality2 <- swiss$Infant.Mortality^2
mod3 <- lm(Fertility ~ Infant.Mortality + Infant.Mortality2, data = swiss)
summary(mod3)
```

```
Call:
lm(formula = Fertility ~ Infant.Mortality + Infant.Mortality2,
    data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max
-31.245  -5.358  -0.030   7.120  28.474

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       59.00214   46.24197   1.276    0.209
Infant.Mortality  -0.78971    4.74020  -0.167    0.868
Infant.Mortality2  0.06623    0.12093   0.548    0.587

Residual standard error: 11.57 on 44 degrees of freedom
Multiple R-squared:  0.1791,    Adjusted R-squared:  0.1418
F-statistic:   4.8 on 2 and 44 DF,  p-value: 0.01301
```

"Inhibit interpretation" of polynomial terms.

A polynomial term is nothing more than an interaction (multiplication) of a variable with itself. What would happen if we just tried that?

```
mod4 <- lm(Fertility ~ Infant.Mortality*Infant.Mortality, data = swiss)
summary(mod4)
```

```
Call:
```

```
lm(formula = Fertility ~ Infant.Mortality * Infant.Mortality,
    data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max 
-31.672  -5.687  -0.381   7.239  28.565 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)   
(Intercept)       34.5155    11.7113   2.947  0.00507 **
Infant.Mortality   1.7865     0.5812   3.074  0.00359 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.48 on 45 degrees of freedom
Multiple R-squared:  0.1735,    Adjusted R-squared:  0.1552 
F-statistic: 9.448 on 1 and 45 DF,  p-value: 0.003585
```

R basically ignored it. We can use the `I()` function to prevent R from trying to over-interpret the results:

```r
mod5 <- lm(Fertility ~ Infant.Mortality + I(Infant.Mortality*Infant.Mortality),
           data = swiss)
summary(mod5)
```

```
Call:
lm(formula = Fertility ~ Infant.Mortality + I(Infant.Mortality *
    Infant.Mortality), data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max 
-31.245  -5.358  -0.030   7.120  28.474 

Coefficients:
                                      Estimate Std. Error t value Pr(>|t|)
(Intercept)                           59.00214   46.24197   1.276    0.209
Infant.Mortality                      -0.78971    4.74020  -0.167    0.868
I(Infant.Mortality * Infant.Mortality)  0.06623    0.12093   0.548    0.587

Residual standard error: 11.57 on 44 degrees of freedom
Multiple R-squared:  0.1791,    Adjusted R-squared:  0.1418 
```

```
F-statistic:   4.8 on 2 and 44 DF,  p-value: 0.01301
```

Formally what `I()` is doing is to tell R to not interpret any algebraic symbols (`+`, `-`, `*`, etc) as formula operators, and to instead treat them purely as algebraic.

`poly` function

Finally, the most concise way to write a model with polynomial terms is the `poly`function:

```r
mod6 <- lm(Fertility ~ poly(Infant.Mortality, 2), data = swiss)
summary(mod6)
```

```

Call:
lm(formula = Fertility ~ poly(Infant.Mortality, 2), data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max 
-31.245  -5.358  -0.030   7.120  28.474 

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)    
(Intercept)                  70.143      1.688  41.554  < 2e-16 ***
poly(Infant.Mortality, 2)1   35.292     11.572   3.050  0.00387 ** 
poly(Infant.Mortality, 2)2    6.338     11.572   0.548  0.58668    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.57 on 44 degrees of freedom
Multiple R-squared:  0.1791,	Adjusted R-squared:  0.1418 
F-statistic:   4.8 on 2 and 44 DF,  p-value: 0.01301
```

By default, `poly` will produce orthogonal polynomial terms - these do **not** change the model fit (note that the R^2 is identical), but do change the interpretation of the coefficients. The `raw = TRUE` option suppresses this:

```r
mod7 <- lm(Fertility ~ poly(Infant.Mortality, 2, raw = TRUE), data = swiss)
summary(mod7)
```

```
Call:
lm(formula = Fertility ~ poly(Infant.Mortality, 2, raw = TRUE),
    data = swiss)

Residuals:
    Min      1Q  Median      3Q     Max
-31.245  -5.358  -0.030   7.120  28.474

Coefficients:
                                       Estimate Std. Error t value Pr(>|t|)
(Intercept)                            59.00214   46.24197   1.276    0.209
poly(Infant.Mortality, 2, raw = TRUE)1 -0.78971    4.74020  -0.167    0.868
poly(Infant.Mortality, 2, raw = TRUE)2  0.06623    0.12093   0.548    0.587

Residual standard error: 11.57 on 44 degrees of freedom
Multiple R-squared:  0.1791,	Adjusted R-squared:  0.1418
F-statistic:   4.8 on 2 and 44 DF,  p-value: 0.01301
```

The reason to include the orthogonal polynomials is computation - non-linear models which require convergence of an optimization problem can struggle when including very large or very small numbers, or two variables on very different scales. Standardizing and orthogonalizing can help address this.

Pros and cons of each approach

Manual pros:

- Easy to implement
- Easy to exclude lower order polynomials.
- Produces the nicest looking output

Manual cons:

- Need to remember to update if values change
- R doesn't know the terms are related
- Clutters your data

`I()` Pros:

- Easy to exclude lower order polynomials
- Precise control over what you're including in the models
- R will know terms are related

`I()` Cons:

- Longest syntax
- `I(x)` notation makes output less readable

`poly()` pros:

- Most concise syntax
- R will know terms are related

`poly()` cons:

- Hard to exclude lower order polynomials

Overall, I recommend using `poly()` in almost all situations (with or without `raw = TRUE`), dropping down to `I()` only if more precise control is needed.

Model extractors

There are a few functions that most well-written model objects should support to extract useful model artifacts.

```
head(predict(mod1)) # predicted values
```

```
  Courtelary     Delemont Franches-Mnt      Moutier   Neuveville   Porrentruy
    69.29743     78.76860     80.03561     72.91831     64.48474     86.54060
```

```
head(residuals(mod1)) # residual values
```

```
  Courtelary     Delemont Franches-Mnt      Moutier   Neuveville   Porrentruy
   10.902569     4.331405    12.464392    12.881689    12.415261   -10.440597
```

```
coefficients(mod1) # coefficients
```

```
     (Intercept)        Education         Catholic Infant.Mortality
     48.67707330      -0.75924577       0.09606607       1.29614813
```

While in the `lm` case, each of these could be extracted directly (`mod1$fitted`, `mod1$residuals`, `mod1$coefficients`), in other models, it may not be as straightforward so these functions come in handy.

Summary objects may not support all these functions:

```r
head(predict(smod1)) # predicted values
```

```
Error in UseMethod("predict"): no applicable method for 'predict' applied to an object of cla
```

```r
head(residuals(smod1)) # residual values
```

```
  Courtelary     Delemont Franches-Mnt      Moutier   Neuveville   Porrentruy
   10.902569     4.331405    12.464392    12.881689    12.415261   -10.440597
```

```r
coefficients(smod1) # coefficients
```

```
                   Estimate Std. Error   t value     Pr(>|t|)
(Intercept)      48.67707330 7.91908348  6.146806 2.235983e-07
Education        -0.75924577 0.11679763 -6.500524 6.833658e-08
Catholic          0.09606607 0.02721795  3.529511 1.006201e-03
Infant.Mortality  1.29614813 0.38698777  3.349326 1.693753e-03
```

Design matrices

Recall when fitting a linear regression model, the estimated coefficients can be calculated as

$$\hat{\beta} = (X'X)^{-1}X'y$$

where y is the vector of outcomes, and X is an $n \times p$ matrix of predictors, where there are n observations and p predictors. X is often called the "design matrix" and includes one column for every variable in the model, including the intercept.

The `model.matrix` functions can be used to automatically generate this matrix.

```r
head(model.matrix(form, data = swiss))
```

```
             (Intercept) Education Catholic Infant.Mortality
Courtelary             1        12     9.96             22.2
Delemont               1         9    84.84             22.2
Franches-Mnt           1         5    93.40             20.2
Moutier                1         7    33.77             20.3
Neuveville             1        15     5.16             20.6
Porrentruy             1         7    90.57             26.6
```

This call is agnostic of the model - it only required the formula. Passing a model instead often is preferred because if the model drops any observations, they will be dropped from the output as well.

```
head(model.matrix(mod1, data = swiss))
```

```
             (Intercept) Education Catholic Infant.Mortality
Courtelary             1        12     9.96             22.2
Delemont               1         9    84.84             22.2
Franches-Mnt           1         5    93.40             20.2
Moutier                1         7    33.77             20.3
Neuveville             1        15     5.16             20.6
Porrentruy             1         7    90.57             26.6
```

In addition, in the presence of interactions or categorical variables, `model.matrix` will expand these out as appropriate:

```
data(mtcars)
mtcars$gear <- as.factor(mtcars$gear)
head(model.matrix(mpg ~ gear + cyl*wt, data = mtcars))
```

```
                  (Intercept) gear4 gear5 cyl    wt cyl:wt
Mazda RX4                   1     1     0   6 2.620  15.72
Mazda RX4 Wag               1     1     0   6 2.875  17.25
Datsun 710                  1     1     0   4 2.320   9.28
Hornet 4 Drive              1     0     0   6 3.215  19.29
Hornet Sportabout           1     0     0   8 3.440  27.52
Valiant                     1     0     0   6 3.460  20.76
```

Note the use of `as.factor` to specify that "gear" should be treated as categorical.

Proving the equivalence of `:` and `*`, and demonstrating `-`:

```
head(model.matrix(mpg ~ cyl*wt, data = mtcars))
```

```
                  (Intercept) cyl    wt cyl:wt
Mazda RX4                   1   6 2.620  15.72
Mazda RX4 Wag               1   6 2.875  17.25
Datsun 710                  1   4 2.320   9.28
Hornet 4 Drive              1   6 3.215  19.29
```

```
Hornet Sportabout           1   8 3.440  27.52
Valiant                     1   6 3.460  20.76
```

```r
head(model.matrix(mpg ~ cyl + wt + cyl:wt, data = mtcars))
```

```
                  (Intercept) cyl    wt cyl:wt
Mazda RX4                   1   6 2.620  15.72
Mazda RX4 Wag               1   6 2.875  17.25
Datsun 710                  1   4 2.320   9.28
Hornet 4 Drive              1   6 3.215  19.29
Hornet Sportabout           1   8 3.440  27.52
Valiant                     1   6 3.460  20.76
```

```r
head(model.matrix(mpg ~ cyl*wt - wt, data = mtcars))
```

```
                  (Intercept) cyl cyl:wt
Mazda RX4                   1   6  15.72
Mazda RX4 Wag               1   6  17.25
Datsun 710                  1   4   9.28
Hornet 4 Drive              1   6  19.29
Hornet Sportabout           1   8  27.52
Valiant                     1   6  20.76
```

There is a similar function, `model.frame` which does not do any expansion but merely includes all variables involved in the model, including the outcome:

```r
head(model.frame(mpg ~ cyl*wt + gear, data = mtcars))
```

```
                   mpg cyl    wt gear
Mazda RX4         21.0   6 2.620    4
Mazda RX4 Wag     21.0   6 2.875    4
Datsun 710        22.8   4 2.320    4
Hornet 4 Drive    21.4   6 3.215    3
Hornet Sportabout 18.7   8 3.440    3
Valiant           18.1   6 3.460    3
```

Model post-estimation

After fitting a statistical model, you may want to test various hypotheses that involve linear (or non-linear combinations of coefficients). There are a number of different R packages that can do this, we'll discuss a few here.

Hypotheses tests between estimated coefficients

The `glht()` function from the *multcomp* package directly tests hypotheses. Let's load in a data-set which records information on husband and wive pairs from Great Britain (downloaded from https://www.openintro.org/data/index.php?data=husbands_wives).

```
hw <- read.csv("data/husbands_wives.csv")
head(hw)
```

```
  age_husband age_wife ht_husband ht_wife age_husb_at_marriage
1          49       43       1809    1590                   25
2          25       28       1841    1560                   19
3          40       30       1659    1620                   38
4          52       57       1779    1540                   26
5          58       52       1616    1420                   30
6          32       27       1695    1660                   23
  age_wife_at_marriage years_married
1                   19            24
2                   22             6
3                   28             2
4                   31            26
5                   24            28
6                   18             9
```

Let's fit a model predicting number of years married by the heights of the partners, and see whether there is a difference in the relationship between genders:

```
mod <- lm(years_married ~ ht_husband + ht_wife, data = hw)
library(multcomp)
glht(mod, "ht_husband - ht_wife = 0")
```

```
	 General Linear Hypotheses
```

```
Linear Hypotheses:
                           Estimate
ht_husband - ht_wife == 0 -0.003919
```

```
summary(glht(mod, "ht_husband - ht_wife = 0"))
```

```
	 Simultaneous Tests for General Linear Hypotheses

Fit: lm(formula = years_married ~ ht_husband + ht_wife, data = hw)

Linear Hypotheses:
                           Estimate Std. Error t value Pr(>|t|)
ht_husband - ht_wife == 0 -0.003919   0.020623   -0.19    0.849
(Adjusted p values reported -- single-step method)
```

The right hand side of the "equation" must be numeric, so instead of `ht_husband = ht_wife`, we move the variables to one side.

Another case where this would be useful is estimating the average value of a response across groups. The `mtcars` data contains a categorical variable, `gear`, indicating the number of forward gears (3, 4, or 5).

```
data(mtcars)
mtcars$gear <- as.factor(mtcars$gear)
(mod <- lm(mpg ~ gear, data = mtcars))
```

```
Call:
lm(formula = mpg ~ gear, data = mtcars)

Coefficients:
(Intercept)        gear4        gear5
     16.107        8.427        5.273
```

Recall from basic statistical modeling classes. We can estimate the average response within each level of `gear` via linear combination of predictors.

$$E(\text{mpg}|\text{gear}) = \beta_0 + \beta_1 * gear_4 + \beta_2 * gear_5$$

`gear` level	Equation	Estimate
3	β_0	16.11
4	$\beta_0 + \beta_1$	24.53
5	$\beta_0 + \beta_2$	21.38

```r
list(glht(mod, "(Intercept) = 0"),
     glht(mod, "(Intercept) + gear4 = 0"),
     glht(mod, "(Intercept) + gear5 = 0"))
```

```
[[1]]

	 General Linear Hypotheses

Linear Hypotheses:
                 Estimate
(Intercept) == 0    16.11


[[2]]

	 General Linear Hypotheses

Linear Hypotheses:
                         Estimate
(Intercept) + gear4 == 0    24.53


[[3]]

	 General Linear Hypotheses

Linear Hypotheses:
                         Estimate
(Intercept) + gear5 == 0    21.38
```

We can of course test for differences in these means. E.g., to test `gear` 4 vs `gear` 5:

$$
\begin{aligned}
\beta_0 + \beta_1 &= \beta_0 + \beta_2 \\
\beta_1 &= \beta_2 \\
\beta_1 - \beta_2 &= 0
\end{aligned}
$$

```r
summary(glht(mod, "gear4 - gear5 = 0"))
```

```

	 Simultaneous Tests for General Linear Hypotheses

Fit: lm(formula = mpg ~ gear, data = mtcars)

Linear Hypotheses:
                   Estimate Std. Error t value Pr(>|t|)
gear4 - gear5 == 0    3.153      2.506   1.258    0.218
(Adjusted p values reported -- single-step method)
```

Other packages

There are a number of packages which produce "marginal effects". This term can mean two things. The first is linear combinations of coefficients, just as we did above. The second is more formally on the idea of "marginalizing" over some coefficients in the model. We'll focus on the first definition here and demonstrate the *emmeans* package.

```r
library(emmeans)
emmeans(mod, "gear")
```

```
 gear emmean   SE df lower.CL upper.CL
 3      16.1 1.22 29     13.6     18.6
 4      24.5 1.36 29     21.8     27.3
 5      21.4 2.11 29     17.1     25.7

Confidence level used: 0.95 
```

```r
test(emmeans(mod, "gear"))
```

```
 gear emmean   SE df t.ratio p.value
 3      16.1 1.22 29  13.250  <.0001
 4      24.5 1.36 29  18.051  <.0001
 5      21.4 2.11 29  10.154  <.0001
```

```
pairs(emmeans(mod, "gear"))
```

```
 contrast      estimate   SE df t.ratio p.value
 gear3 - gear4    -8.43 1.82 29  -4.621  0.0002
 gear3 - gear5    -5.27 2.43 29  -2.169  0.0937
 gear4 - gear5     3.15 2.51 29   1.258  0.4296

P value adjustment: tukey method for comparing a family of 3 estimates
```

You can see we've replicated the results from above, but in much more precise code and without worrying about deriving the equations ourselves.

Some other packages that do similar things:

- *emmeans*: One of the oldest of these packages, extremely powerful but can be complicated to use for non-basic stuff.
- *marginaleffects*: A very new package that is quite slick, but is in early development and changes the API frequently still.
- *ggeffects*: Has less functionality than the other packages, but makes it very easy to plot results using GGplot2.

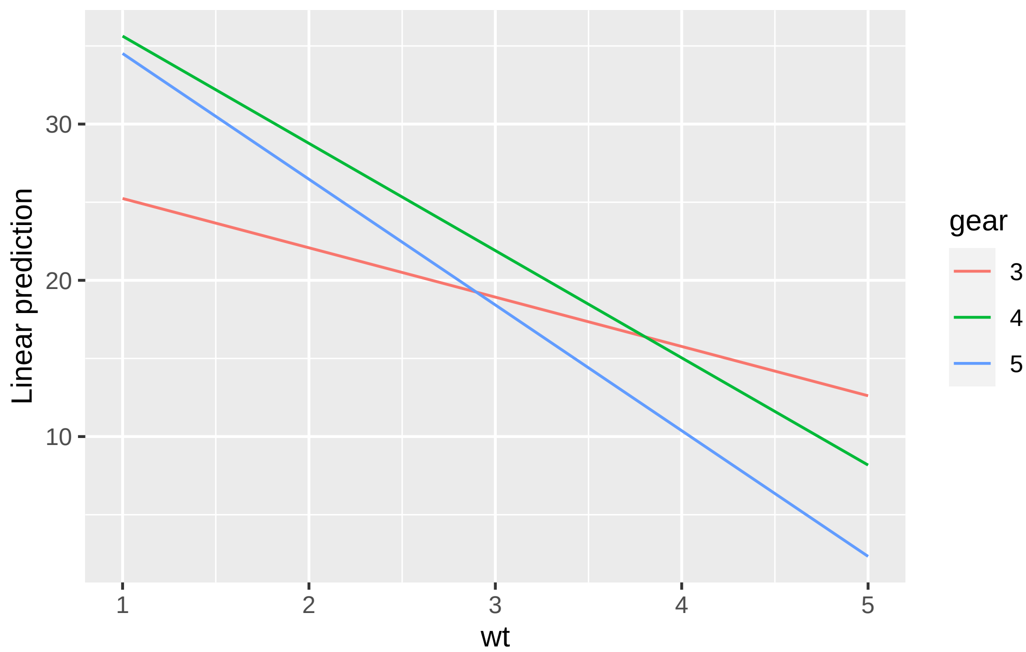
Interaction plots

Consider a model where we have a continuous variable, X, and a binary predictor, Z:

$$E(Y|X, Z) = \beta_0 + \beta_1 X + \beta_2 Z + \beta_3 XZ$$

By including an interaction, we are allowing each group as defined by Z to have its own slope on X. An interaction plot visualizes this relationship. We'll return to the *emmeans* package:

```
mod <- lm(mpg ~ gear*wt, data = mtcars)
emmip(mod, gear ~ wt, at = list(wt = 1:5))
```



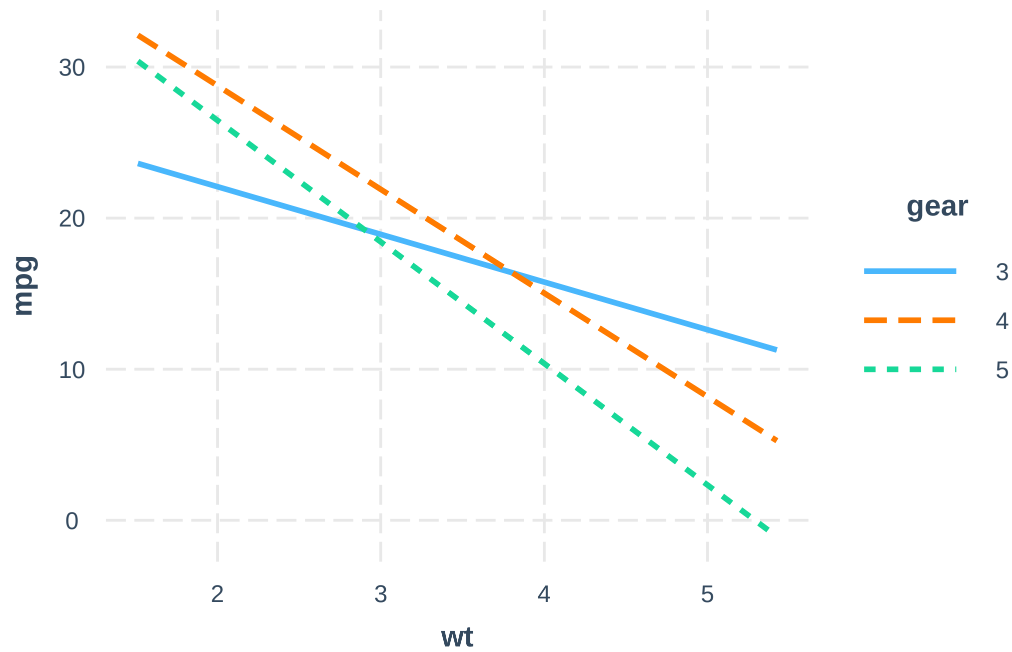
The values for `wt` are chosen by examining the variable:

```
summary(mtcars$wt)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.513   2.581   3.325   3.217   3.610   5.424 
```

Another package, *interactions*, can also easily produce these plots:

```
library(interactions)
interact_plot(mod, pred = wt, modx = gear)
```



`emmip` is more flexible and offers more functionality, whereas `interact_plot` is generally more straightforward for simple plots.

Generalized Linear Models

While `lm` fits linear models, `glm` fits generalized linear models:

```
glm(am ~ wt + disp, data = mtcars, family = binomial)
```

```
Call:  glm(formula = am ~ wt + disp, family = binomial, data = mtcars)

Coefficients:
(Intercept)           wt         disp  
   15.59942     -5.95982      0.01124  

Degrees of Freedom: 31 Total (i.e. Null);  29 Residual
Null Deviance:	    43.23 
Residual Deviance: 17.78 	AIC: 23.78
```

See `help(family)` for details on the various distributions and link functions supported. Generally, things carry forward from the linear model: how to specify the formula, extracting model artifacts, and various post-estimation functionality such as hypothesis tests and interaction plots.