Advanced Linear Modeling in R

Dillon Niederhut

November 11, 2015

Introduction

Motivation

Most of what you'll want to do in inferential statistics is either a linear model, or based on a linear model. At one end of the spectrum, t-tests, ANCOVA, and linear constrasts are highly restricted versions of lienar models that allow easy calculation. At the other end of the spectrum, machine learning algorithms like logistic classifiers, ridge regression, and lasso regression, are all based on the repeated use of calculating full linear models. When used correctly, the linear model is one of the best ways to perform inferential statistics. When used incorrectly, the linear model is highly susceptible to spurious results (that lead to retracted papers!).

Orientation

The linear model is one which assumes a deterministic world, where present observations are linear combinations of past observations.

Contrasts

Contrasts are useful for testing specific hypotheses

With an F-test, you can learn that a variable is causing a difference, but not what that difference is. You can think about contrasts as a way to look at the levels of a variable and ask which level, or groups of levels, are causing the difference. Maybe all of the levels are involved, but each one only a little bit. Contrasts allow you to test this possibility too.

The default contrast set for a factor is `contr.treatment`

This is the same thing as 'one-hot encoding'. R treats the first level of the variable as the basis for comparison, and then every subsequent level is represented as a binary 'dummy' variable.

```
contr.treatment(4)
```

```
##   2 3 4
## 1 0 0 0
## 2 1 0 0
## 3 0 1 0
## 4 0 0 1
```

If your data don't follow a treatment logic, you can try the sum or Helmert contrasts

```
contr.helmert(4)
```

```
##   [,1] [,2] [,3]
## 1   -1   -1   -1
## 2    1   -1   -1
## 3    0    2   -1
## 4    0    0    3
```

```
contr.sum(4)
```

```
##   [,1] [,2] [,3]
## 1    1    0    0
## 2    0    1    0
## 3    0    0    1
## 4   -1   -1   -1
```

If your data have inherent order, try the polynomial contrast set

```
contr.poly(4)
```

```
##              .L   .Q         .C
## [1,] -0.6708204  0.5 -0.2236068
## [2,] -0.2236068 -0.5  0.6708204
## [3,]  0.2236068 -0.5 -0.6708204
## [4,]  0.6708204  0.5  0.2236068
```

This produces linear and quadratic contrasts. It's a bit hard to see that though, right? This is because R is making all of the contrasts orthogonal for you. Non-orthogonal contrasts risk explaining the same bit of variance twice, and make you think that you are predicting more than you actually are.

Assigning contrasts to a factor is easy

The syntax looks like this:

```
my.factor = factor(c('Dillon', 'Andrew', 'Shinhye', 'Patty'))
contrasts(my.factor) <- contr.helmert
contrasts(my.factor)
```

```
##         [,1] [,2] [,3]
## Andrew    -1   -1   -1
## Dillon     1   -1   -1
## Patty      0    2   -1
## Shinhye    0    0    3
```

But you don't have to let R do it for you

The benefit to using R's built-ins is that they are guaranteed to be correct. But, you are all freethinking rebels (you are at Berkeley after all), so let's look at how to do this by hand.

```
my.contrast <- matrix(c(
  -2, -1, 1, 2,
  0, 0, 1, 1,
  0, 1, 0, 1
  ), nrow=4, ncol=3)
contrasts(my.factor) <- my.contrast
contrasts(my.factor)
```

```
##         [,1] [,2] [,3]
## Andrew    -2    0    0
## Dillon    -1    0    1
## Patty      1    1    0
## Shinhye    2    1    1
```

Your turn!

Let's load in some real data from the data folder. This is a dataset containing information about different primates - how large they are, how large their brains are, and how much energy they use in a day.

```
dat <- read.csv('data/primate_energetics.csv')
```

Look at the variable `clade`. This variable contains five levels:

Strepsirrhini	lemurs and lorises
Platyrrhini	New World monkeys like capuchins and howlers
Cercopithecoidea	Old World monkeys like macaques and babboons
Hominoidea	apes like gibbons and gorillas
Homo	people like you

How would you construct a contrast for this variable?

Robusticity

Some assumptions of the linear model

1. Errors are normally distributed
2. Errors are independent
3. Variation is homoscedastic
4. X is measured without error

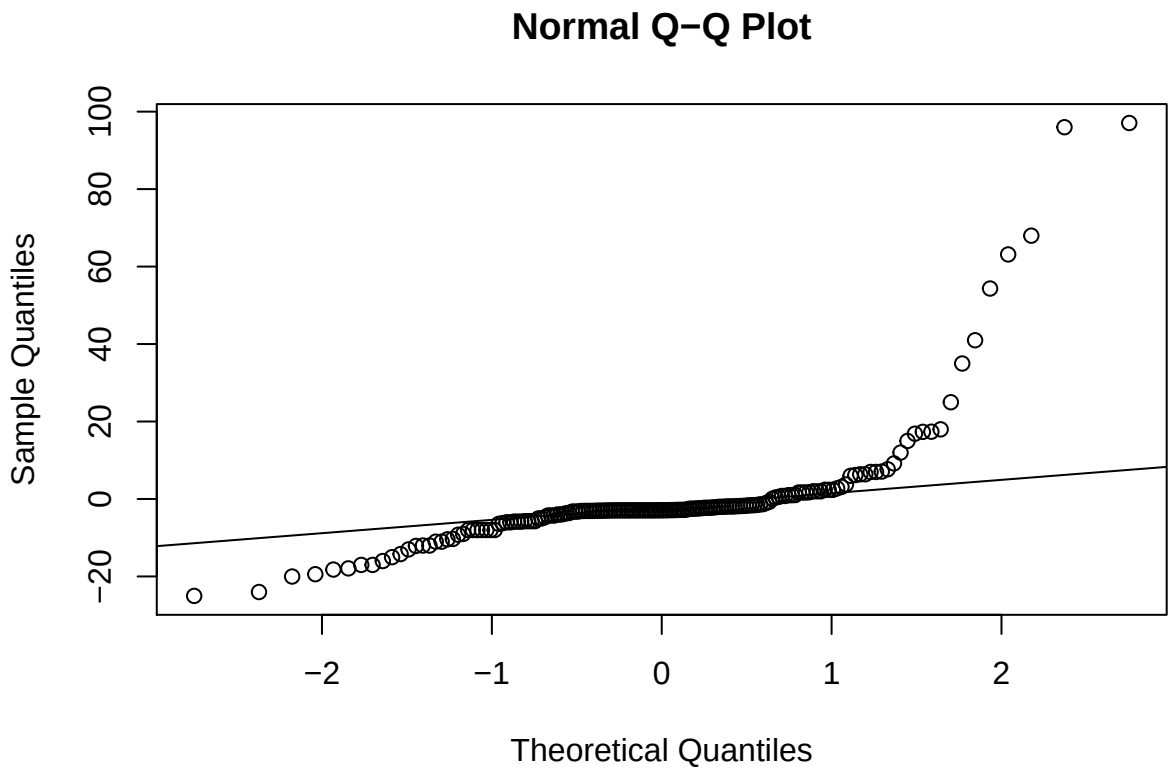
Checking normality of errors

Suppose we were interested in predicting body size from brain size. We could construct a model like this:

```
model.1 <- lm(W ~ BrainW, data=dat)
```

And then plot the residuals from the model against a normal distribution

```
qqnorm(model.1$residuals)
qqline(model.1$residuals)
```



Checking independence of errors

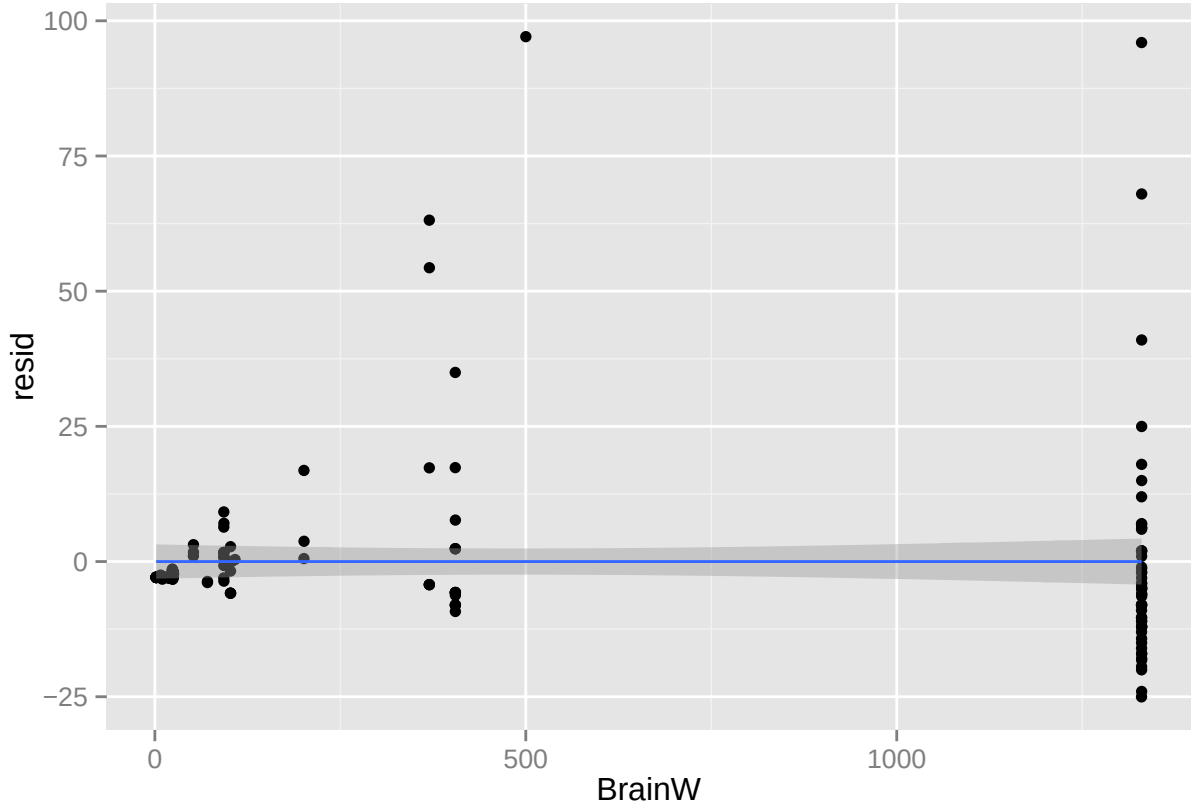
This one is pretty easy - you can simply regress your predictors on your residuals (or plot them)

```
summary(lm(model.1$resid ~ BrainW, data=dat[!(is.na(dat$BrainW)), ]))
```

```
##
## Call:
## lm(formula = model.1$resid ~ BrainW, data = dat[!(is.na(dat$BrainW)),
##     ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -25.023  -4.269  -2.915   0.373  97.062
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.303e-16  1.607e+00       0        1
## BrainW       7.947e-19  2.115e-03       0        1
##
```

```
## Residual standard error: 15.95 on 167 degrees of freedom
## Multiple R-squared:  7.715e-34,  Adjusted R-squared:  -0.005988
## F-statistic: 1.288e-31 on 1 and 167 DF,  p-value: 1
```

```
library(ggplot2)
dat.1 <- dat[!(is.na(dat$BrainW)), ]
dat.1$resid <- model.1$residuals
ggplot(data=dat.1, aes(x=BrainW, y=resid)) + geom_jitter() + stat_smooth(method='lm')
```

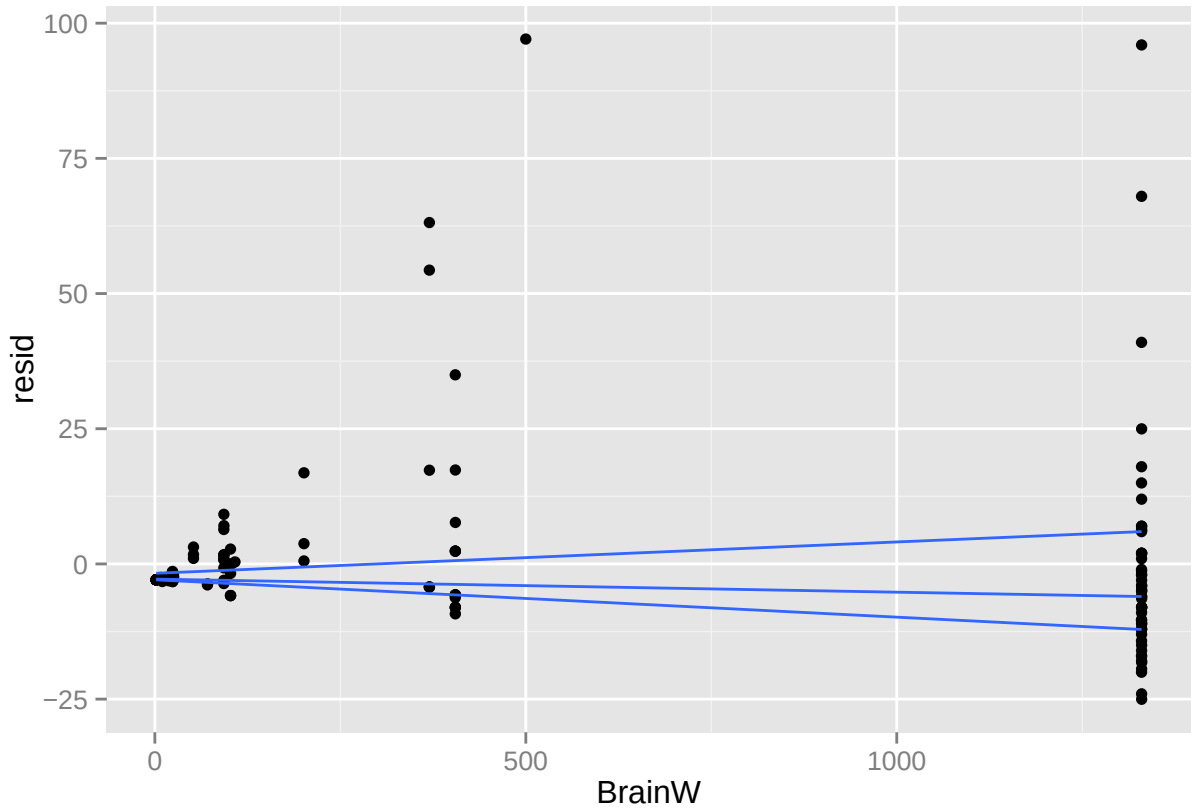


Checking homoscedasticity of errors

You can eyeball this by plotting your residuals against your predictor, and looking at rolling estimate of variance

```
ggplot(data=dat.1, aes(x=BrainW, y=resid)) + geom_jitter() + stat_quantile()
```

```
## Smoothing formula not specified. Using: y ~ x
```



A more formal test, called the Breusch-Pagan test, is found in the lmtest package. Essentially, it is just squaring the residuals and regressing your predictors on it.

```
library(lmtest)
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
bptest(model.1)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model.1
## BP = 4.2271, df = 1, p-value = 0.03978
```

Checking for error in X

There isn't really a way to establish statistically that your predictors have been measured without error. The good news is that linear models are fairly robust to violations of this assumption already. The better news is that if you suspect your predictors have very large margins of error, you can use major axis regressions.

```
library(lmodel2)
model.2 <- lmodel2(W ~ BrainW, data=dat, nperm=10)
```

```
## RMA was not requested: it will not be computed.
```

```
model.2
```

```
##
## Model II regression
##
## Call: lmodel2(formula = W ~ BrainW, data = dat, nperm = 10)
##
## n = 169   r = 0.8665701   r-square = 0.7509437
## Parametric P-values:   2-tailed = 2.767738e-52    1-tailed = 1.383869e-52
## Angle between the two OLS regression lines = 0.8989437 degrees
##
## Permutation tests of OLS, MA, RMA slopes: 1-tailed, tail corresponding to sign
## A permutation test of r is equivalent to a permutation test of the OLS slope
## P-perm for SMA = NA because the SMA slope cannot be tested
##
## Regression results
##   Method  Intercept      Slope Angle (degrees) P-perm (1-tailed)
## 1    OLS  2.9121421 0.04745212        2.716768        0.09090909
## 2     MA  2.8947748 0.04748750        2.718791        0.09090909
## 3    SMA -0.6740652 0.05475855        3.134304                NA
##
## Confidence intervals
##   Method 2.5%-Intercept 97.5%-Intercept 2.5%-Slope 97.5%-Slope
## 1    OLS     -0.2608942        6.085178 0.04327719  0.05162705
## 2     MA      0.8436461        4.945092 0.04331025  0.05166641
## 3    SMA     -2.8012460        1.297108 0.05074254  0.05909240
##
## Eigenvalues: 339487 252.3885
##
## H statistic used for computing C.I. of MA: 1.737758e-05
```

Linear models are very susceptible to outliers

This isn't really an assumption that we are violating so much as a consequence of parameterizing with means (highly susceptible to outliers) and variance (squared outliers). You can test for outliers with various measures of leverage, including:

1. DFBetas
2. DFFit
3. Cook's Distance

These can be calculated with:

```
influence.measures(model.1)
```

```
## Influence measures of
##   lm(formula = W ~ BrainW, data = dat) :
##
##        dfb.1_  dfb.BrnW     dffit cov.r   cook.d     hat inf
## 8    0.000306 -0.000171  0.000308 1.021 4.77e-08 0.00855
## 13   0.009798 -0.005569  0.009848 1.021 4.88e-05 0.00870
## 14  -0.004242  0.002411 -0.004264 1.021 9.15e-06 0.00870
## 15   0.009798 -0.005569  0.009848 1.021 4.88e-05 0.00870
## 16   0.009798 -0.005569  0.009848 1.021 4.88e-05 0.00870
## 17   0.005703 -0.003241  0.005732 1.021 1.65e-05 0.00870
## 18   0.037312 -0.021206  0.037503 1.019 7.07e-04 0.00870
## 19   0.053728 -0.030537  0.054003 1.017 1.46e-03 0.00870
## 20  -0.021211  0.012056 -0.021320 1.020 2.29e-04 0.00870
## 21  -0.017700  0.010060 -0.017791 1.021 1.59e-04 0.00870
## 22   0.004533 -0.002576  0.004556 1.021 1.04e-05 0.00870
## 23   0.004533 -0.002576  0.004556 1.021 1.04e-05 0.00870
## 24   0.041413 -0.023538  0.041626 1.019 8.71e-04 0.00870
## 26   0.088935 -0.040901  0.091534 1.006 4.19e-03 0.00739
## 27   0.002893 -0.001330  0.002978 1.020 4.46e-06 0.00739
## 28   0.019729 -0.009073  0.020306 1.019 2.07e-04 0.00739
## 39   0.401156  0.008417  0.532604 0.615 1.11e-01 0.00592   *
## 40  -0.033962  0.019024 -0.034175 1.019 5.87e-04 0.00857
## 41  -0.033962  0.019024 -0.034175 1.019 5.87e-04 0.00857
## 42  -0.033962  0.019024 -0.034175 1.019 5.87e-04 0.00857
## 43  -0.010165  0.005694 -0.010229 1.021 5.26e-05 0.00857
## 45  -0.033391  0.005749 -0.039294 1.015 7.75e-04 0.00605
## 46  -0.033391  0.005749 -0.039294 1.015 7.75e-04 0.00605
## 47  -0.033391  0.005749 -0.039294 1.015 7.75e-04 0.00605
## 48  -0.023818  0.004101 -0.028029 1.017 3.95e-04 0.00605
## 49  -0.023818  0.004101 -0.028029 1.017 3.95e-04 0.00605
## 50  -0.023818  0.004101 -0.028029 1.017 3.95e-04 0.00605
## 51  -0.023818  0.004101 -0.028029 1.017 3.95e-04 0.00605
## 52   0.009847 -0.001695  0.011588 1.018 6.75e-05 0.00605
## 53   0.009847 -0.001695  0.011588 1.018 6.75e-05 0.00605
## 54   0.009847 -0.001695  0.011588 1.018 6.75e-05 0.00605
## 55   0.072430 -0.012471  0.085235 1.004 3.63e-03 0.00605
## 56   0.031890 -0.005491  0.037528 1.015 7.07e-04 0.00605
## 57   0.147449 -0.025389  0.173515 0.960 1.47e-02 0.00605   *
## 58  -0.038390  0.006610 -0.045177 1.014 1.02e-03 0.00605
## 59  -0.025899  0.004459 -0.030477 1.016 4.67e-04 0.00605
## 60  -0.018550  0.004302 -0.021101 1.017 2.24e-04 0.00617
## 61  -0.018550  0.004302 -0.021101 1.017 2.24e-04 0.00617
## 62  -0.018550  0.004302 -0.021101 1.017 2.24e-04 0.00617
## 63   0.288170 -0.066822  0.327795 0.835 4.89e-02 0.00617   *
## 64   0.075552 -0.017520  0.085941 1.004 3.69e-03 0.00617
## 65   0.244716 -0.056746  0.278366 0.881 3.63e-02 0.00617   *
## 69   0.015941 -0.008929  0.016041 1.020 1.29e-04 0.00857
## 70   0.016795 -0.142496 -0.173246 1.011 1.50e-02 0.01829
## 71   0.021046 -0.178566 -0.217099 1.000 2.34e-02 0.01829
## 72   0.005364 -0.045515 -0.055336 1.029 1.54e-03 0.01829
## 73   0.011901 -0.100978 -0.122769 1.021 7.54e-03 0.01829
## 74   0.005030 -0.042678 -0.051887 1.029 1.35e-03 0.01829
## 75   0.015019 -0.127429 -0.154927 1.015 1.20e-02 0.01829
## 76   0.020193 -0.171328 -0.208299 1.003 2.15e-02 0.01829
```

```
## 77   0.010893 -0.092423 -0.112367 1.023 6.33e-03 0.01829
## 78   0.010137 -0.086012 -0.104573 1.024 5.48e-03 0.01829
## 79  -0.005325  0.045182  0.054932 1.029 1.52e-03 0.01829
## 80   0.012574 -0.106688 -0.129710 1.020 8.42e-03 0.01829
## 81   0.001105 -0.009373 -0.011396 1.031 6.53e-05 0.01829
## 82   0.003944 -0.033461 -0.040682 1.030 8.32e-04 0.01829
## 83   0.008712 -0.073917 -0.089868 1.026 4.05e-03 0.01829
## 84   0.008628 -0.073206 -0.089004 1.026 3.97e-03 0.01829
## 85   0.015272 -0.129579 -0.157541 1.015 1.24e-02 0.01829
## 86   0.016287 -0.138187 -0.168006 1.012 1.41e-02 0.01829
## 87  -0.005158  0.043763  0.053207 1.029 1.42e-03 0.01829
## 88   0.014259 -0.120984 -0.147092 1.017 1.08e-02 0.01829
## 89   0.006702 -0.056867 -0.069139 1.028 2.40e-03 0.01829
## 90  -0.012535  0.106353  0.129303 1.020 8.36e-03 0.01829
## 91   0.002524 -0.021415 -0.026036 1.030 3.41e-04 0.01829
## 92  -0.034913  0.296225  0.360149 0.950 6.26e-02 0.01829   *
## 93  -0.060171  0.510524  0.620691 0.815 1.72e-01 0.01829   *
## 94  -0.090756  0.770028  0.936194 0.626 3.44e-01 0.01829   *
## 95  -0.001650  0.013999  0.017020 1.031 1.46e-04 0.01829
## 96   0.006702 -0.056867 -0.069139 1.028 2.40e-03 0.01829
## 97   0.005030 -0.042678 -0.051887 1.029 1.35e-03 0.01829
## 98   0.006702 -0.056867 -0.069139 1.028 2.40e-03 0.01829
## 99   0.006702 -0.056867 -0.069139 1.028 2.40e-03 0.01829
## 100  0.001689 -0.014331 -0.017424 1.031 1.53e-04 0.01829
## 101 -0.021006  0.178226  0.216686 1.000 2.33e-02 0.01829
## 102 -0.005827  0.049438  0.060106 1.029 1.82e-03 0.01829
## 103 -0.010014  0.084966  0.103301 1.024 5.35e-03 0.01829
## 104  0.001689 -0.014331 -0.017424 1.031 1.53e-04 0.01829
## 105 -0.001650  0.013999  0.017020 1.031 1.46e-04 0.01829
## 106 -0.001650  0.013999  0.017020 1.031 1.46e-04 0.01829
## 107 -0.004991  0.042345  0.051483 1.029 1.33e-03 0.01829
## 108  0.009215 -0.078184 -0.095056 1.025 4.53e-03 0.01829
## 109  0.007539 -0.063968 -0.077772 1.027 3.04e-03 0.01829
## 110 -0.015064  0.127809  0.155389 1.015 1.21e-02 0.01829
## 111  0.003359 -0.028500 -0.034651 1.030 6.04e-04 0.01829
## 112 -0.000815  0.006916  0.008409 1.031 3.56e-05 0.01829
## 113  0.004194 -0.035588 -0.043267 1.030 9.41e-04 0.01829
## 114  0.003359 -0.028500 -0.034651 1.030 6.04e-04 0.01829
## 115  0.004194 -0.035588 -0.043267 1.030 9.41e-04 0.01829
## 116  0.013416 -0.113832 -0.138396 1.018 9.58e-03 0.01829
## 117  0.010054 -0.085300 -0.103707 1.024 5.39e-03 0.01829
## 118  0.009215 -0.078184 -0.095056 1.025 4.53e-03 0.01829
## 119  0.014259 -0.120984 -0.147092 1.017 1.08e-02 0.01829
## 120  0.010054 -0.085300 -0.103707 1.024 5.39e-03 0.01829
## 121  0.000854 -0.007248 -0.008813 1.031 3.91e-05 0.01829
## 122 -0.005827  0.049438  0.060106 1.029 1.82e-03 0.01829
## 123  0.018959 -0.011453  0.018987 1.021 1.81e-04 0.00930
## 124  0.006199 -0.003745  0.006208 1.022 1.94e-05 0.00930
## 125  0.010573 -0.006387  0.010589 1.021 5.64e-05 0.00930
## 126 -0.018011  0.011392 -0.018014 1.022 1.63e-04 0.00986
## 127  0.002152 -0.001193  0.002167 1.021 2.36e-06 0.00849
## 128 -0.019484  0.012294 -0.019488 1.022 1.91e-04 0.00983
## 129 -0.018256  0.011634 -0.018257 1.022 1.68e-04 0.00996
## 130 -0.019390  0.012357 -0.019392 1.022 1.89e-04 0.00996
```

```
## 131 -0.017871  0.011389 -0.017873 1.022 1.61e-04 0.00996
## 132 -0.018124  0.011550 -0.018125 1.022 1.65e-04 0.00996
## 133 -0.017764  0.011321 -0.017765 1.022 1.59e-04 0.00996
## 134 -0.017745  0.011309 -0.017747 1.022 1.58e-04 0.00996
## 135 -0.017840  0.011417 -0.017841 1.022 1.60e-04 0.01002
## 136 -0.023178  0.013625 -0.023245 1.021 2.72e-04 0.00901
## 137 -0.021983  0.012923 -0.022047 1.021 2.44e-04 0.00901
## 138 -0.020316  0.012738 -0.020322 1.022 2.08e-04 0.00975
## 139 -0.019319  0.012113 -0.019325 1.022 1.88e-04 0.00975
## 140 -0.018584  0.011858 -0.018585 1.022 1.74e-04 0.00998
## 141 -0.020521  0.013094 -0.020522 1.022 2.12e-04 0.00998
## 142 -0.019121  0.012200 -0.019122 1.022 1.84e-04 0.00998
## 143 -0.019468  0.012422 -0.019469 1.022 1.91e-04 0.00998
## 144 -0.018837  0.012019 -0.018838 1.022 1.78e-04 0.00998
## 145 -0.020254  0.012699 -0.020260 1.022 2.06e-04 0.00975
## 146 -0.018648  0.011898 -0.018648 1.022 1.75e-04 0.00998
## 147 -0.019468  0.012422 -0.019469 1.022 1.91e-04 0.00998
## 148 -0.017991  0.011480 -0.017992 1.022 1.63e-04 0.00998
## 149 -0.019398  0.012377 -0.019399 1.022 1.89e-04 0.00998
## 150 -0.018383  0.011729 -0.018384 1.022 1.70e-04 0.00998
## 151 -0.017083  0.010900 -0.017084 1.022 1.47e-04 0.00998
## 152 -0.018389  0.011733 -0.018390 1.022 1.70e-04 0.00998
## 153 -0.014243  0.008919 -0.014248 1.022 1.02e-04 0.00973
## 154 -0.015675  0.009816 -0.015680 1.022 1.24e-04 0.00973
## 155 -0.014555  0.009114 -0.014559 1.022 1.07e-04 0.00973
## 156 -0.015115  0.009465 -0.015120 1.022 1.15e-04 0.00973
## 157 -0.012003  0.007516 -0.012006 1.022 7.25e-05 0.00973
## 158 -0.018601  0.011648 -0.018607 1.022 1.74e-04 0.00973
## 159 -0.014990  0.009387 -0.014995 1.022 1.13e-04 0.00973
## 160 -0.015675  0.009816 -0.015680 1.022 1.24e-04 0.00973
## 161 -0.012003  0.007516 -0.012006 1.022 7.25e-05 0.00973
## 162 -0.013247  0.008296 -0.013252 1.022 8.83e-05 0.00973
## 163 -0.012812  0.008023 -0.012816 1.022 8.26e-05 0.00973
## 164 -0.014243  0.008919 -0.014248 1.022 1.02e-04 0.00973
## 165 -0.011391  0.007145 -0.011395 1.022 6.53e-05 0.00975
## 166 -0.011454  0.007184 -0.011457 1.022 6.60e-05 0.00975
## 167 -0.011952  0.007497 -0.011956 1.022 7.19e-05 0.00975
## 168 -0.011204  0.007028 -0.011208 1.022 6.32e-05 0.00975
## 169 -0.010145  0.006363 -0.010148 1.022 5.18e-05 0.00975
## 170 -0.009958  0.006246 -0.009961 1.022 4.99e-05 0.00975
## 171 -0.009584  0.006011 -0.009587 1.022 4.62e-05 0.00975
## 172 -0.010332  0.006481 -0.010335 1.022 5.37e-05 0.00975
## 173 -0.014133  0.008865 -0.014137 1.022 1.01e-04 0.00975
## 174 -0.012139  0.007614 -0.012143 1.022 7.42e-05 0.00975
## 175 -0.008649  0.005425 -0.008652 1.022 3.77e-05 0.00975
## 176 -0.013697  0.008591 -0.013701 1.022 9.44e-05 0.00975
## 177 -0.015818  0.010122 -0.015818 1.022 1.26e-04 0.01002
## 178 -0.018680  0.012038 -0.018680 1.022 1.75e-04 0.01012
## 179 -0.018686  0.012042 -0.018686 1.022 1.76e-04 0.01012
## 180 -0.018635  0.012009 -0.018635 1.022 1.75e-04 0.01012
## 181 -0.018630  0.012006 -0.018630 1.022 1.75e-04 0.01012
## 182 -0.018635  0.012009 -0.018635 1.022 1.75e-04 0.01012
## 183 -0.018690  0.012045 -0.018690 1.022 1.76e-04 0.01012
## 184 -0.018610  0.011993 -0.018610 1.022 1.74e-04 0.01012
```

```
## 185 -0.018610  0.011993 -0.018610 1.022 1.74e-04 0.01012
## 186 -0.018654  0.012021 -0.018654 1.022 1.75e-04 0.01012
## 187 -0.018673  0.012034 -0.018673 1.022 1.75e-04 0.01012
## 188 -0.018591  0.011981 -0.018591 1.022 1.74e-04 0.01012
## 189 -0.018635  0.012009 -0.018635 1.022 1.75e-04 0.01012
## 190 -0.018413  0.011866 -0.018413 1.022 1.71e-04 0.01012
## 191 -0.018616  0.011997 -0.018616 1.022 1.74e-04 0.01012
## 192 -0.018673  0.012034 -0.018673 1.022 1.75e-04 0.01012
## 193 -0.018578  0.011972 -0.018578 1.022 1.74e-04 0.01012
## 194 -0.018575  0.011970 -0.018575 1.022 1.74e-04 0.01012
## 195 -0.018727  0.012069 -0.018727 1.022 1.76e-04 0.01012
```

If you find your model contains high leverage cases, you can either pluck them out by hand (preferably by selecting against something like Cook's Distace), or try running a regression build to be robust to outliers.

```
library(robust)
```

```
## Loading required package: fit.models
## Loading required package: lattice
## Loading required package: MASS
## Loading required package: robustbase
## Loading required package: rrcov
## Scalable Robust Estimators with High Breakdown Point (version 1.3-8)
```

```
model.3 <- lmRob(W ~ BrainW, data=dat)
summary(model.3)
```

```
##
## Call:
## lmRob(formula = W ~ BrainW, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6681  -0.8612  -0.4966   3.5102 103.3319
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.7390757  0.3309017   2.234   0.0268 *
## BrainW      0.0435557  0.0005608  77.665   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.749 on 167 degrees of freedom
## Multiple R-Squared: 0.3895
##
## Test for Bias:
##             statistic p-value
## M-estimate      6.112 0.04707
## LS-estimate     6.212 0.04477
## 26 observations deleted due to missingness
```

Exluding the outliers in this case has brought the coefficient of our predictor close to zero. Can you guess why? If you aren't sure, plot it.

Your turn!

So far we've been predicting body weight from brain weight. Try predicting body weight from brain weight and clade. Your base model should look like this:

```
<- lm(W ~ BrainW + Clade, data=dat)
```

Rerun your regression diagnostics. What do you find?

Stepwise Models and Uncertainty Reduction

Fundamental constraints in prediction

We'll start with the bad news. Any kind of predicting method is constrained in how many independent variables you can include in the model (this is why machine learning algorithms always start with a dimensionality reduction method like PCA).

Let's do a little test to show how much of a problem this is. Run the code below a couple of times. How often do you get significant results? Keep in mind that these are randomly generated data, so we already know that there *shouldn't be* any significant predictors.

```
dat.toy <- data.frame(value=runif(4), var1=runif(4), var2=runif(4))
summary(lm(value ~ var1 + var2, data=dat.toy))
```

```
##
## Call:
## lm(formula = value ~ var1 + var2, data = dat.toy)
##
## Residuals:
##         1         2         3         4
## -0.004851 -0.005974 -0.004030  0.014856
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09061    0.03648  -2.484   0.2437
## var1        -0.12770    0.03865  -3.304   0.1871
## var2         1.22576    0.10951  11.193   0.0567 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01721 on 1 degrees of freedom
## Multiple R-squared:  0.9949, Adjusted R-squared:  0.9848
## F-statistic: 98.29 on 2 and 1 DF,  p-value: 0.07114
```

R won't let us run `lm` with four variables, but if it did, we would be able to 100% accurately predict four random number from four sets of other random numbers. Overfitting, combined with p hacking and the file drawer problem, are responsible for the giant world of false findings and irreproducable science that you keep hearing about. Generally speaking, it is bad science to throw predictors haphazardly into a model and see what comes out; however, there are times when this approach is either required or useful in its own right.

Weeding out predictors the smart way

It may happen to you in your life that you have no theory, hypothesis, or intuitions about how your predictors could be related to your outcome variable. In this case, you can weed out predictors using what is called a stepwise regression.

```
library(MASS)
dat.omit <- na.omit(dat)
model.base <- lm(W ~ ., data=dat.omit)
model.4 <- stepAIC(model.base)
```

```
## Start:  AIC=10.23
## W ~ Clade + Genus + Species + Group + N + BrainW + RMR + TEE +
##     Source + Method
##
##
## Step:  AIC=10.23
## W ~ Clade + Genus + Species + Group + N + BrainW + RMR + TEE +
##     Source
##
##
## Step:  AIC=10.23
## W ~ Clade + Genus + Species + Group + N + BrainW + RMR + TEE
##
##
## Step:  AIC=10.23
## W ~ Clade + Genus + Species + Group + N + RMR + TEE
##
##
## Step:  AIC=10.23
## W ~ Clade + Genus + Group + N + RMR + TEE
##
##
## Step:  AIC=10.23
## W ~ Genus + Group + N + RMR + TEE
##
##         Df Sum of Sq     RSS     AIC
## - N      1     0.576  57.225   8.976
## <none>                56.648  10.227
## - RMR    1    13.813  70.461  24.373
## - TEE    1    26.708  83.356  36.810
## - Genus  8   178.782 235.430  99.644
## - Group  2   230.458 287.106 126.328
##
## Step:  AIC=8.98
## W ~ Genus + Group + RMR + TEE
##
##         Df Sum of Sq     RSS     AIC
## <none>                57.225   8.976
## - RMR    1    13.952  71.177  23.122
## - TEE    1    30.698  87.922  38.757
## - Genus  8   221.384 278.609 110.105
## - Group  2   230.098 287.322 124.384
```

```
summary(model.4)
```

```
##
## Call:
## lm(formula = W ~ Genus + Group + RMR + TEE, data = dat.omit)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0820 -0.1801  0.0071  0.2316  3.6508
##
## Coefficients: (1 not defined because of singularities)
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -3.478e+00  6.041e-01  -5.758 3.11e-07 ***
## GenusCebus       3.158e+00  8.792e-01   3.592 0.000662 ***
## GenusEulemur     1.925e+00  6.803e-01   2.829 0.006334 **
## GenusHomo        4.979e+01  2.210e+00  22.527  < 2e-16 ***
## GenusLemur       2.151e+00  6.646e-01   3.236 0.001973 **
## GenusMacaca      7.444e+00  6.170e-01  12.065  < 2e-16 ***
## GenusMarmoset    3.306e+00  8.044e-01   4.110 0.000122 ***
## GenusMicrocebus  2.717e+00  6.089e-01   4.461 3.64e-05 ***
## GenusPan         1.017e+01  8.714e-01  11.673  < 2e-16 ***
## GenusTamarin     3.270e+00  6.737e-01   4.853 9.02e-06 ***
## GroupEcuador    -1.347e+01  1.053e+00 -12.800  < 2e-16 ***
## GroupEvenki     -1.465e+01  1.007e+00 -14.555  < 2e-16 ***
## GroupRussian            NA         NA      NA       NA
## RMR              7.939e-03  2.076e-03   3.825 0.000314 ***
## TEE              4.167e-03  7.344e-04   5.673 4.28e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9766 on 60 degrees of freedom
## Multiple R-squared:  0.9976, Adjusted R-squared:  0.997
## F-statistic:  1885 on 13 and 60 DF,  p-value: < 2.2e-16
```

Doing it this way isn't particularly helpful - genera tend to be around the same body size, so really we are just recapitulating taxonomies here.

Specifying forward stepwise models

Part of the problem is that backwards models tend to keep predictors that aren't particularly useful. The other part of the problem is that we threw every variable we had into the model.

```
model.base <- lm(W ~ 1, data=dat.omit)
model.5 <- stepAIC(model.base, scope=list(upper = ~ W + BrainW + RMR + TEE), direction='forward')
```

```
## Start:  AIC=428.05
## W ~ 1
##
##          Df Sum of Sq     RSS    AIC
## + BrainW  1     22726   697.2 169.98
## + RMR     1     22445   978.2 195.04
```

```
## + TEE     1     20070  3352.9 286.20
## <none>                23422.7 428.05
##
## Step:  AIC=169.98
## W ~ BrainW
##
##        Df Sum of Sq    RSS    AIC
## + RMR   1    257.52 439.64 137.86
## + TEE   1    133.83 563.33 156.21
## <none>              697.16 169.98
##
## Step:  AIC=137.86
## W ~ BrainW + RMR
##
##        Df Sum of Sq    RSS    AIC
## <none>              439.64 137.86
## + TEE   1    10.833 428.81 138.01
```

```
summary(model.5)
```

```
##
## Call:
## lm(formula = W ~ BrainW + RMR, data = dat.omit)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.5206  0.0926  0.3106  0.4514  9.7928
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.897984   0.400010  -2.245   0.0279 *
## BrainW       0.025710   0.002757   9.326 5.89e-14 ***
## RMR          0.016591   0.002573   6.449 1.18e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.488 on 71 degrees of freedom
## Multiple R-squared:  0.9812, Adjusted R-squared:  0.9807
## F-statistic:  1856 on 2 and 71 DF,  p-value: < 2.2e-16
```

Comparing homogeneous models

Stepwise regression is evalued by maximizing the likelihood that a model will reduce your uncertainty about the dependent variable. This can't be assessed in an exact sense, but models that differ only the inclusion of a parameter can be compared my relative likelihood (this is sometimes called comparing 'nested models'). This can be calculated with:

```
p <- exp((model.5$anova$AIC[3] - model.5$anova$AIC[2]) / 2)
p
```

```
## [1] 1.06087e-07
```

To put it another way, the probability that the complex model is better is 0.9999999.

Your turn!

So far, we've been predicting body weight from brain weight. What happens if you run the model the other way around? E.g., you try:

```
BrainW ~ .
```

Comparing Heterogeneous Models

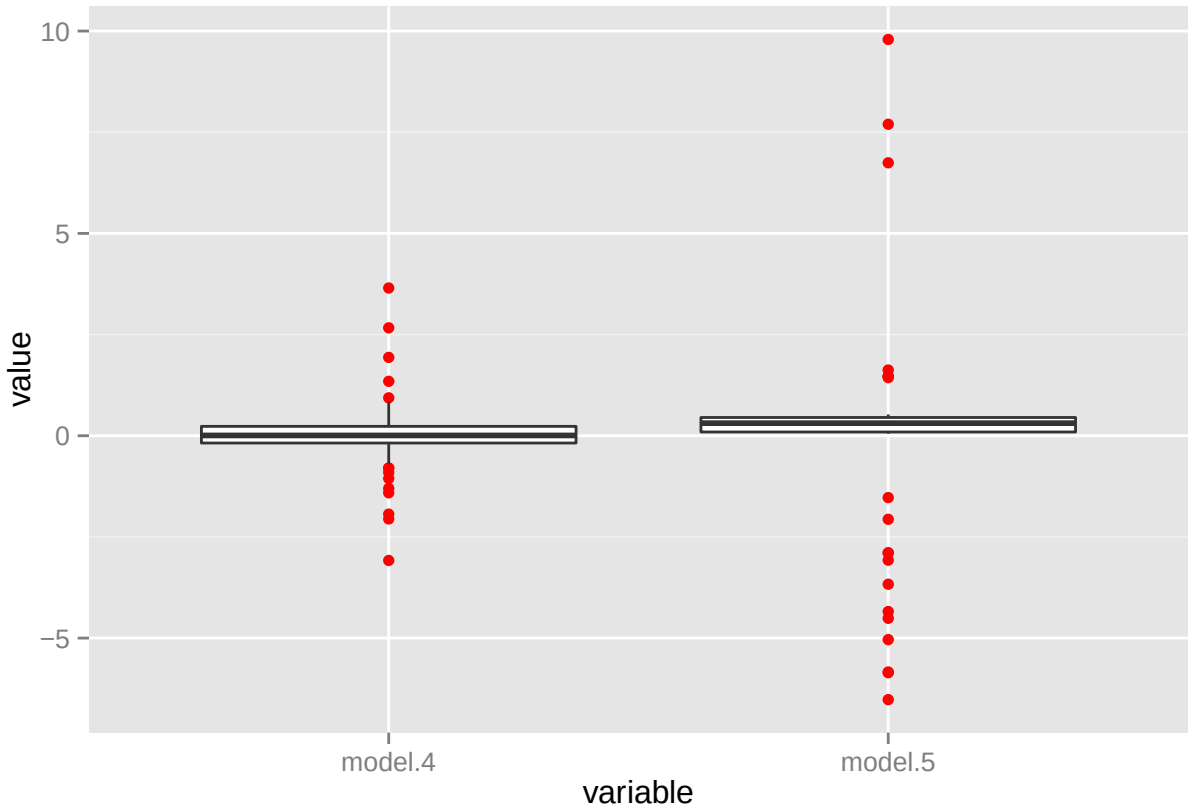
Visualizing prediction error

One easy way to visualize this is to use make a dataframe out of your error terms and plot it.

```
library(reshape2)
errors <- data.frame(model.4 = model.4$residuals, model.5 = model.5$residuals)
errors <- melt(errors)
```

```
## No id variables; using all as measure variables
```

```
ggplot(data=errors, aes(x=variable, y=value)) + geom_boxplot(outlier.colour='red')
```



Quantifying prediction error

- The good news: the math used to quantify prediction error is simple

- The bad news: R will not do it for you

  Cautionary note - none of these methods take model complexity into account

RMSE

The root mean square error is the preferred mean-based way to quantify model fit. The math looks like `sqrt(mean(error^2))`

```
sqrt(mean(model.4$residuals**2))
```

```
## [1] 0.8793775
```

```
sqrt(mean(model.5$residuals**2))
```

```
## [1] 2.437431
```

MdAE

The median absolute error is the preferred median-based way to quantify model fit. It is particularly useful when a model's fit is unduly influenced by one or a few outliers. The math looks like `median(abs(error))`

```
median(abs(model.4$residuals))
```

```
## [1] 0.1880393
```

```
median(abs(model.5$residuals))
```

```
## [1] 0.3898049
```

> side note - this is sometimes abbreviated `MAD` for 'median absolute deviation', but it could also mean 'mean absolute deviation'

Your turn!

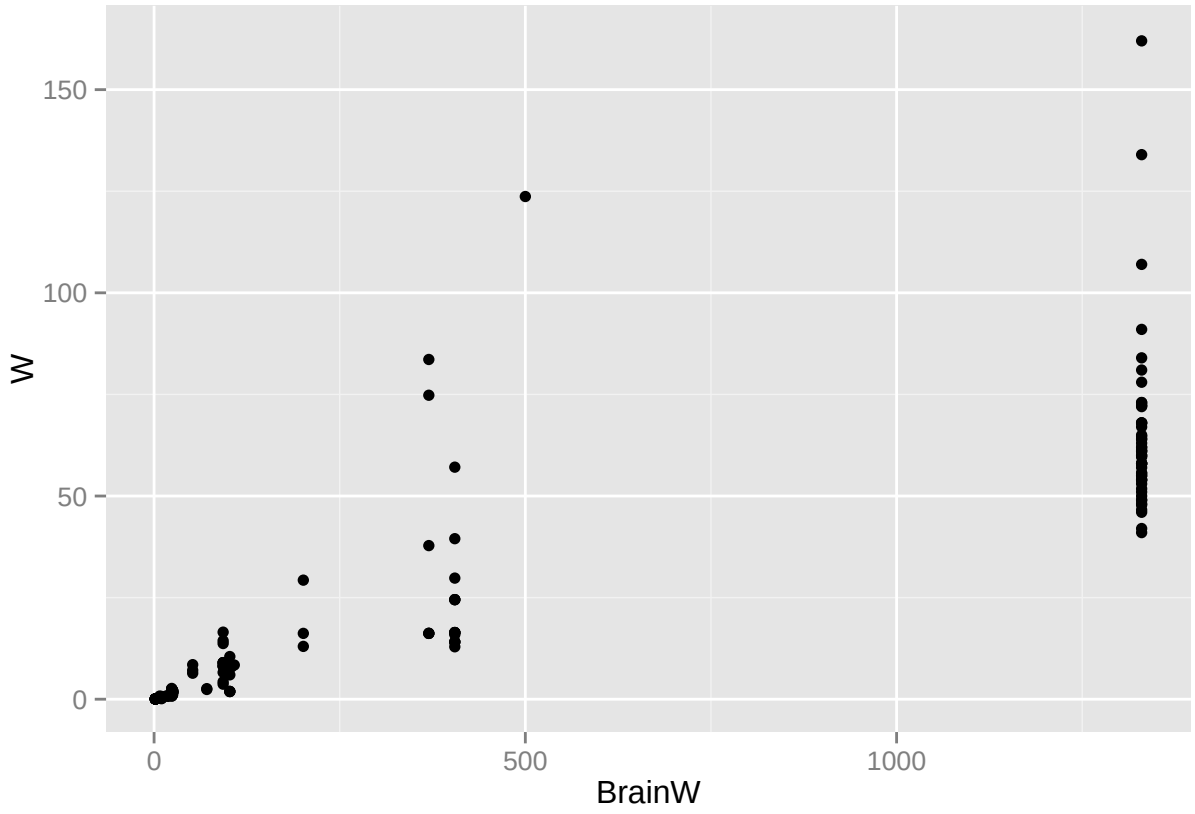
You're going to compare two heterogeneous models for explaining body weight. Try using `TEE ~ W` and `TEE ~ RMR`. Which is a better fit?

Generalized Models

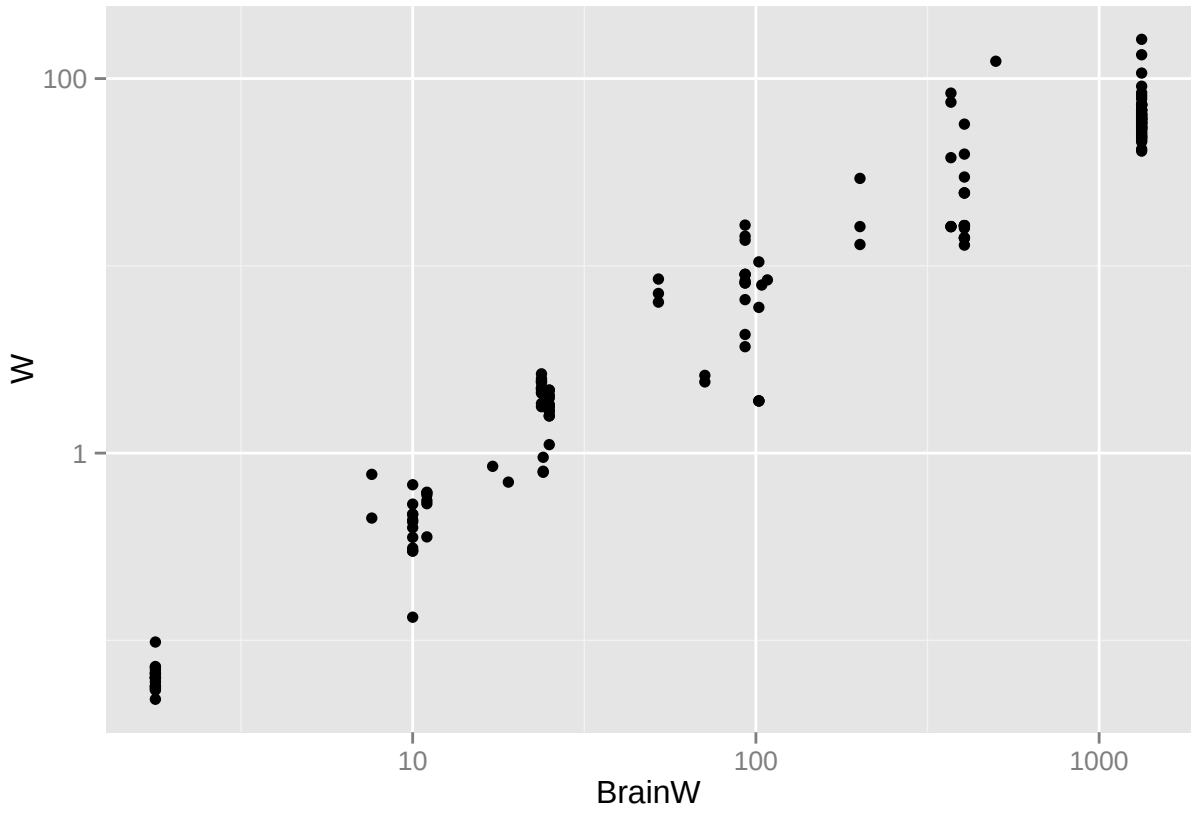
LM assumes linear relationships

The linear model we have been using is assuming a couple of things that we haven't talked much about. One of these is that the relationship between the predictors and the outcome is linear (as opposed to quadratic, for example). This assumption can often be met by nonlinear transforms of the data. For example, we've been using data that are usually analyzed after they've been log transformed.

```
qplot(data=dat, x=BrainW, y=W)
```



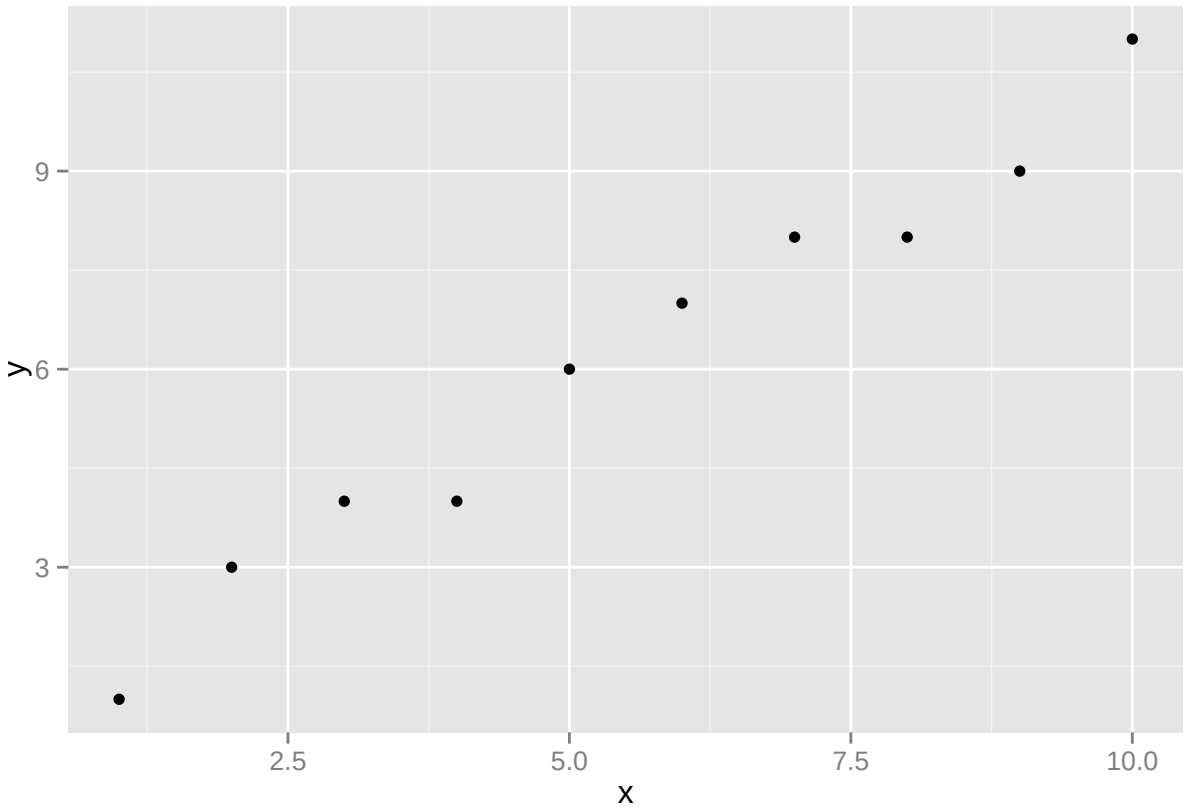

```
qplot(data=dat, x=BrainW, y=W, log='xy')
```



LM also assumes normal variance

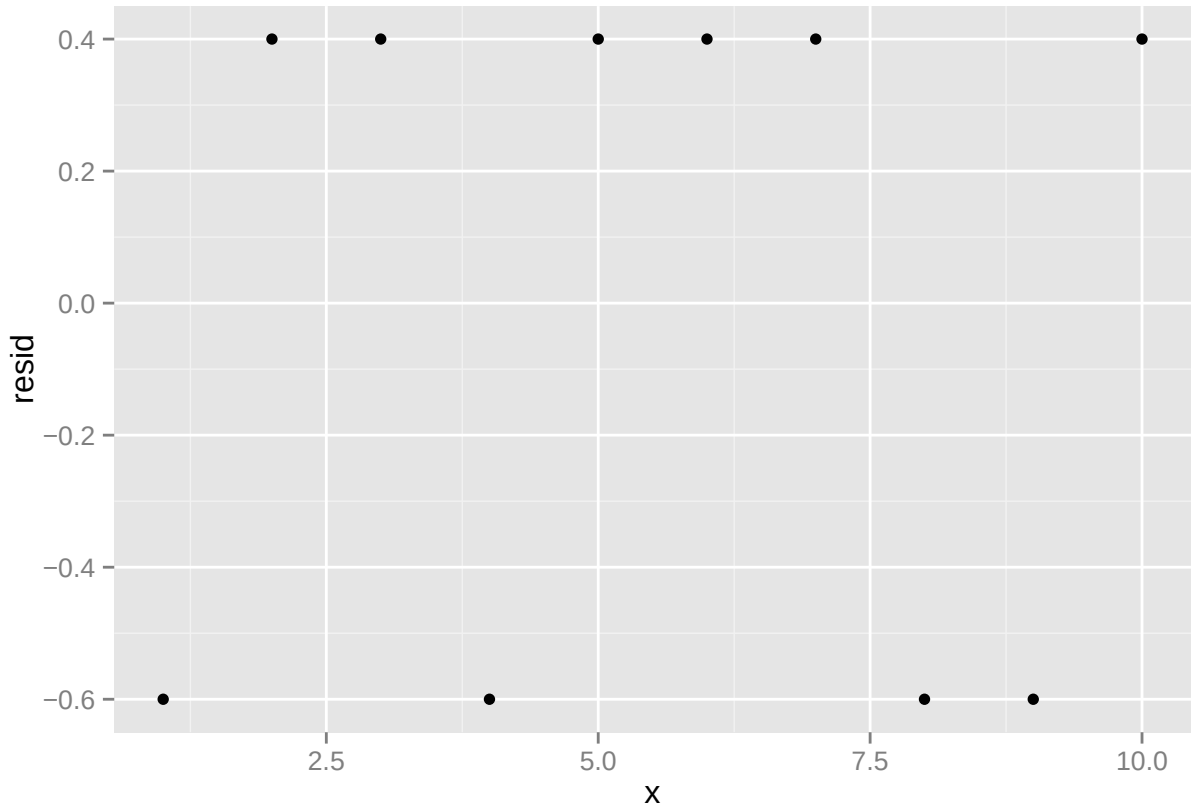
This one is a little trickier. To show why this is a problem, let's make another toy dataset.

```
dat.toy <- data.frame(x=1:10, y=1:10+rbinom(10, 1, .5))
qplot(data=dat.toy, x=x, y=y)
```



If we predict `y` from `x` and plot the residuals against, x, we get:

```
model.toy <- lm(y ~ x, data=dat.toy)
dat.toy$resid <- model.toy$residuals
qplot(data=dat.toy, x=x, y=resid)
```



The actual prediction equation here should be `y = 0.5 + x`, but because the errors around y are non-normal, linear regression produces an incorrect solution something like 0.6 + 1 x. This *usually* cannot be corrected by nonlinear transforms.

The generalized linear model

The good news is that `lm` is a subset of of `glm` or generalized linear models, where the link is linear and the variance is normal. `lm` in R is really just a convenience function that (behind the scenes) is running:

```
summary(glm(y~x, data=dat.toy, family=gaussian))
```

```
##
## Call:
## glm(formula = y ~ x, family = gaussian, data = dat.toy)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##   -0.6    -0.6     0.4     0.4     0.4
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6000     0.3742   1.604    0.147
## x             1.0000     0.0603  16.583 1.77e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.3)
```

```
##
##     Null deviance: 84.9  on 9  degrees of freedom
## Residual deviance:  2.4  on 8  degrees of freedom
## AIC: 20.108
##
## Number of Fisher Scoring iterations: 2
```

You will typically only encounter two types of non-linear, linear models

The binomial family

The binomial distribution models binary variables. Typically, this is presented as modeling coin flips, with a certain probability.

```
dat.toy <- data.frame(x=1:10, y=rbinom(10,1,0.5))
summary(glm(y~x, data=dat.toy, family=binomial))
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial, data = dat.toy)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0873  -0.8452  -0.7323   0.8465   1.8014
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.6993     1.6547  -1.027    0.304
## x             0.1484     0.2505   0.592    0.554
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 12.217  on 9  degrees of freedom
## Residual deviance: 11.852  on 8  degrees of freedom
## AIC: 15.852
##
## Number of Fisher Scoring iterations: 4
```

The poisson family

The poisson distribution models count variables. Typically, this is presented as something like 'how many people enter a store per minute'.

```
data.toy <- data.frame(x=1:10, y=rpois(10,1))
summary(glm(y~x, data=dat.toy, family=poisson))
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson, data = dat.toy)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -0.9553  -0.7680  -0.6842   0.4297   1.2708
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.8125     1.4288  -1.269    0.205
## x             0.1028     0.2064   0.498    0.618
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 7.2238  on 9  degrees of freedom
## Residual deviance: 6.9691  on 8  degrees of freedom
## AIC: 16.969
##
## Number of Fisher Scoring iterations: 6
```

Your turn!

The binomial family glm is more commonly known as the logistic regression. In logistic regression, each 0 and 1 outcome is modeled as the odds ratio that a row will be equal to 0 or 1 based on the predictor variables. This is the engine behind most classification algorithms.

Let's go back to the `Clade` variable, and make a new binary variable based on it.

```
dat$bin <- 0
dat$bin[dat$Clade == 'Homo'] <- 1
```

Using logistic regression and the other variables in the dataset, find the best way to predict whether a case is from a human or not.