Stats Bootcamp Day 3: Transformations + MT

Your Name

9/1/2021

Recap + Goals for Today

Last time, we showed you how to run a linear regression in R, and the basics of interpreting the results in a statistical way. We also discussed the assumptions that underly a linear model.

Today, we will find ways to adapt the linear model framework for situations where the assumptions won't hold under the basic framework.

With great power comes great responsibility. We will learn how adapting the basic framework comes at the risk of *p-hacking*. We will demonstrate ways to reduce multiple testing and selective inference concerns.

Examples inspired by Rina Barber's STAT 343, Autumn 2020

Transformations

Load in the dataset. We may need to install the faraway package.

```
#install.packages("faraway")
library(faraway)
```

```
## Warning: package 'faraway' was built under R version 4.0.2
```

```
data(gala)

head(gala)
```

```
##              Species Endemics  Area Elevation Nearest Scruz Adjacent
## Baltra            58       23 25.09       346     0.6   0.6     1.84
## Bartolome         31       21  1.24       109     0.6  26.3   572.33
## Caldwell           3        3  0.21       114     2.8  58.7     0.78
## Champion          25        9  0.10        46     1.9  47.4     0.18
## Coamano            2        1  0.05        77     1.9   1.9   903.82
## Daphne.Major      18       11  0.34       119     8.0   8.0     1.84
```

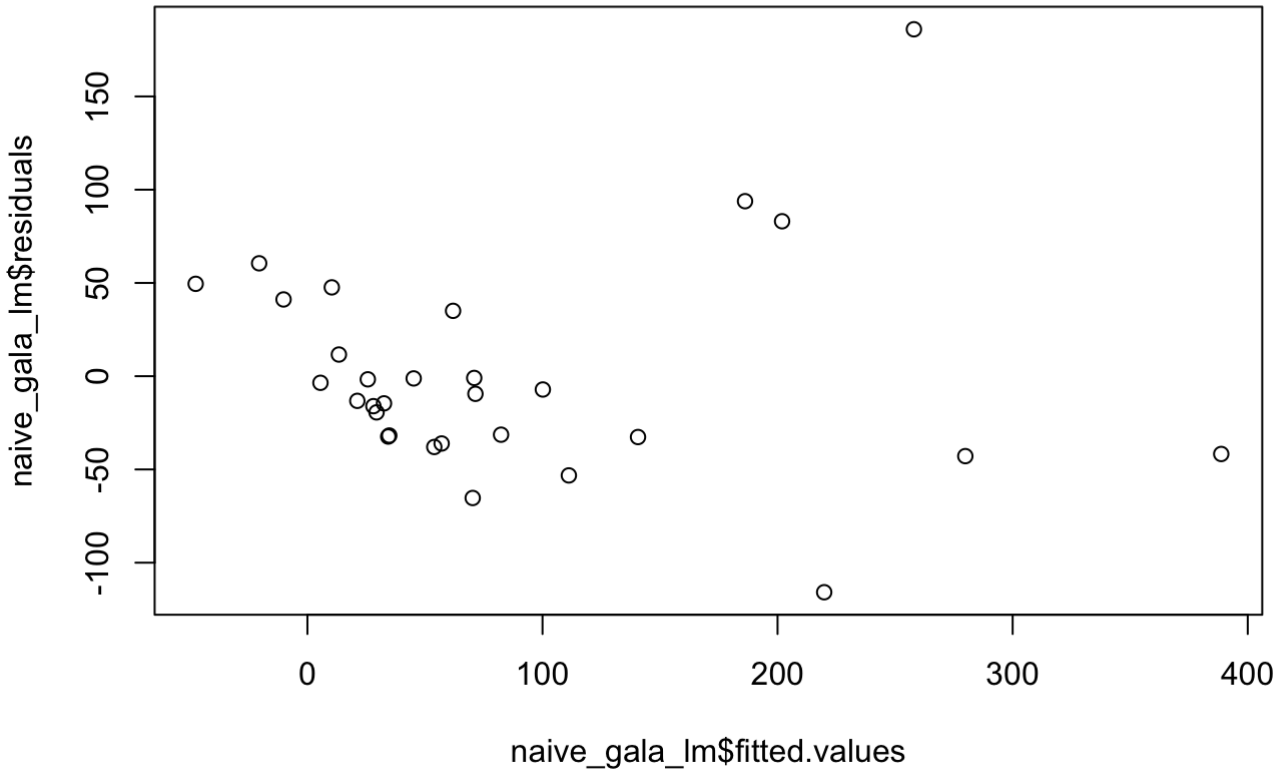
Can we predict the number of Species based on characteristics of the island?

```
naive_gala_lm = lm(Species ~ Area + Elevation + Nearest + Adjacent, data=gala)
summary(naive_gala_lm)
```

```
##
## Call:
## lm(formula = Species ~ Area + Elevation + Nearest + Adjacent,
##     data = gala)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -115.84  -32.56  -11.34   29.19  186.00
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.17925   18.10976  -0.010 0.992181
## Area        -0.02489    0.02252  -1.105 0.279599
## Elevation    0.32540    0.05366   6.064 2.46e-06 ***
## Nearest     -0.72732    0.82637  -0.880 0.387167
## Adjacent    -0.07858    0.01746  -4.501 0.000136 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 61.28 on 25 degrees of freedom
## Multiple R-squared:  0.7537, Adjusted R-squared:  0.7143
## F-statistic: 19.12 on 4 and 25 DF,  p-value: 2.58e-07
```

Let's build ourselves some of the charts Ander showed us yesterday.

```
plot(naive_gala_lm$fitted.values, naive_gala_lm$residuals)
```



Non-linear trend? Heteroskedasticity?

Optional: understanding the derivation of leverage scores Recall that the our OLS formula for our coefficients estimates is: $\hat{\beta} = (X^T X)^{-1} X^T y$

So our fitted values are $\hat{y} = X\hat{\beta} = (X^T X)^{-1} X^T y$

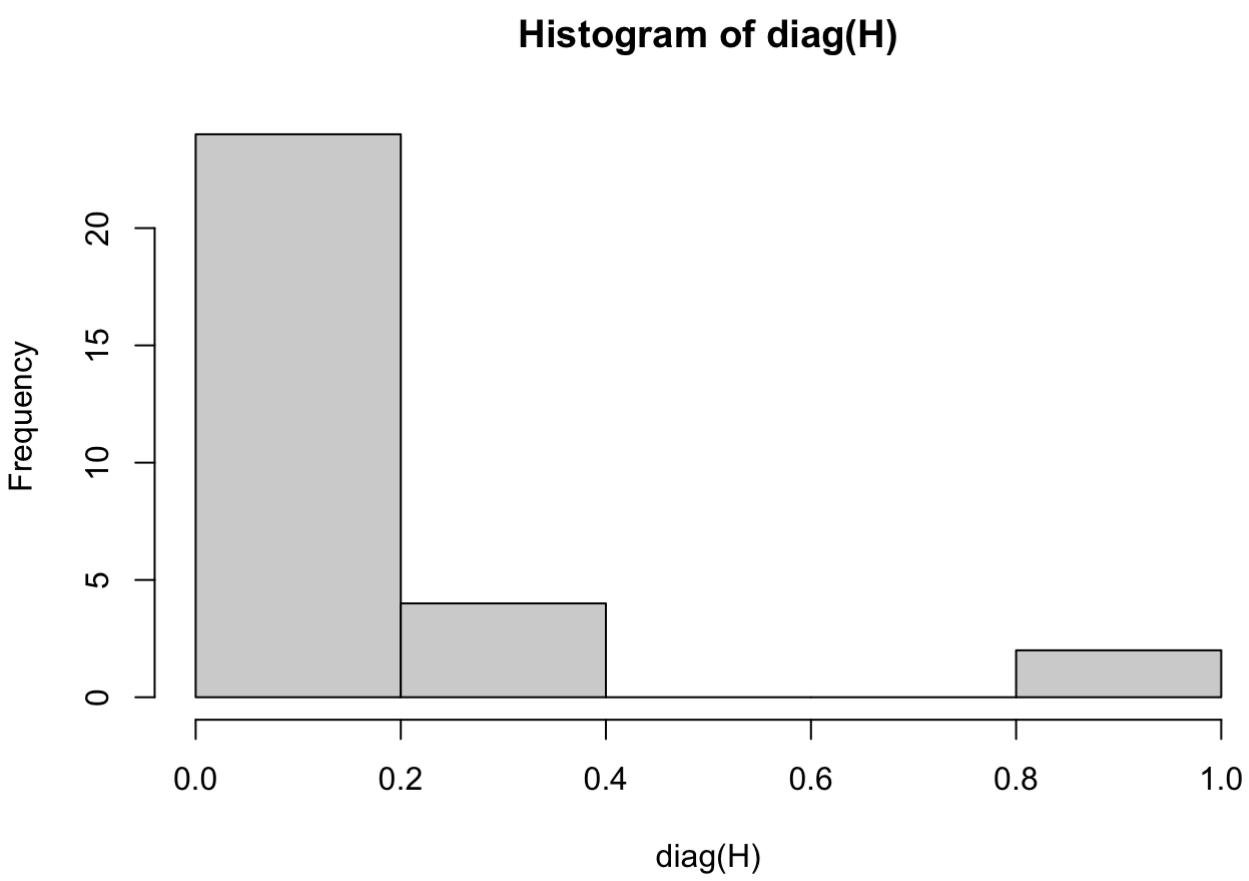
So multiplying y by the matrix $(X^T X)^{-1} X^T$ will give us our fitted values.

Therefore, we call $(X^T X)^{-1} X^T$ the "hat" matrix. The i'th diagonal element of the hat matrix roughly correspond to how much the i'th response y_i affects *all* of our coefficient estimates. It's the "leverage" score.

Let's calculate the hat matrix here:

```
X = model.matrix(naive_gala_lm)
# or equivalently
X = cbind(1,gala$Area, gala$Elevation, gala$Nearest, gala$Adjacent )


H = X %*% solve(t(X) %*% X, t(X))
hist(diag(H))
```



We can see that we have some very high leverage scores (1 is the upper bound)!

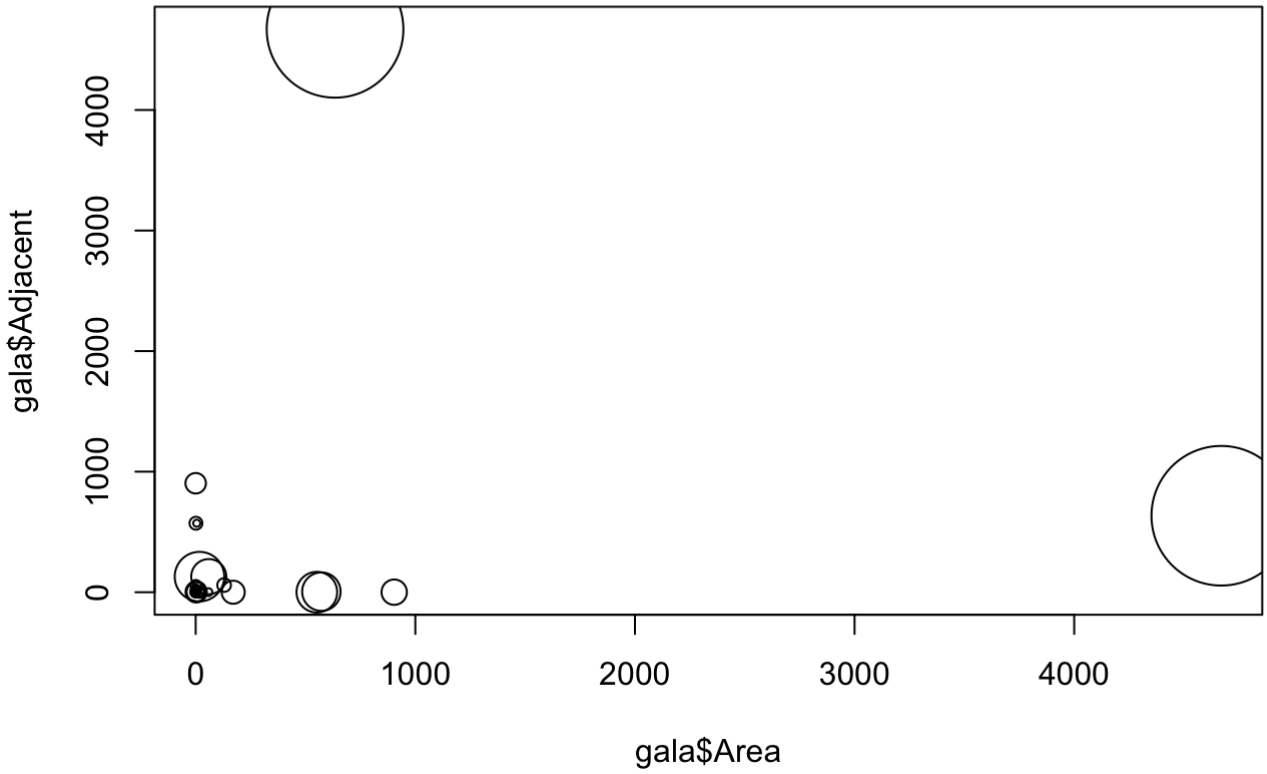
Let's see where these high leverage points are. It might be useful to know which variables have the largest outliers:

```
summary(gala)
```

```
##     Species          Endemics          Area            Elevation
##  Min.   :  2.00   Min.   : 0.00   Min.   :   0.010   Min.   :  25.00
##  1st Qu.: 13.00   1st Qu.: 7.25   1st Qu.:   0.258   1st Qu.:  97.75
##  Median : 42.00   Median :18.00   Median :   2.590   Median : 192.00
##  Mean   : 85.23   Mean   :26.10   Mean   : 261.709   Mean   : 368.03
##  3rd Qu.: 96.00   3rd Qu.:32.25   3rd Qu.:  59.237   3rd Qu.: 435.25
##  Max.   :444.00   Max.   :95.00   Max.   :4669.320   Max.   :1707.00
##     Nearest           Scruz           Adjacent
##  Min.   : 0.20   Min.   :  0.00   Min.   :   0.03
##  1st Qu.: 0.80   1st Qu.: 11.03   1st Qu.:   0.52
##  Median : 3.05   Median : 46.65   Median :   2.59
##  Mean   :10.06   Mean   : 56.98   Mean   : 261.10
##  3rd Qu.:10.03   3rd Qu.: 81.08   3rd Qu.:  59.24
##  Max.   :47.40   Max.   :290.20   Max.   :4669.32
```

Adjacent and Area seem to be the variables of biggest concern.

```
plot(gala$Area, gala$Adjacent, cex=10*diag(H))
```

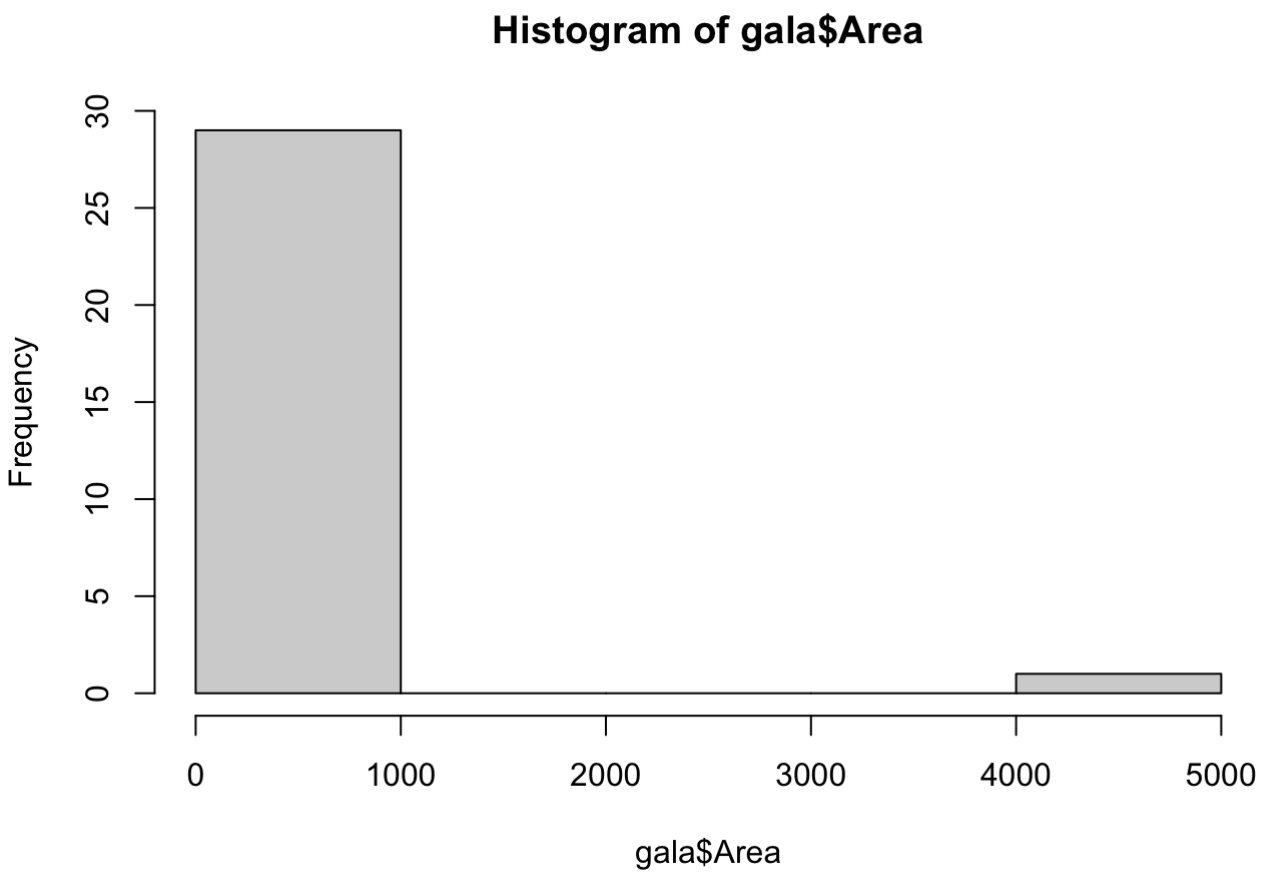


So the biggest leverage points are the observations with the very high Area and Adjacent entries.

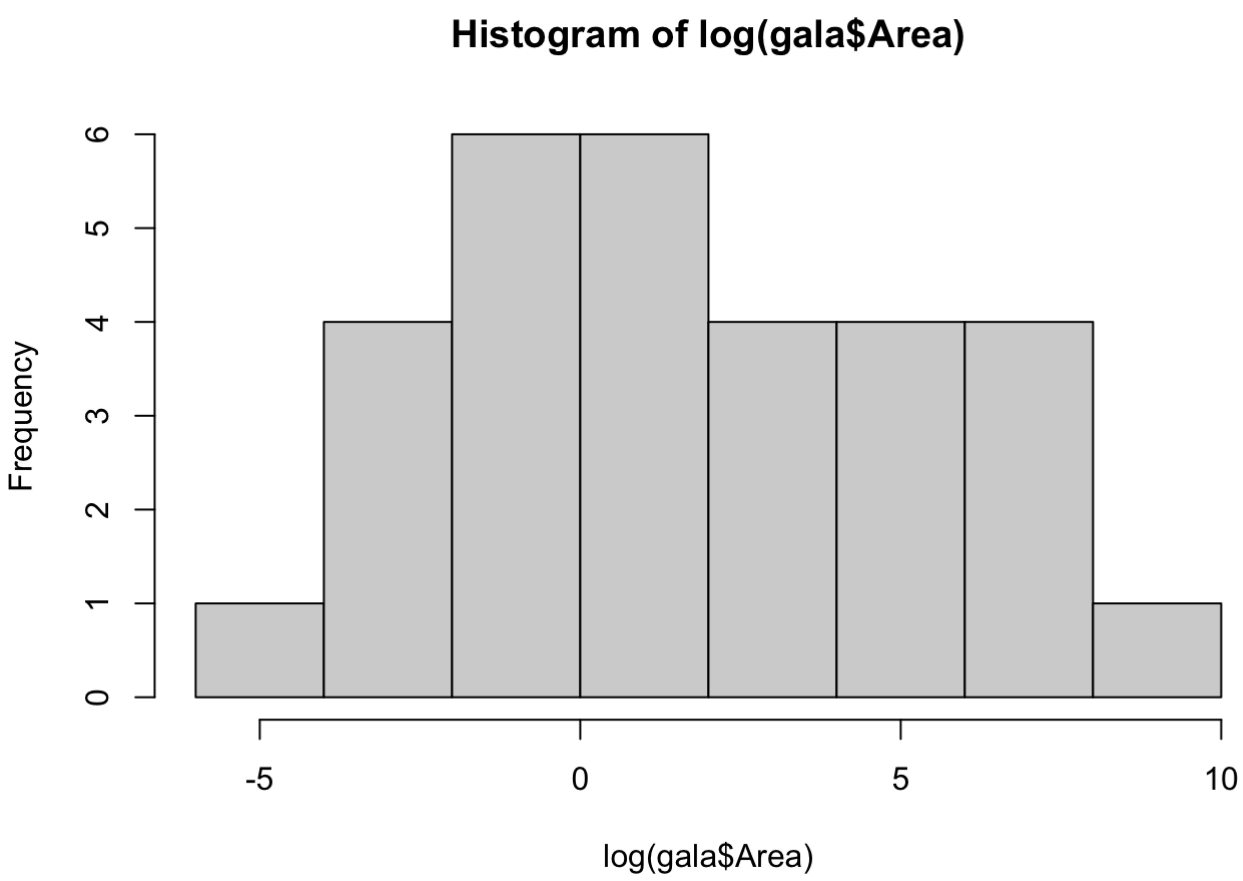
One way to mitigate this phenomenon is to apply a log transformation.

Let's observe why:

```
hist(gala$Area)
```




```
hist(log(gala$Area))
```



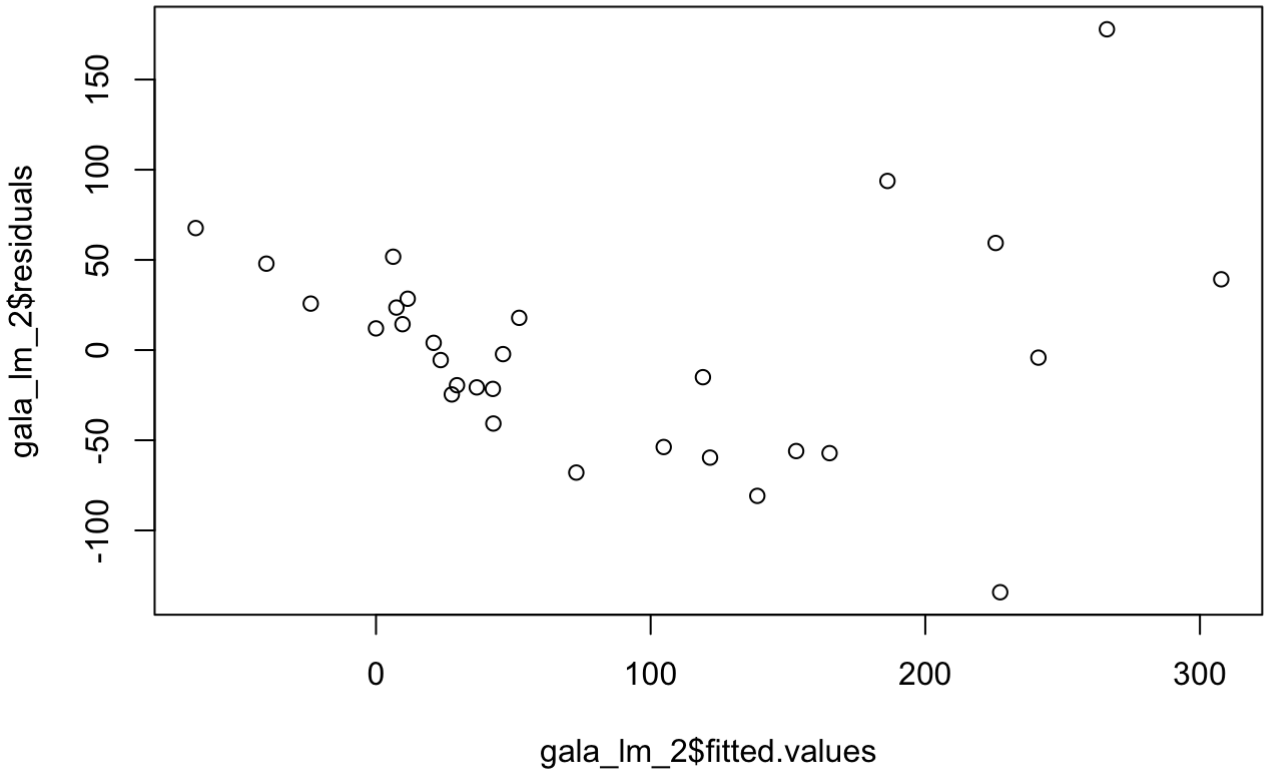
Applying a log transformation is a good way to reduce the impact of outliers in a linear model. Just using the variable as is won't allow for "diminishing returns." Should Area = 5000 have 10x the effect of Area = 500?

The only thing to be concerned about with log transforms is that it is undefined for negative values unstable while approaching zero. One way around this is to do a log(c+variable) transformation, where c is a positive constant.

Let's see how applying these log transforms affects our diagnostics:

```
gala_lm_2 = lm(Species ~ log(Area) + Elevation + Nearest + log(Adjacent), data=gala)

plot(gala_lm_2$fitted.values, gala_lm_2$residuals)
```

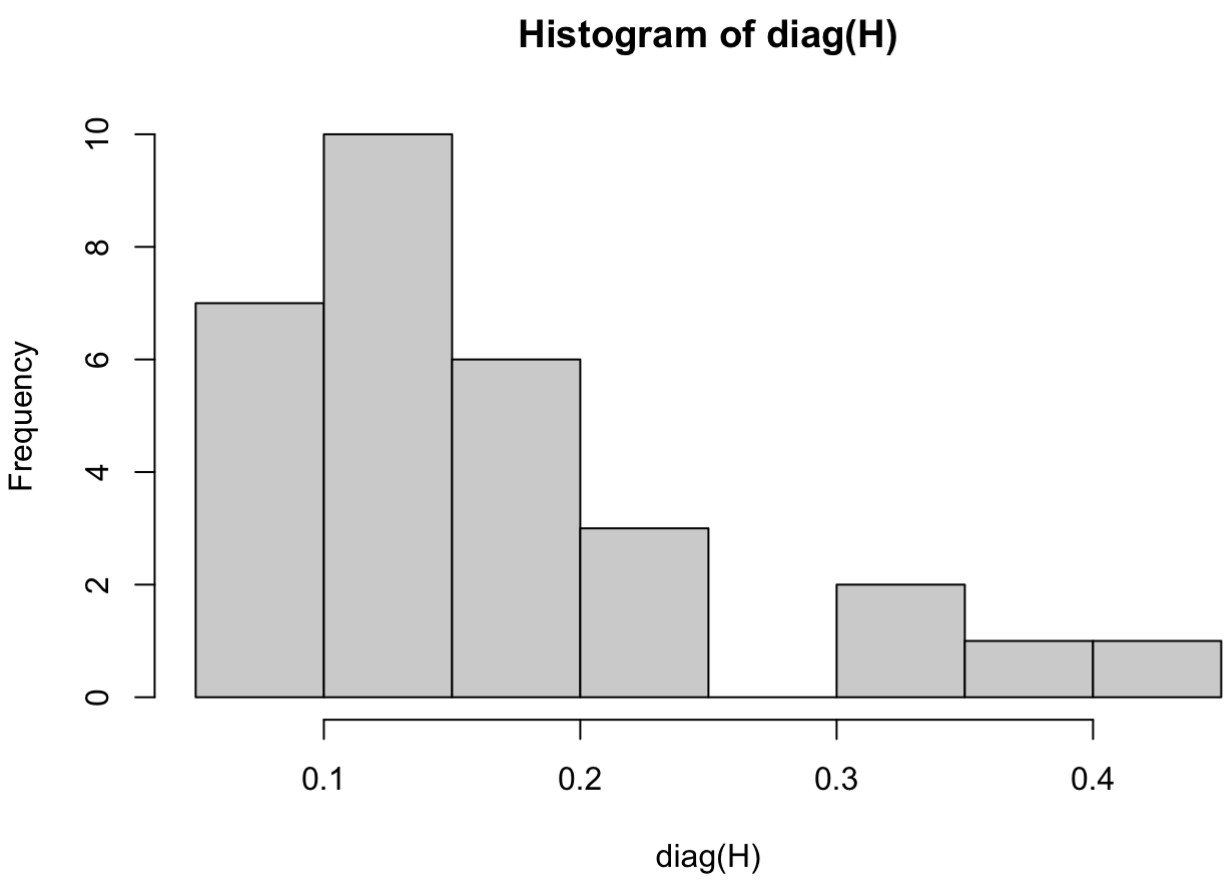


Our residuals versus fitted values chart still looks suspicious: non-linear trend?

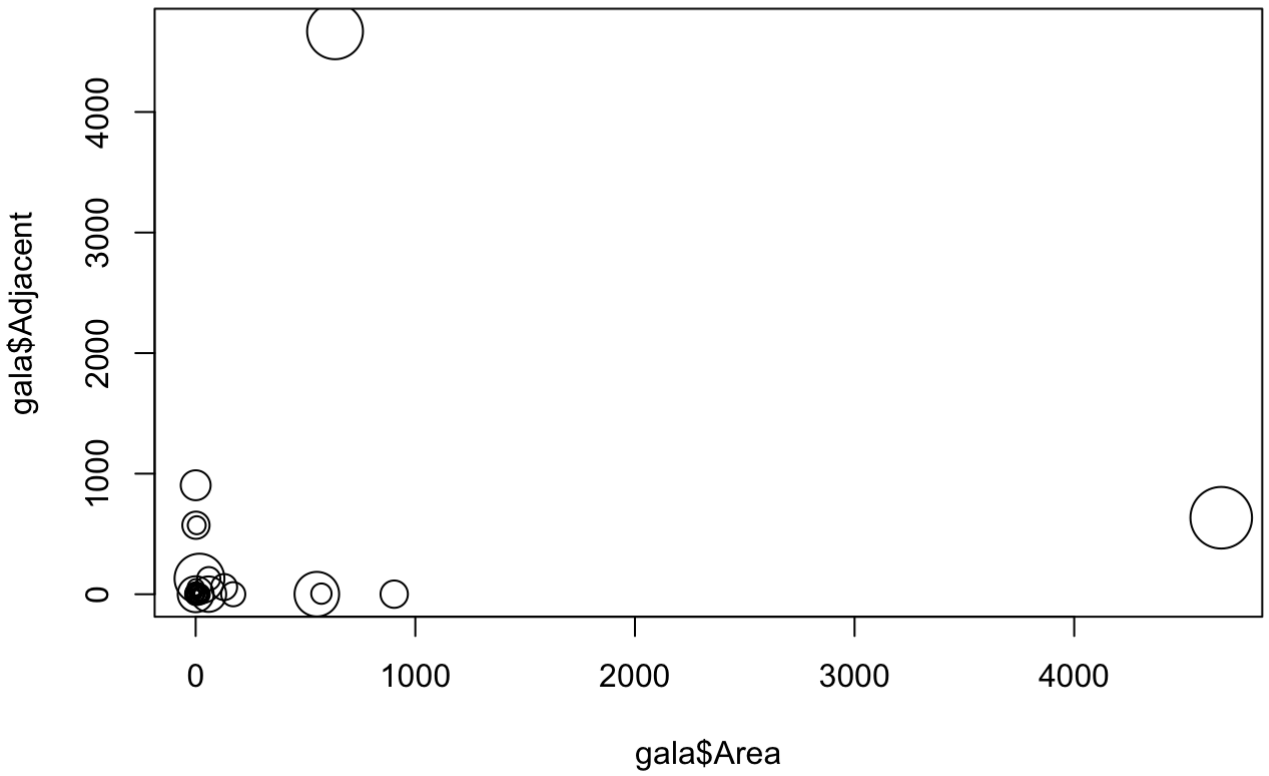
Let's see how the leverage scores are doing nonetheless:

```
X = model.matrix(gala_lm_2)

H = X %*% solve(t(X) %*% X, t(X))
hist(diag(H))
```




```
plot(gala$Area, gala$Adjacent, cex=10*diag(H))
```



It looks a lot better! No longer are those couple points determining the entire line of best fit.

Check-in 1

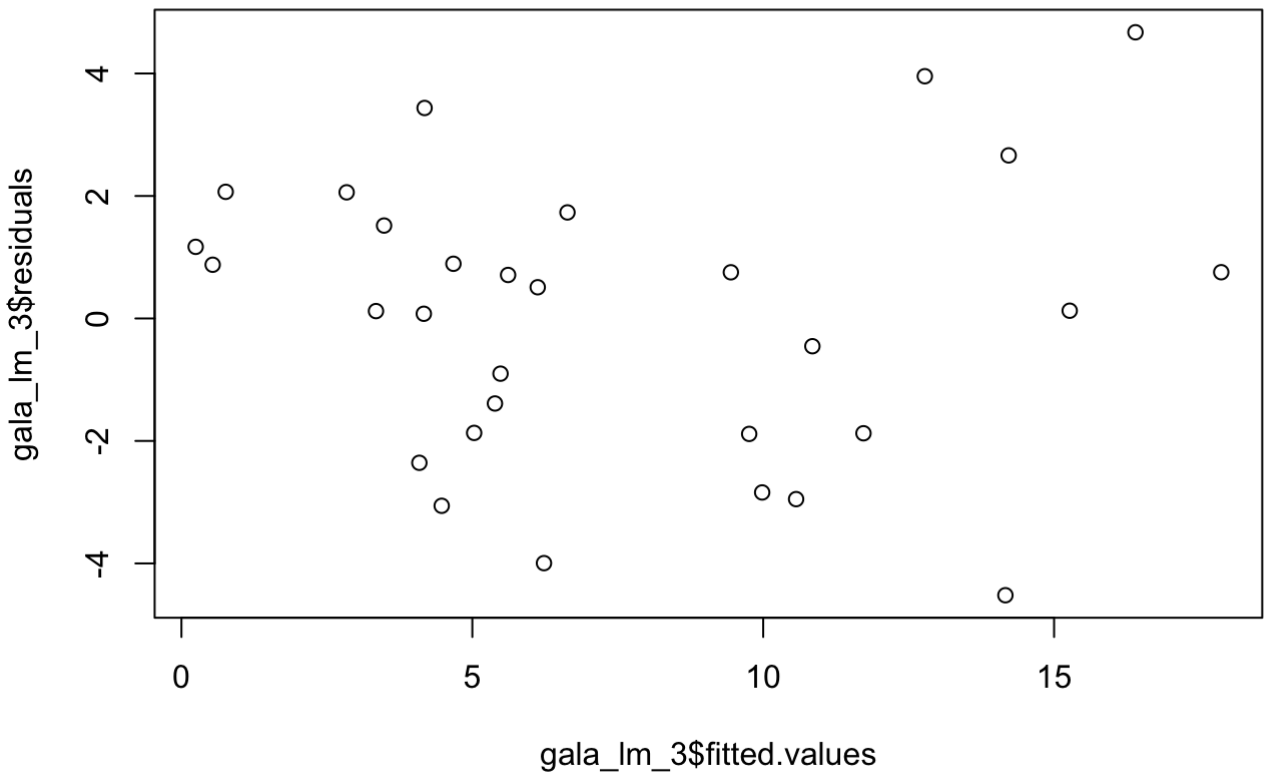
Try transforming the response scatterplot appearance. The main options are “sqrt”, “log”, and “^2”.

```
## add in your attempt here
```

Check-in 1 Solution

```
gala_lm_3 = lm(sqrt(Species) ~ log(Area) + Elevation + Nearest + log(Adjacent), data=gal
a)

plot(gala_lm_3$fitted.values, gala_lm_3$residuals)
```

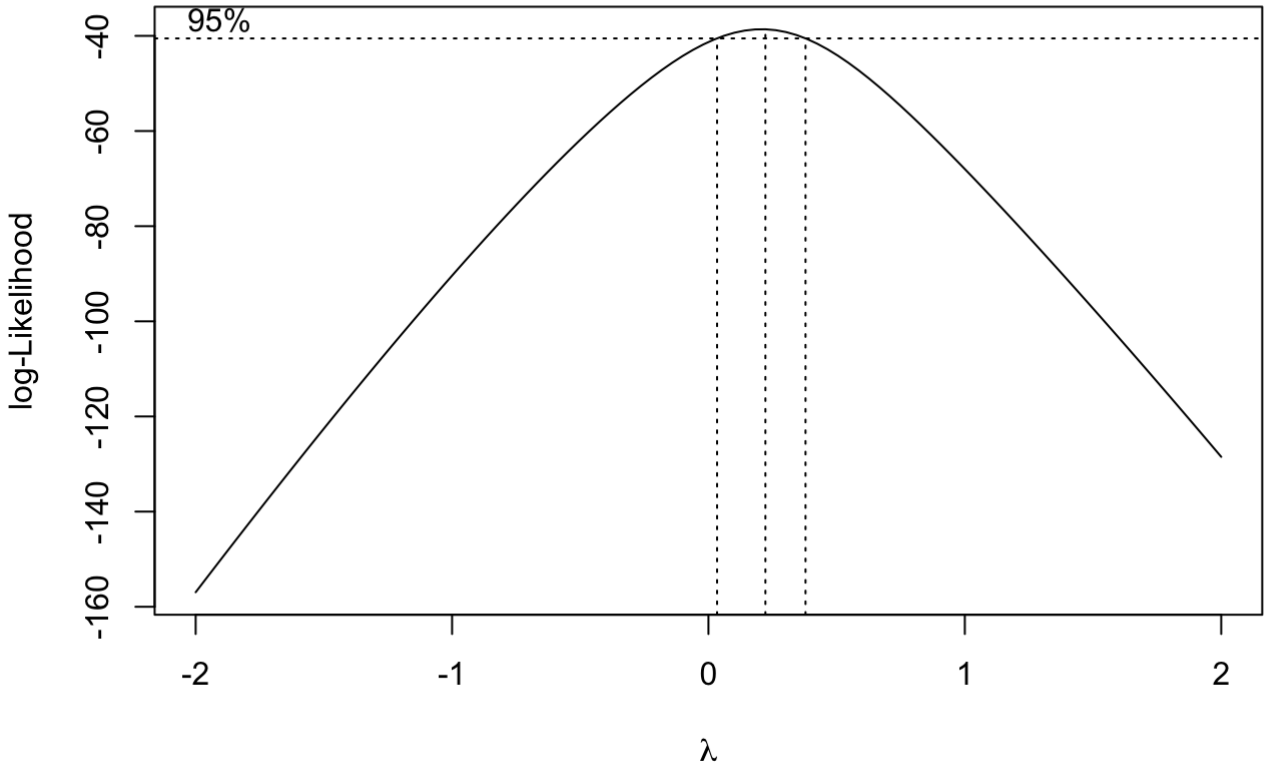


Perfect!

Box-Cox

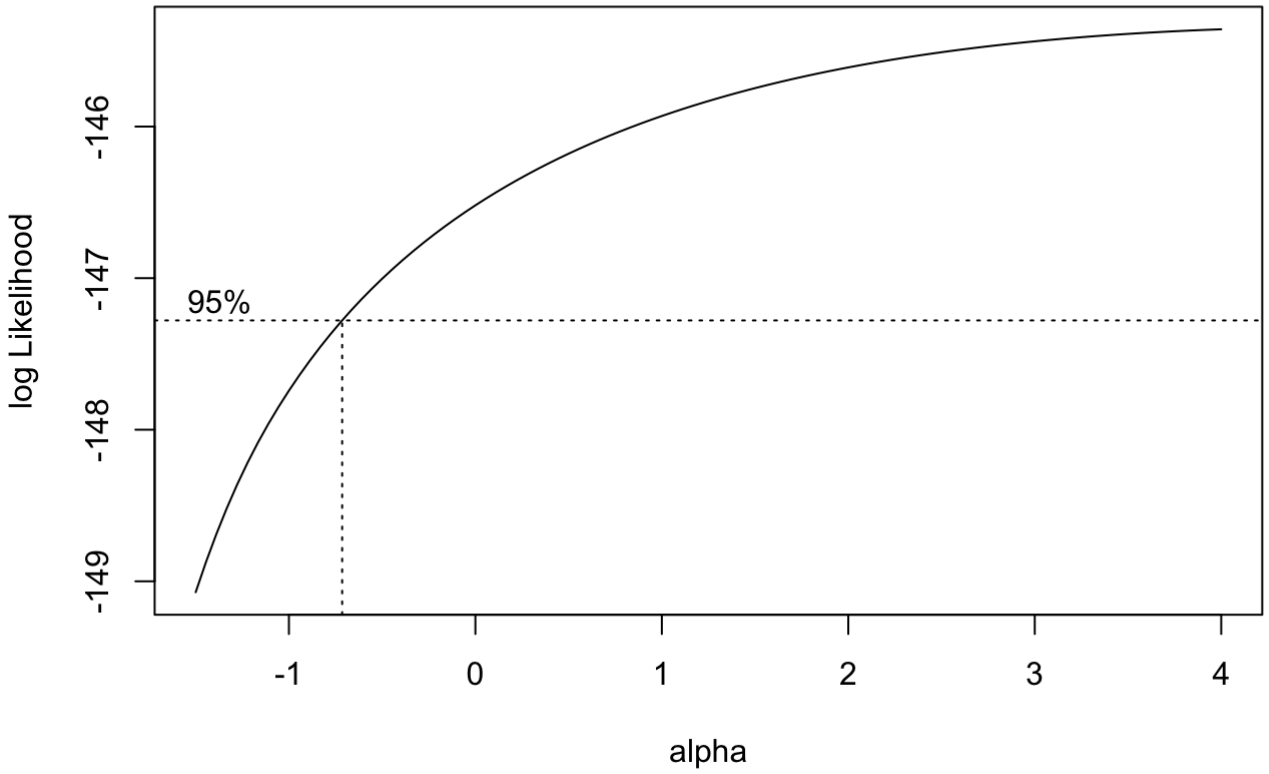
Suppose we want to raise the response to some power λ. Using the boxcox function on the linear model (of the untransformed response) will tell us the value of λ that maximizes the log-likelihood. As λ goes to 0, we Box-Cox approaches the log transformation.

```
# install.packages("MASS")
library(MASS)
boxcox(gala_lm_2)
```



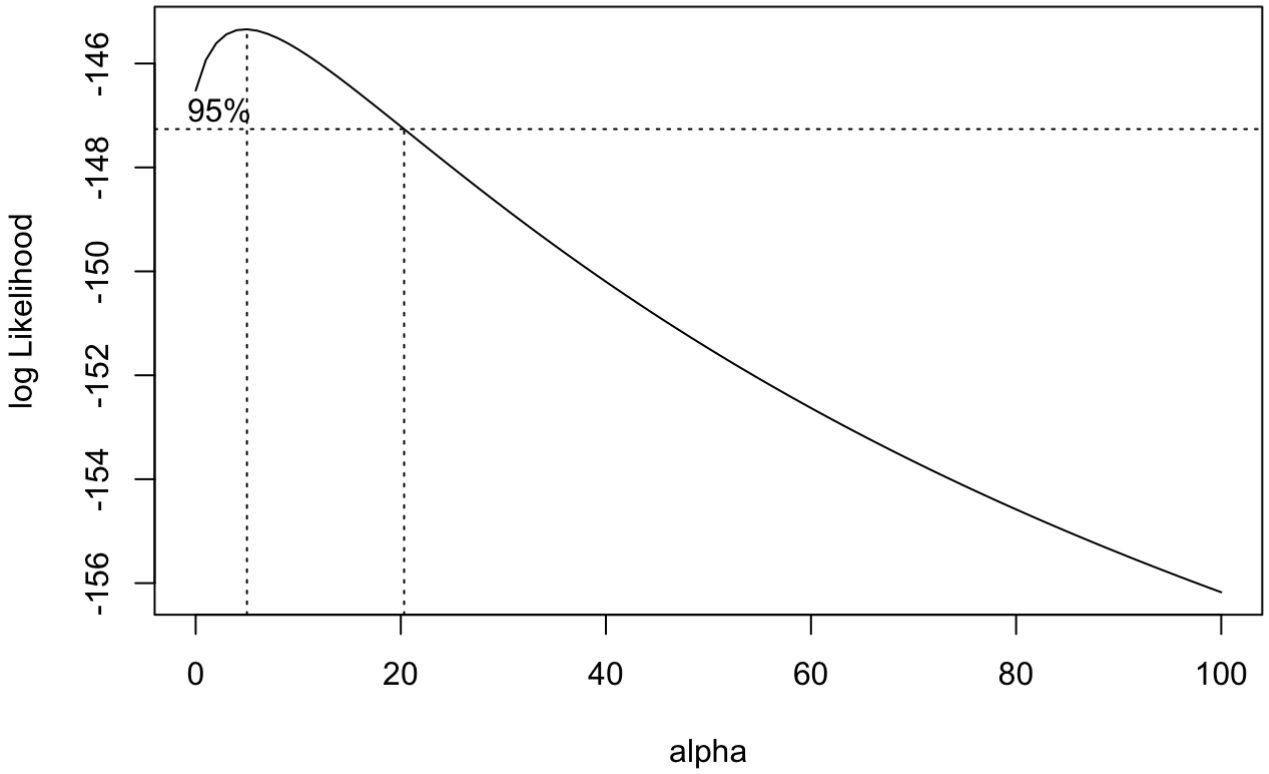
Suppose we wanted to do a $log(\alpha + y_i)$ and wanted to know the best α. We can find this via the logtrans function:

```
logtrans(gala_lm_2)
```



We don't see the likelihood peak exactly, so let's expand the axes:

```
logtrans(gala_lm_2, alpha=0:100)
```



Polynomial terms

Suppose the true model is $y = 5 * x^2 + -x + 8$. If we just regress y on x, then we will struggle to fit the data. However, there is nothing stopping us from regressing y on BOTH x and x^2 (if we know in advance that the square term might be necessary).

Similarly, it is possible that home prices don't just depend on number of bathrooms and number of bedrooms – perhaps it depends on the *interaction* of number of bathrooms and bedrooms. One way to incorporate this in a regression is to regress home price on bedrooms, bathrooms, and bedrooms TIMES bathrooms.

Let's try this out on the gala dataset:

```
gala_lm_3 = lm(sqrt(Species) ~ poly(log(Area),2, raw=TRUE) + poly(Elevation,2, raw=TRUE)
+ Nearest + log(Adjacent) + log(Area)*Elevation,
              data=gala)
summary(gala_lm_3)
```

```
##
## Call:
## lm(formula = sqrt(Species) ~ poly(log(Area), 2, raw = TRUE) +
##     poly(Elevation, 2, raw = TRUE) + Nearest + log(Adjacent) +
##     log(Area) * Elevation, data = gala)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1906 -1.0522  0.0307  1.0761  2.4810
##
## Coefficients: (2 not defined because of singularities)
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     5.619e+00  1.021e+00   5.502 1.58e-05 ***
## poly(log(Area), 2, raw = TRUE)1  3.145e-01  3.076e-01   1.022 0.317792
## poly(log(Area), 2, raw = TRUE)2 -9.954e-02  8.722e-02  -1.141 0.266005
## poly(Elevation, 2, raw = TRUE)1  1.708e-03  4.460e-03   0.383 0.705390
## poly(Elevation, 2, raw = TRUE)2 -2.098e-05  5.370e-06  -3.907 0.000757 ***
## Nearest                         -4.315e-02  2.748e-02  -1.570 0.130579
## log(Adjacent)                   -2.860e-02  1.270e-01  -0.225 0.823825
## log(Area)                              NA         NA      NA       NA
## Elevation                              NA         NA      NA       NA
## log(Area):Elevation              5.271e-03  1.592e-03   3.312 0.003172 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.829 on 22 degrees of freedom
## Multiple R-squared:  0.9133, Adjusted R-squared:  0.8858
## F-statistic: 33.12 on 7 and 22 DF,  p-value: 2.933e-10
```

```
head(poly(gala$Area, 2, raw=TRUE))
```

```
##          1        2
## [1,] 25.09 629.5081
## [2,]  1.24   1.5376
## [3,]  0.21   0.0441
## [4,]  0.10   0.0100
## [5,]  0.05   0.0025
## [6,]  0.34   0.1156
```

```
head(gala$Area)
```

```
## [1] 25.09  1.24  0.21  0.10  0.05  0.34
```

While this sort of adhoc approach is okay, there is a quick way to include all two-way interactions:

```
gala_lm_4 = lm(sqrt(Species) ~ (log(Area) + Elevation + Nearest + log(Adjacent))^2,
               data=gala)
summary(gala_lm_4)
```

```
##
## Call:
## lm(formula = sqrt(Species) ~ (log(Area) + Elevation + Nearest +
##     log(Adjacent))^2, data = gala)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4371 -1.0247  0.2525  0.7804  3.0283
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              5.5085039  1.2033988   4.577 0.000206 ***
## log(Area)                0.8014927  0.2798851   2.864 0.009941 **
## Elevation               -0.0019381  0.0064263  -0.302 0.766238
## Nearest                 -0.0977756  0.0554701  -1.763 0.094035 .
## log(Adjacent)            0.2544941  0.2334956   1.090 0.289374
## log(Area):Elevation      0.0017043  0.0006140   2.776 0.012046 *
## log(Area):Nearest       -0.0137785  0.0199620  -0.690 0.498391
## log(Area):log(Adjacent)  0.0143194  0.0768037   0.186 0.854075
## Elevation:Nearest        0.0002328  0.0001721   1.352 0.192150
## Elevation:log(Adjacent) -0.0015301  0.0006022  -2.541 0.019941 *
## Nearest:log(Adjacent)    0.0083069  0.0128376   0.647 0.525323
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.759 on 19 degrees of freedom
## Multiple R-squared:  0.9308, Adjusted R-squared:  0.8944
## F-statistic: 25.56 on 10 and 19 DF,  p-value: 6.272e-09
```

We can do a single test of whether ALL the interaction terms are irrelevant. This is called an F-test:

```
anova(gala_lm_4)
```

```
## Analysis of Variance Table
##
## Response: sqrt(Species)
##                          Df Sum Sq Mean Sq  F value    Pr(>F)
## log(Area)                 1 643.59  643.59 207.9993 1.096e-11 ***
## Elevation                 1   6.07    6.07   1.9606  0.177564
## Nearest                   1  21.94   21.94   7.0895  0.015379 *
## log(Adjacent)             1  21.38   21.38   6.9088  0.016547 *
## log(Area):Elevation       1  11.35   11.35   3.6670  0.070678 .
## log(Area):Nearest         1  11.51   11.51   3.7203  0.068830 .
## log(Area):log(Adjacent)   1  44.05   44.05  14.2375  0.001286 **
## Elevation:Nearest         1   1.71    1.71   0.5539  0.465841
## Elevation:log(Adjacent)   1  27.94   27.94   9.0312  0.007278 **
## Nearest:log(Adjacent)     1   1.30    1.30   0.4187  0.525323
## Residuals                19  58.79    3.09
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interpretation: we should include the interaction terms.

Just be careful: if we try enough transformations, one of them is bound to look good – even if we are totally misspecifying the model.

Check in 3: Why can R-squared be a misleading metric?

The following chunk of code returns simulated data with a very high R-squared. Can you make the R-squared small by altering any of the numbers? Tell us what you changed in Slack.

```
set.seed(42)

n=1000
p=5
X = matrix(rnorm(n*p), ncol=p) # simulate p covariates
b = rnorm(p, mean=5, sd=20) # randomly draw the 20 true coefficients that determine y
y = X %*% b + rnorm(n, sd=1) # Define y to be the linear model plus noise

summary(lm(y ~ X))
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9413 -0.7413  0.0195  0.7241  3.2926
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.01988    0.03294   0.603    0.546
## X1           6.41635    0.03286 195.265   <2e-16 ***
## X2          24.44045    0.03346 730.353   <2e-16 ***
## X3          11.22505    0.03201 350.675   <2e-16 ***
## X4           2.17123    0.03336  65.090   <2e-16 ***
## X5          -1.52811    0.03237 -47.206   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.041 on 994 degrees of freedom
## Multiple R-squared:  0.9986, Adjusted R-squared:  0.9986
## F-statistic: 1.398e+05 on 5 and 994 DF,  p-value: < 2.2e-16
```

By varying the standard deviation of the noise, we can make our R-squared arbitrarily bad, even if we correctly specify the linear relationship!

Takeaway: You can have a low R-squared even with a correctly specified model. Moreover, it is possible for a misspecified model to have a higher R^2 by just happening to fit the random noise well!

Simulation to show that you can p-hack by using variable selection.

```
set.seed(42)
n=100
p=50
X = matrix(rnorm(n*p), ncol=p)
y = rnorm(n)
```

```
max_cor = max(abs(cor(y,X)))
max_cor
```

```
## [1] 0.2140486
```

```
idx_max_cor = which(abs(cor(y,X)) == max_cor)
idx_max_cor
```

```
## [1] 39
```

```
cor(y,X[,idx_max_cor])
```

```
## [1] -0.2140486
```

```
summary(lm(y ~ X[,idx_max_cor]))
```

```
##
## Call:
## lm(formula = y ~ X[, idx_max_cor])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0586 -0.6768  0.1323  0.6157  2.2046
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -0.10719    0.10169  -1.054   0.2944
## X[, idx_max_cor] -0.21667    0.09988  -2.169   0.0325 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.013 on 98 degrees of freedom
## Multiple R-squared:  0.04582,    Adjusted R-squared:  0.03608
## F-statistic: 4.706 on 1 and 98 DF,  p-value: 0.03248
```

Woah! We got got a very small p-value, which suggests this column of X is related to the response. We know this is fiction, since in our simulation they were drawn independently!

The holdout set approach

What if we had only calculated the correlations on the first half of the dataset, and then ran the regression on the second half?

```
max_cor_train = max(abs(cor(y[1:n/2],X[1:n/2,])))
max_cor_train
```

```
## [1] 0.3390454
```

```
idx_max_cor_train = which(abs(cor(y[1:n/2],X[1:n/2,])) == max_cor_train)
idx_max_cor_train
```

```
## [1] 2
```

```
summary(lm(y[n/2:n] ~ X[n/2:n,idx_max_cor_train]))
```

```
##
## Call:
## lm(formula = y[n/2:n] ~ X[n/2:n, idx_max_cor_train])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.4046 -0.1452 -0.1452  0.2880  1.1375
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  0.11053    0.07165   1.543    0.126
## X[n/2:n, idx_max_cor_train]  0.08822    0.06519   1.353    0.179
##
## Residual standard error: 0.5215 on 97 degrees of freedom
## Multiple R-squared:  0.01853,    Adjusted R-squared:  0.008408
## F-statistic: 1.831 on 1 and 97 DF,  p-value: 0.1792
```

We can see that, even when selecting the column most correlated on the “training” set, we don’t achieve a significant p-value on the holdout set. This is what we would’ve expected, given how we constructed our simulation.

Bonferroni correction

Take your significance level (e.g. 5%) and divide it by the number of tests you’ve performed on the same dataset. This is your new threshold for p-values to be significant.

Note: Bonferroni correction is “conservative”: while it upper bounds your Type I error at 5%, it may end up being far below 5%.