BIOF339: Lecture 5

Eugen Buehler

October 16, 2016

Multiple Comparison Procedures

- A p-value from a statistical test is an estimate of the probability of the data occurring under the null hypothesis
- Although a p-value of 0.05 or 0.01 is often treated as statistically significant, if we do 1000 statistical tests some things will appear significant by chance (50 or 10 respectively).
- We need to employ a multiple comparison procedure to keep from being fooled when we are doing lots if statistical tests

Bonferonni Correction

- The simplest multiple comparison procedure
- If we are doing n tests, we can multiply the p-values generated by n (or equivalently divide the significance α level by n). Then the probability that any of the tests will generate a false positive will be α.
- While simple to implement and understand, the Bonferonni correction is overly conservative.

False Discovery Rate

- False Discovery Rate: The expected proportion of false positives (incorrectly rejected null hypotheses).
- For example, if we set our false discovery rate to 0.05 and we identify 100 positives (cases where we will reject the null hypothesis), 5 of those 100 will be incorrect (only by chance).

Benjamini–Hochberg Procedure

- Controls false discovery rate.
- Orders p-values from least to most significant and then judges their significance on a sliding scale.
- Like Bonferroni, can be thought of as a modification to the p-values instead of an adjustment to the significance threshold.
- p-values modified to control for false discovery rate are sometimes called "q-values".

10 Heads in a Row

```
> prop.test(10,10,0.5)
```

```
    1-sample proportions test with continuity correction

data:  10 out of 10, null probability 0.5
X-squared = 8.1, df = 1, p-value = 0.004427
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.6554628 1.0000000
sample estimates:
p
1
```

Searching for a biased coin

Let's say that we hear reports of biased coins circulating in the country. We gather together 20000 coins and flip each of them 10 times, recording the number of heads we get for each coin. We can simulate this in r using the rbinom function (random binomial). Then we can use the proportion test together with sapply (covered in later lectures) to calculate a p-value for each simulated coin.

```
> coinTable <- data.frame(heads=rbinom(n = 20000,size=10,prob=0.5))
> coinTable$p.value <- sapply(coinTable$heads,
+                             function(x) prop.test(x,10,0.5)$p.value )
> sum(coinTable$p.value < 0.01)
```

```
[1] 34
```

Using p.adjust

p.adjust takes a vactor of p-values and applies a multiple comparison procedure. For example:

```
> coinTable$bonferonni <- p.adjust(coinTable$p.value,
+                                  method = "bonferroni")
> coinTable$q.value <- p.adjust(coinTable$p.value, method="fdr")
> sum(coinTable$bonferonni < .01)
```

```
[1] 0
```

```
> sum(coinTable$fdr < .1)
```

```
[1] 0
```

Models in Science

- All models are wrong, but some are useful.
- Given this assumption, we want to pick not the right or wrong model, but the one that is most useful.
- So what makes a model useful?

Uses for Models

- Establishing relationships (eg. people who smoke are x times more at risk for lung cancer)
- Making predictions (eg. this site has a x% chance of being a functional binding site for this protein)

The Spectrum of Data Models

- Highly Interpretable Models
- Can be made with relatively small data sets.
- It is easy to see how the model makes predictions.
- The "weight" and significance of individual variables in the model is obvious. – Examples include linear regression and decision trees.
- Opaque or "Black Box" Methods – Require large sets of data. – Little to know ability to see how the method makes predictions. – Can be highly accurate. – Examples include Support Vector Machines, Neural Networks, and Random Forests

Our first model in R

To explore building models in R, let's build a linear model of the cholesterol based on bilirubin from the pbc dataset in the survival package.

```
> library(survival)
```

```
Warning: package 'survival' was built under R version 3.2.5
```

```
> myLinearModel <- lm(chol ~ bili, data=pbc)
```

Note that R doesn't tell us anything about the model we have built, it just makes the model and puts it in the variable "myLinearModel". Nor does R know whether it makes any sense to model cholesterol based on bilirubin, or what it would mean if there was a statistically significant relationship between the two.

Formula Interface in R

Recall that we saw the "~" previously in some of our statistical tests, where we could do certain statistical tests with shortened notation (eg t.test(x ~ y, data=myData)). This formula interface is used multiple times in R. Briefly, the way we write model formulas in R is:

- y ~ x means a model of y based on x.
- y ~ x1 + x2 means a model of y based on a linear combination of x1 and x2
- y ~ x1 + x2 + x1:x2 means that we include an interaction term between x1 and x2
- y ~ x1 * x2 means the same as the line above, it is a shorthand for including interaction terms
- y ~ . means use everything in the data set (other than y) to predict y

Summary command

```
> summary(myLinearModel)
```

```
Call:
lm(formula = chol ~ bili, data = pbc)

Residuals:
    Min      1Q  Median      3Q     Max
-565.39  -89.90  -35.36   44.92 1285.33

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  303.204     15.601  19.435  < 2e-16 ***
bili          20.240      2.785   7.267 3.63e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 213.2 on 282 degrees of freedom
  (134 observations deleted due to missingness)
Multiple R-squared:  0.1577,    Adjusted R-squared:  0.1547
```

Predict command

The predict command will use a model to make predictions. If new data is not supplied to predict, it will generate predictions for the original data used to generate the model.

```
> myResiduals <- pbc$chol[! is.na(pbc$chol)] - predict(myLinearModel)
> max(myResiduals, na.rm=TRUE)
```

```
[1] 1285.333
```

```
> predict(myLinearModel, newdata = data.frame(bili = c(2,5.7,10)))
```

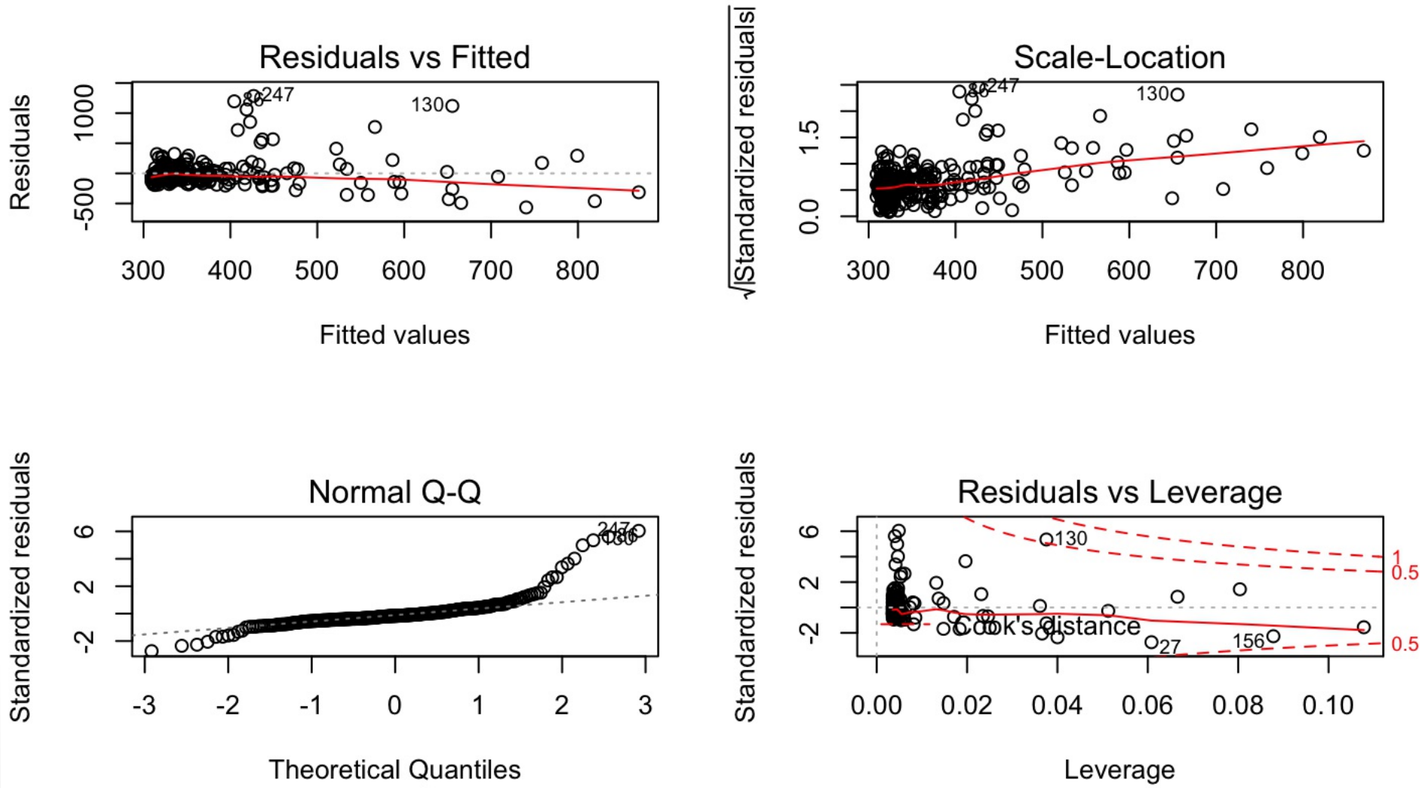
```
       1        2        3
343.6833 418.5709 505.6025
```

Diagnostic Plots

Passing a linear model to the "plot" function will case R to generate 4 different diagnostic plots, prompted by "Hit to see next plot" between each plot. If you're using RStudio, you'll be able to go back to previous plots using the arrows in the "plot" displays. For our purposes, we'll pack the graphs into a single page.

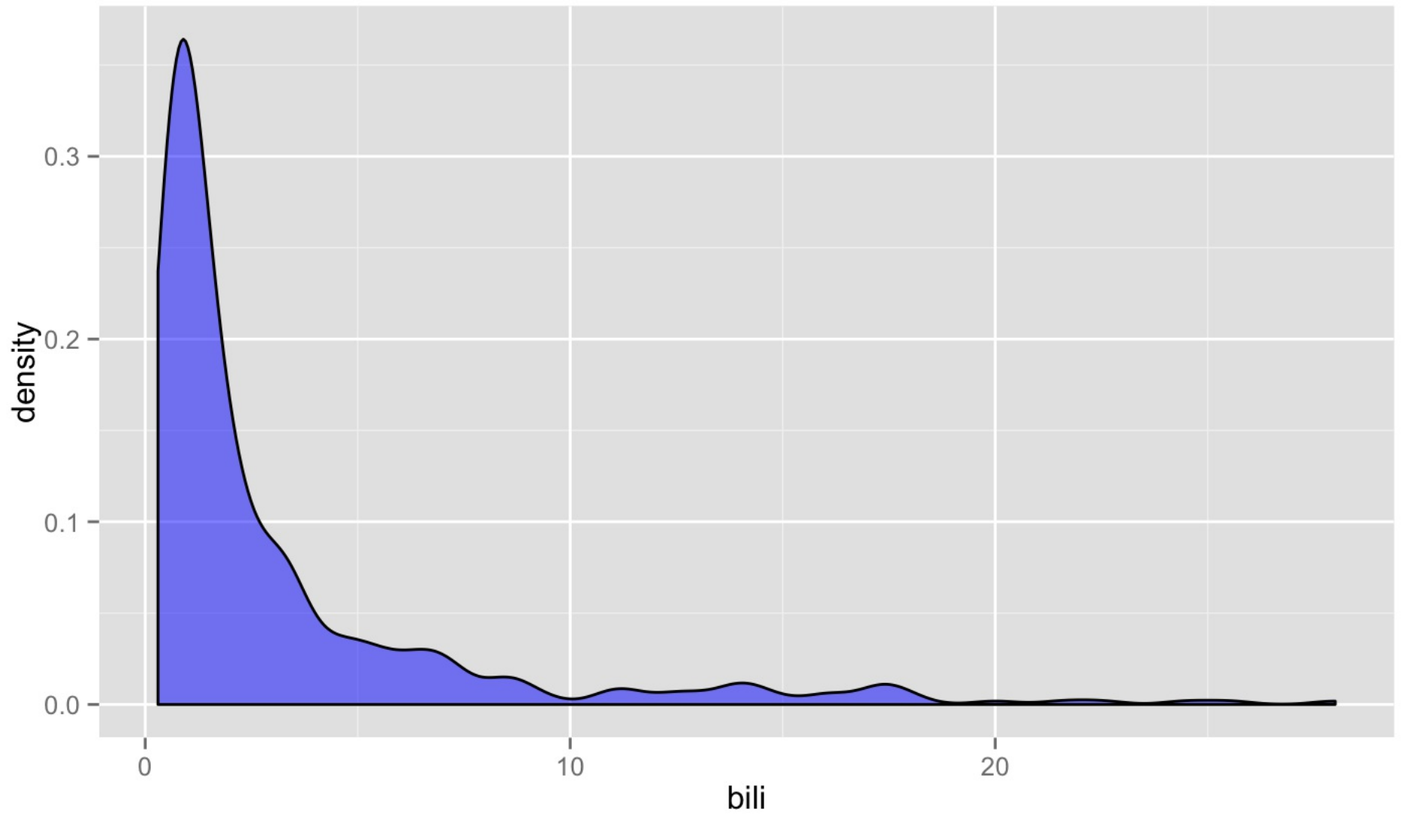
Diagnostic Plots

```
> layout(matrix(c(1,2,3,4),2,2))
> plot(myLinearModel)
```



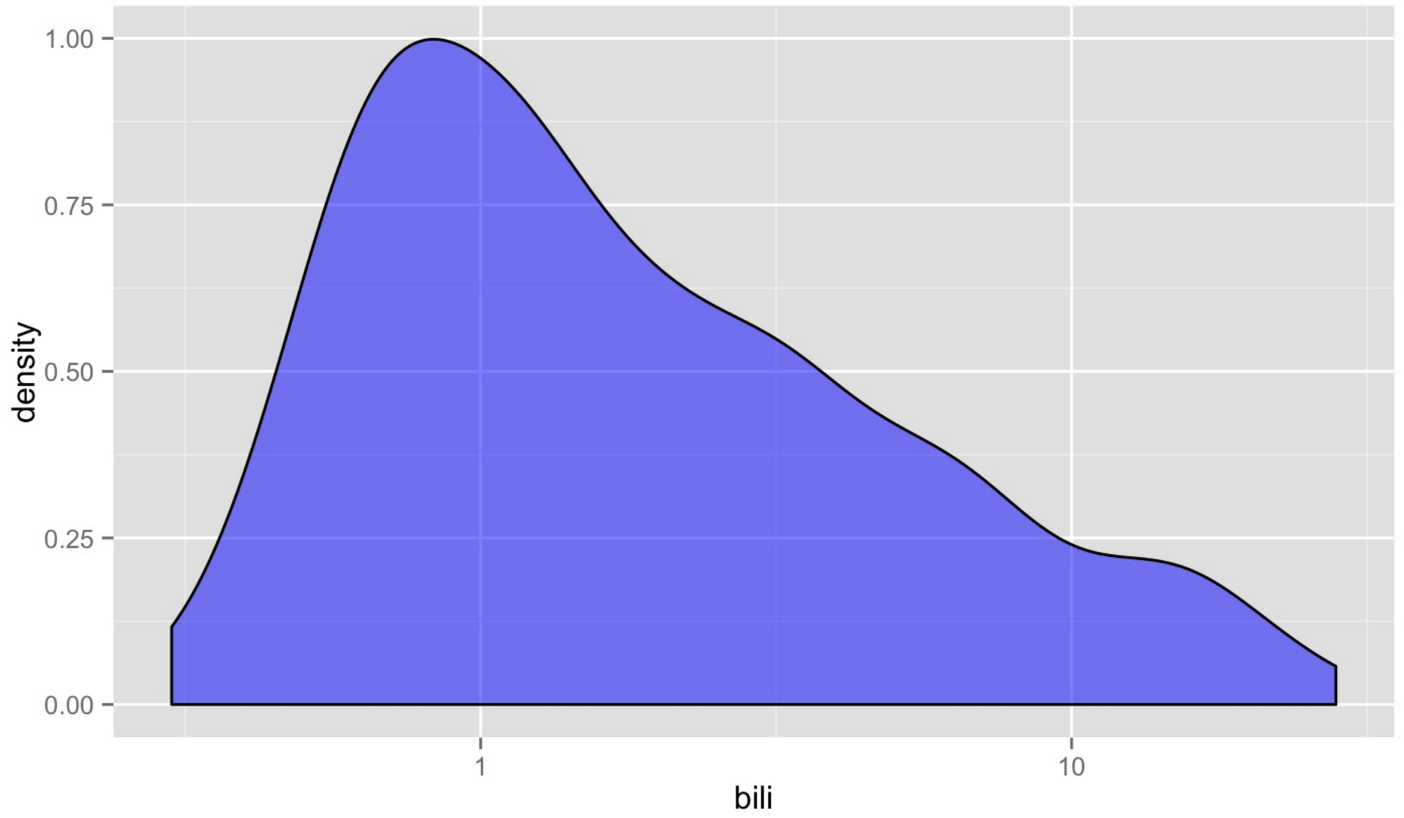
Normality of Variables

```
> library(ggplot2)
> ggplot(pbc, aes(x=bili)) + geom_density(alpha=0.5, fill="blue")
```



Can we "fix" bilirubin?

```
> ggplot(pbc, aes(x=bili)) + geom_density(alpha=0.5, fill="blue") +
+      scale_x_log10()
```



Model with log bilirubin

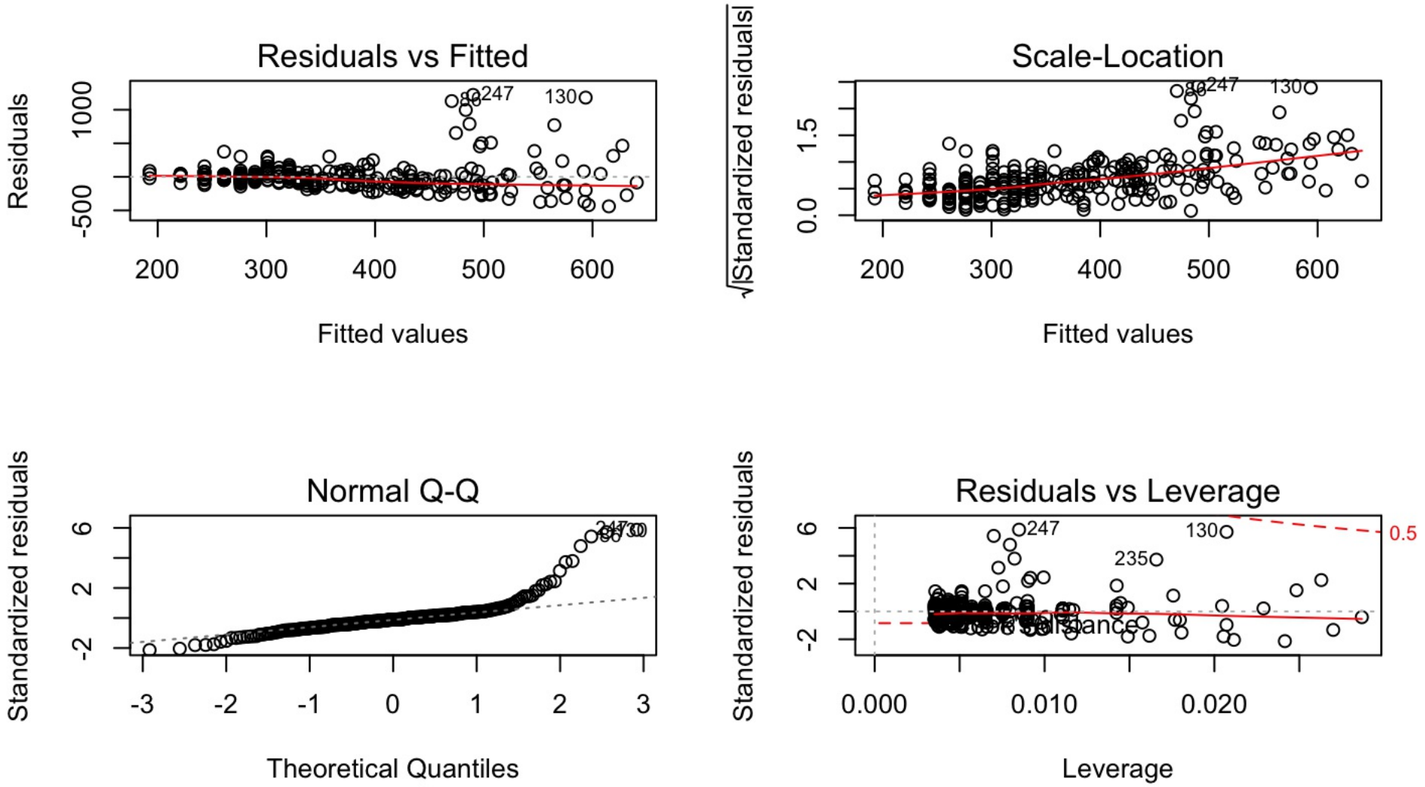
Let's see whether we can generate a better model using the log of bilirubin:

```
> myNewLinearModel <- lm(chol ~ log(bili), data=pbc)
```

Summary of New Model

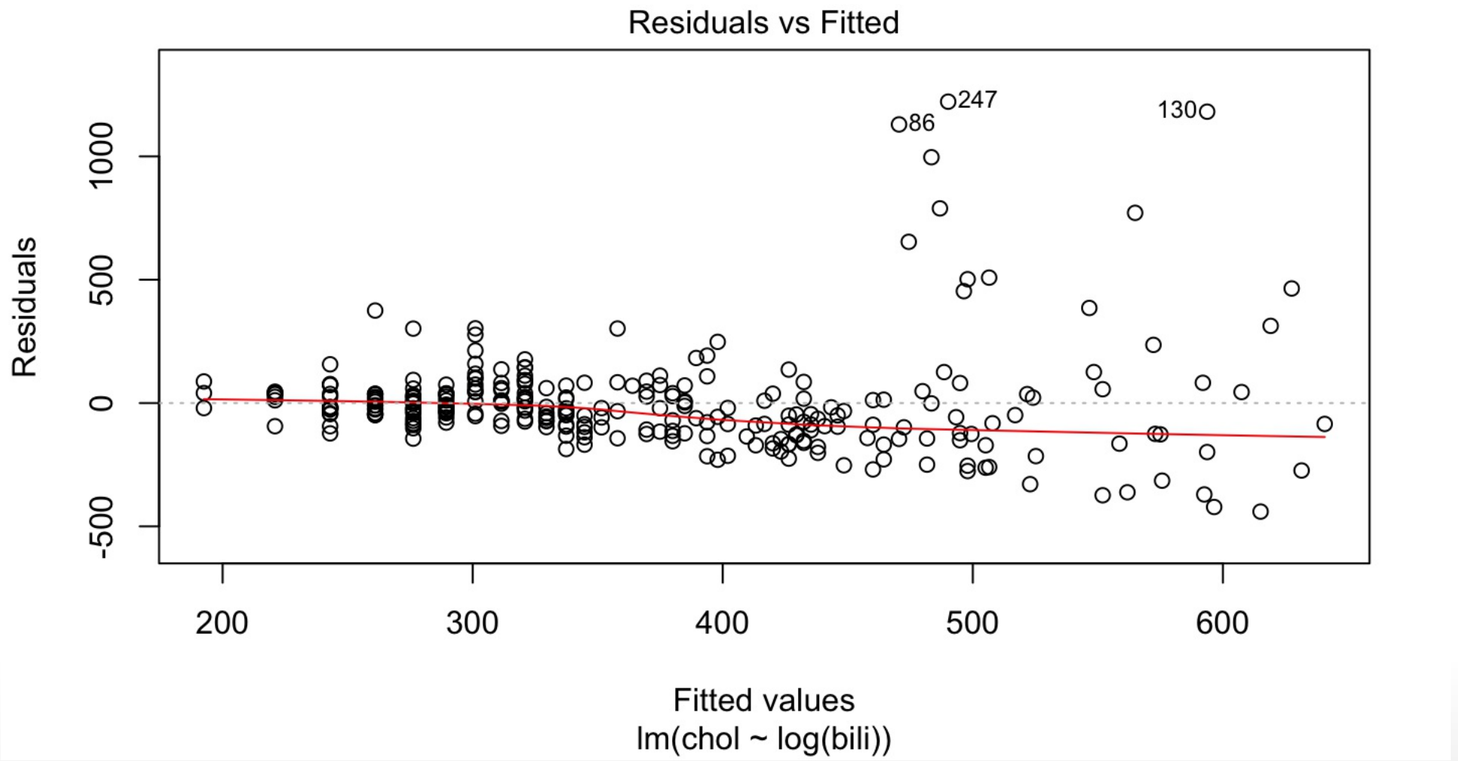
```
Call:
lm(formula = chol ~ log(bili), data = pbc)

Residuals:
    Min      1Q  Median      3Q     Max
-440.07  -94.35  -21.07   42.67 1221.86

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   311.48      14.28  21.816  < 2e-16 ***
log(bili)      98.80      12.07   8.186 9.42e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 208.9 on 282 degrees of freedom
  (134 observations deleted due to missingness)
Multiple R-squared:  0.192, Adjusted R-squared:  0.1891
F-statistic: 67.01 on 1 and 282 DF,  p-value: 9.416e-15
```

Diagnostic Plots



Just the Residuals, Please

```
> plot(myNewLinearModel, which=1)
```



Factors as predictive variables

```
Call:
lm(formula = chol ~ log(bili) + sex, data = pbc)

Residuals:
    Min      1Q  Median      3Q     Max
-446.09  -96.78  -23.01   41.91 1216.86

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   282.55      36.63   7.713 2.14e-13 ***
log(bili)      99.62      12.11   8.224 7.37e-15 ***
sexf           32.45      37.84   0.858    0.392
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 209 on 281 degrees of freedom
  (134 observations deleted due to missingness)
Multiple R-squared:  0.1941,    Adjusted R-squared:  0.1884
F-statistic: 33.84 on 2 and 281 DF,  p-value: 6.793e-14
```

Generalized Linear Models

Generalized Linear Models is a framework for many forms of regression related to linear regression. They function by specifying a transform for the predicted variable and an associated error distribution. Generalized linear models can be used for all kinds of things, including predicting non-normal data like counts (Poisson). However, in practice I have only ever had use of logistic regression.

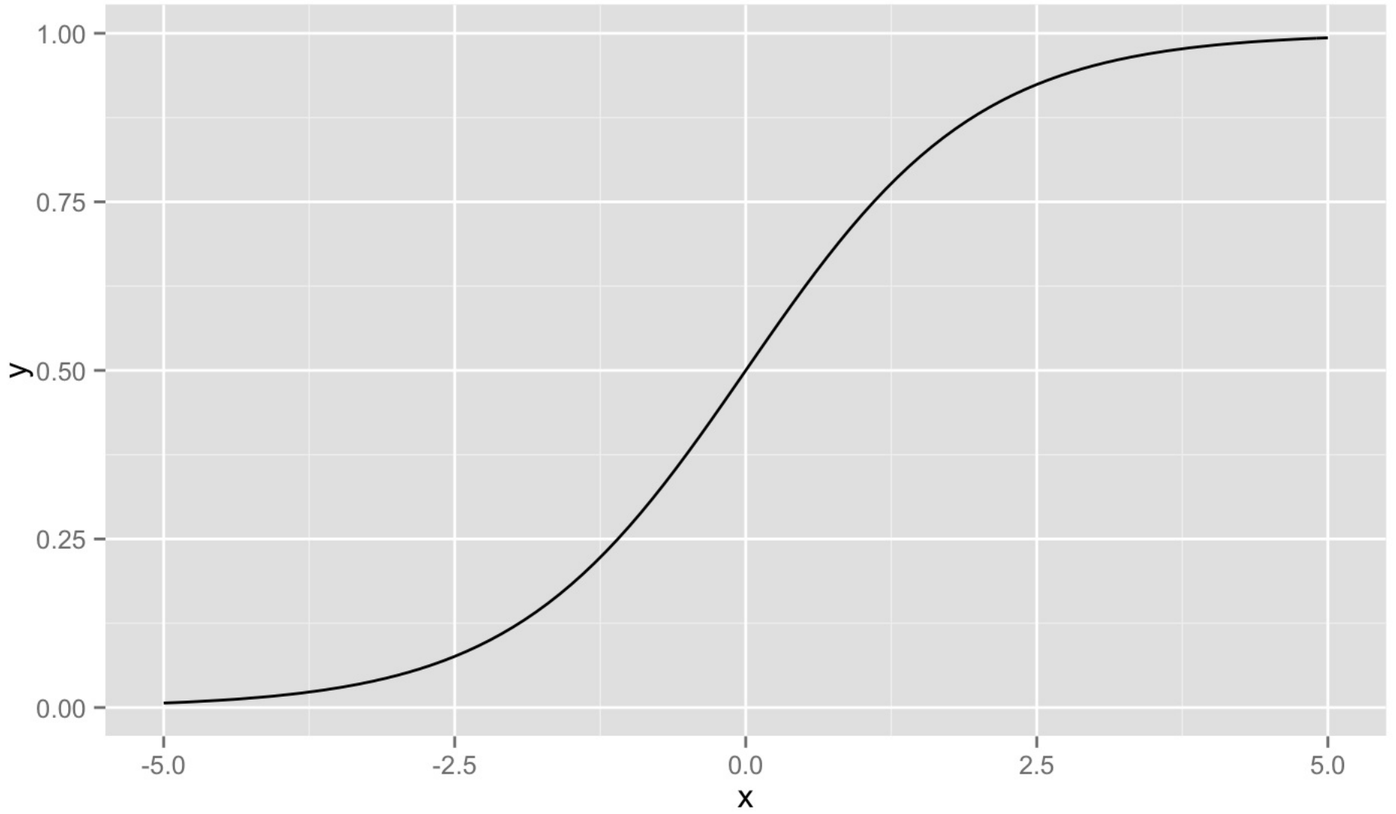
The Logit or Log-odds transform

For data that only has two outcomes, we could encode our predicted variable as a zero or a one. However, what would it mean if our regression generates a "5"? Instead, we can use a mapping of the numbers from negative infinity to positive infinity to 0 to 1, and then any output can be interpreted as a probability.

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Graphing the Logit function

```
> library(ggplot2)
> invlogit <- function(x) exp(x) / (1 + exp(x))
> ggplot(data.frame(x = c(0.5, 0.5)), aes(x = x)) + stat_function(fun = invl
```



Logistic Regression in R

```
Call:
glm(formula = spiders ~ albumin + bili + chol, family = "binomial",
    data = pbc)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0493  -0.7838  -0.6666   0.9396   2.0045

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.3326484  1.2952313   1.801  0.07171 .
albumin     -0.9954927  0.3619294  -2.751  0.00595 **
bili         0.0995915  0.0344236   2.893  0.00381 **
chol        -0.0003176  0.0006147  -0.517  0.60533
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

Prediction from Logit Model

```
> predict(myLogItModel)[1:10]
```

```
          1           2           3           4           5           6
 1.10554163 -1.77506554 -1.04814132 -0.09414055 -0.93144911 -1.62851203
          7           8           9          10
-1.74160268 -1.70838182 -0.59328547  0.79632442
```

Probabilities from Logit Model

```
> library(boot)
```

```
Attaching package: 'boot'

The following object is masked from 'package:survival':

    aml
```

```
> inv.logit(predict(myLogItModel)[1:10])
```

```
        1         2         3         4         5         6         7
0.7512970 0.1449135 0.2595822 0.4764822 0.2826308 0.1640343 0.1491095
        8         9        10
0.1533737 0.3558814 0.6891877
```

Predicted Outcome from Logit Model

```
> round(inv.logit(predict(myLogItModel)[1:10]))
```

```
 1  2  3  4  5  6  7  8  9 10
 1  0  0  0  0  0  0  0  0  1
```

Feature Selection

Feature selection is the process by which predictor variables are selected for a model. This could be done by hand or using some automated method (such as stepwise linear regression or lasso). Feature selection quickly gets brings us back to the issue of multiple comparison corrections, since if we try enough predictive variable eventually one of them will seem to be a statistically significant predictor, even if it is not.

Stepwise regression

Although the idea behind stepwise regression is relatively simple (add/remove the best/worst features one at a time until we stop improving), R's implementation of stepwise regression is cumbersome and counterintuitive. We have to define an initial model and the scope of possible models to consider (if it is larger than the initial model).

```
> myStepwiseModel <- step(lm(chol ~ ., data= pbc[,11:19]))
```

```
Start:  AIC=2938.68
chol ~ bili + albumin + copper + alk.phos + ast + trig + platelet +
    protime

            Df Sum of Sq      RSS    AIC
- albumin    1      6947 10888263 2936.8
- alk.phos   1     44444 10925760 2937.8
<none>                   10881315 2938.7
- trig       1    100834 10982149 2939.2
```

Summary of Stepwise Model

```
Call:
lm(formula = chol ~ bili + copper + ast + trig + platelet + protime,
    data = pbc[, 11:19])

Residuals:
    Min      1Q  Median      3Q     Max
-523.20  -94.63  -21.57   34.99 1261.67

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 308.8959   157.1864   1.965 0.050425 .
bili         17.3373     3.5890   4.831 2.29e-06 ***
copper       -0.2559     0.1586  -1.614 0.107752
ast           1.0217     0.2411   4.238 3.10e-05 ***
trig          0.3658     0.2136   1.713 0.087919 .
platelet      0.4870     0.1344   3.623 0.000347 ***
protime     -25.1200    13.1987  -1.903 0.058080 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Comparing models using Anova

```
> modelA <- lm(chol ~ bili, data=pbc)
> modelB <- lm(chol ~ bili + protime, data=pbc)
> anova(modelA, modelB, test="Chisq")
```

```
Analysis of Variance Table

Model 1: chol ~ bili
Model 2: chol ~ bili + protime
  Res.Df      RSS Df Sum of Sq Pr(>Chi)
1    282 12823770
2    281 12367905  1    455865  0.00129 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Cultivating your "Bonferroni Sense"

As you build models, selecting some variables and rejecting others, you are performing significance tests either explicitly or implicitly. The more you do this, the more suspicious you should be of marginal p-values, since you have already tortured the data a fair bit to get there. You need to develop a "Bonferroni Sense" that starts tingling to tell you that you've performed a lot of tests and so only highly significant results should be included.