# Intro to modeling in R

Tad Dallas

**Reading for this week:**

> Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statistical science, 16(3), 199-231.

## What are models?

Models are representations of a system of interest, used for the purposes of inference or prediction. To say 'model' in ecology is a bit too broad to be useful, and the odd delineation between "modelers" and field researchers is a bit ridiculous. To start, we can delineate between statistical models and theoretical or phenomenological models. Statistical models (e.g., ANOVA, generalized linear models) are what we will cover here, and attempt to gain inference or prediction about some response $y$ given some number of inputs $x_1 ... x_n$. Phenomenological models are those that are constructed independent of theory but are an attempt to construct a system which can explain some ecological process, often in a dynamic way (think of the SIR model in epidemiology). Theoretical models are those that directly arise from first principles independent of empirical data. The main distinction here is between statistical models and "other" models, as both are neat, but we'll focus on simple statistical models here.

> When you create a model of a system, you have two things you don't understand; the system and the model.

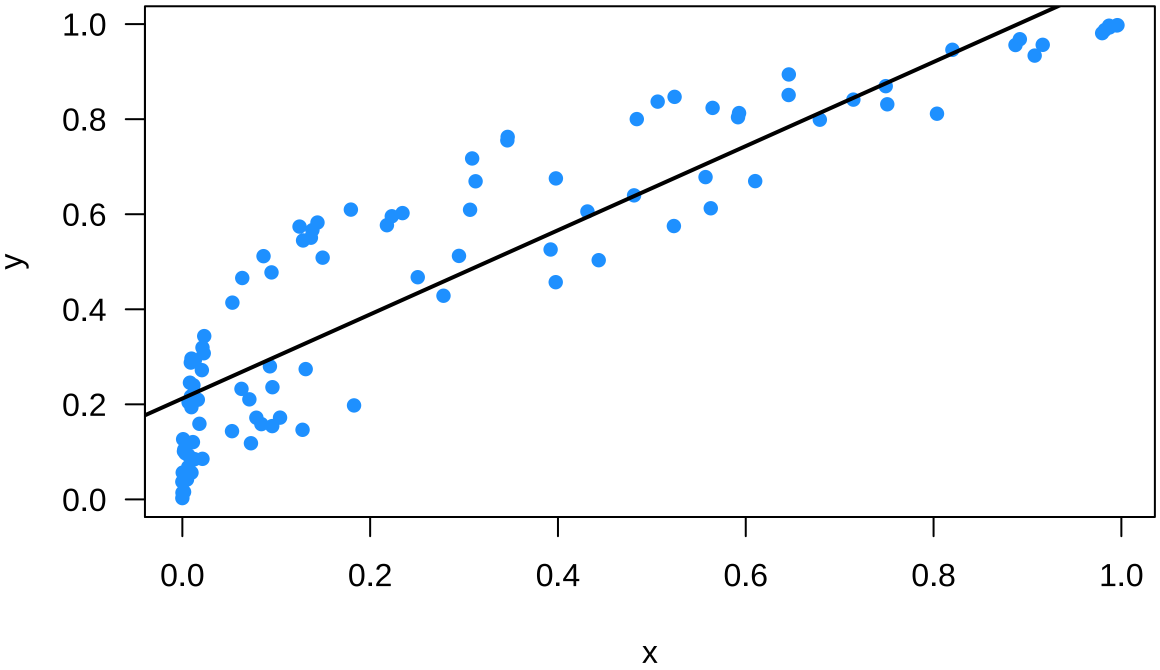
## What are the basic components of a model?

The response variable should be the thing that you want to predict, or the thing whose variability you want to understand (also sometimes referred to as the dependent variable). Often, it is something that you think is the effect produced by some other cause. For example, in examining the relationship between gas usage and outdoor temperature, it seems clear that gas usage should be the response: temperature is a major determinant of gas usage. But suppose that the modeler wanted to be able to measure outdoor temperature from the amount of gas used. Then it would make sense to take temperature as the response variable.

The explanatory variables are used to explain or predict the response variable (also sometimes referred to as independent variables).

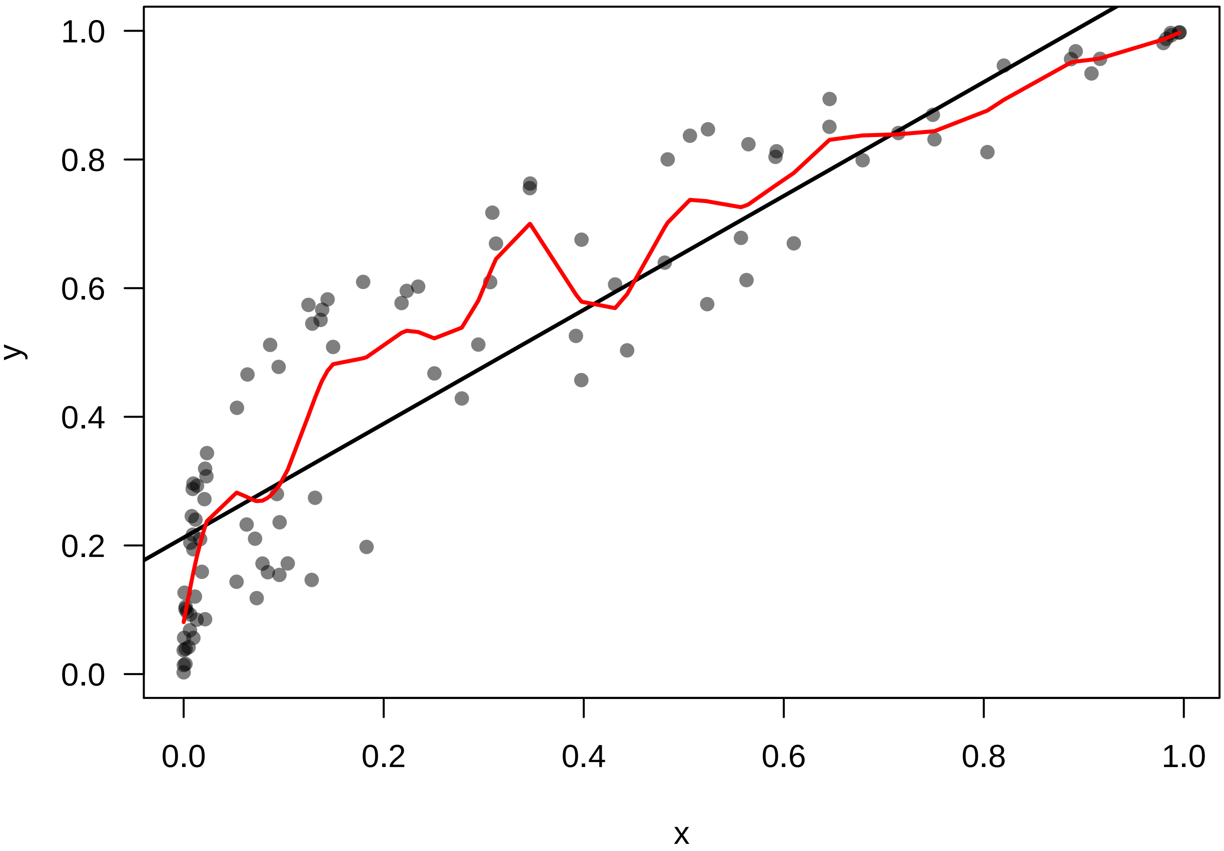
### A simple example

```r
y <- runif(100)
x <- y ** runif(100, 1,4)

m1 <- lm(y~x)


plot(y=y,x=x, pch=16,
    ylab='y', xlab='x',
    col='dodgerblue', las=1)
abline(m1, lwd=2)
```



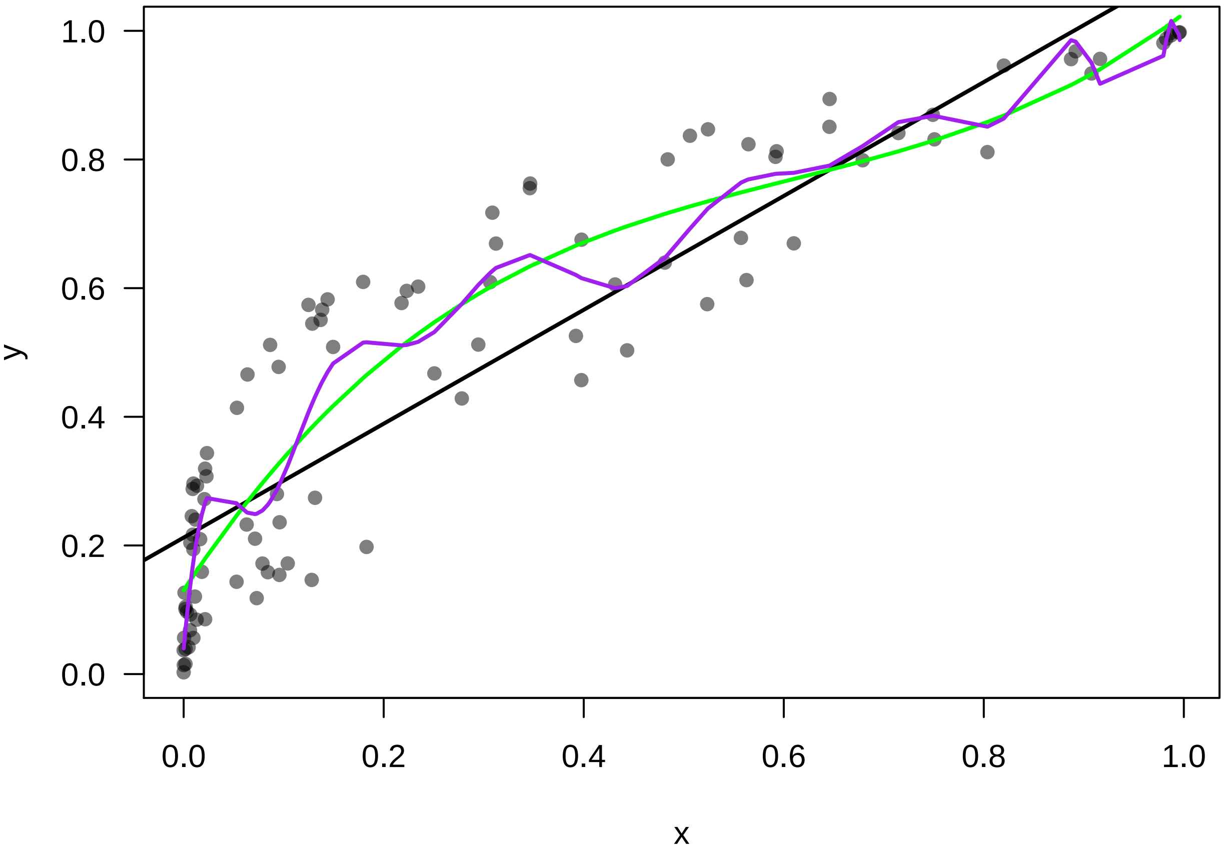
But to minimize these residuals (or the sum of squares of the residuals), we don't necessarily *need* a model, right?

```
par(mar=c(4,4,0.5,0.5))
plot(y=y,x=x, pch=16,
    ylab='y', xlab='x',
    col=adjustcolor(1,0.5), las=1)
abline(m1, lwd=2)

lines(smooth.spline(y=y,x=x), col='red', lwd=2)
```



And if we accept that we need a model, what's stopping us from making the model incredibly complex to capture all the possible variation in the data?

```
par(mar=c(4,4,0.5,0.5))
plot(y=y,x=x, pch=16,
    ylab='y', xlab='x',
    col=adjustcolor(1,0.5), las=1)
abline(m1, lwd=2)

m3 <- lm(y~poly(x,3))
lines(x=sort(x), fitted(m3)[order(x)], col='green', type='l', lwd=2)

m4 <- lm(y~poly(x,15))
lines(x=sort(x), fitted(m4)[order(x)], col='purple', type='l', lwd=2)
```



The model would quickly become uninformative. That is, the model could fit every data point perfectly, but tell us nothing about the general relationship between the two variables. This makes it impossible to use the model to predict. Adding in this level of complexity essentially minimizes the error to the fit, but in turn basically maximizes error when extrapolating to new data. This model would be overfit, and the process of adding in variables that minimize model fit error while making the model useless to other datasets, is called overfitting. Not ideal.

**Breaking the assumptions of the linear model**

Linear models have the standard assumptions of independent and identically distributed (i.i.d.) normal random variables.

More explicitly, these standard assumptions are:

- independence: samples are not autocorrelated or suffer from some other non-independence
- identically distributed: samples come from the same probability distribution
- normality: samples come from a normal probabilty distribution (error distribution is gaussian)

We'll focus on the last assumption here, since it's the easiest and most common to have to work around. For instance, if the response variable is a binary outcome (e.g., dead or alive, present or absent).

To handle these issues, a common approach is to use generalized linear models, which attempt to account for response data with non-normal distribution. This is done by using a link function to provide an appropriate *link* between the random error model (the linear model would assume normally distributed errors) and the predictors.

| Probability distribution | Typical uses | Link name |
|---|---|---|
| Normal | linear-response data | Identity |
| Poisson | count of occurrences in fixed amount of time/space | Log |

| Probability distribution | Typical uses | Link name |
|---|---|---|
| Bernoulli | outcome of single yes/no occurrence | Logit |
| Binomial | count of # of “yes” occurrences out of N yes/no occurrences | Logit |

### An example

Logistic regression

Random component: The distribution of Y is assumed to be Binomial(n, $\pi$), where $\pi$ is a probability of “success”.

Systematic component: X’s are explanatory variables (can be continuous, discrete, or both) and are linear in the parameters, e.g., $\beta_0 + \beta_1 x_i + \ldots$.

Again, transformation of the X’s themselves are allowed like in linear regression; this holds for any GLM.

**Link function**: Logit link,

$$\nu = logit(\nu) = log(\frac{\pi}{1-\pi})$$

```r
y2 <- rbinom(100, 1, 0.5)
m5 <- glm(y2 ~ x, family = binomial)
summary(m5)
```

```
##
## Call:
## glm(formula = y2 ~ x, family = binomial)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1491  -1.1234  -0.9293   1.2327   1.4702
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.06707    0.27350  -0.245    0.806
## x           -0.60214    0.64818  -0.929    0.353
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.19  on 99  degrees of freedom
## Residual deviance: 136.31  on 98  degrees of freedom
## AIC: 140.31
##
## Number of Fisher Scoring iterations: 4
```

`glm` is incredibly flexible and is the basis for more complicated models like hierarchical models, ridge regressions, generalized additive models, etc. etc. Some other forms of generalized regression are:

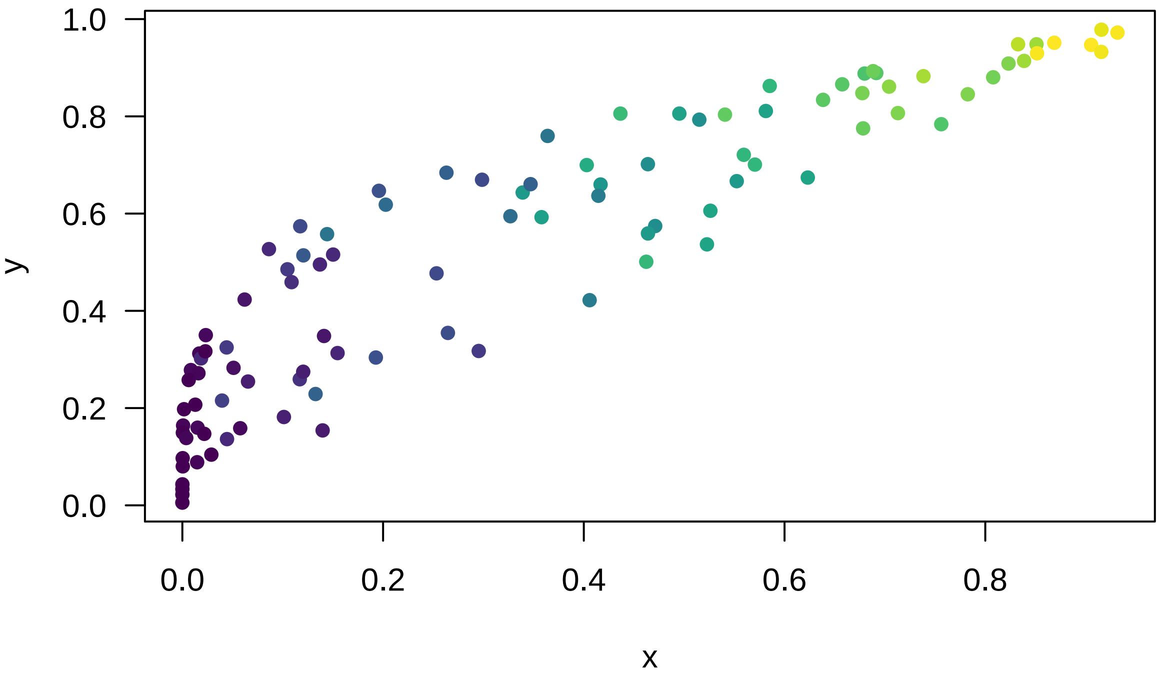
- Generalised additive models, e.g. mgcv::gam(), extend generalised linear models to incorporate arbitrary smooth functions. That means you can write a formula like y ~ s(x) which becomes an equation like y = f(x) and let gam() estimate what that function is (subject to some smoothness constraints to make the problem tractable).
- Generalized linear mixed effects models, e.g.,. lme4::glmer extend generalized linear models to incorporate both fixed and random effects.

- Trees, e.g. rpart::rpart(), attack the problem in a completely different way than linear models. They fit a piece-wise constant model, splitting the data into progressively smaller and smaller pieces. Trees aren't terribly effective by themselves, but they are very powerful when used in aggregate by models like random forests (e.g. randomForest::randomForest()) or gradient boosting machines (e.g. xgboost::xgboost.)

### Feature creation

So we've gone over modeling largely in terms of linear models of some arbitrary number of predictors, which our examples only included a single predictor. When designing a model, the researcher may want to incorporate known variable associations or interactions to estimate the support for a hypothesized effect given the data and model. This can be done by using the * operator in the formulate interface.

```
y <- runif(100)
x1 <- y ** runif(100, 1,4)
x2 <- x1 ** runif(100, 0.5,1.5)
dat <- data.frame(y,x1,x2)

m6 <- lm(y~x1+x2, data=dat)
m7 <- lm(y~x1*x2, data=dat)

plot(y=y,x=x1, pch=16,
    ylab='y', xlab='x',
    col=viridis::viridis(100)[as.numeric(cut(x2,100))], las=1)
```



```
summary(m6)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.21061 -0.07744 -0.01261  0.07818  0.23810
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)   0.2162     0.0173  12.493  < 2e-16 ***
## x1            0.3413     0.1688   2.022  0.04590 *  
## x2            0.5438     0.1667   3.263  0.00152 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1118 on 97 degrees of freedom
## Multiple R-squared:  0.854,  Adjusted R-squared:  0.851 
## F-statistic: 283.6 on 2 and 97 DF,  p-value: < 2.2e-16
```

```
summary(m7)
```

```
## 
## Call:
## lm(formula = y ~ x1 * x2, data = dat)
## 
## Residuals:
##       Min        1Q    Median        3Q       Max 
## -0.237235 -0.072802  0.007768  0.062832  0.228981 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.17529    0.01953   8.974 2.38e-14 ***
## x1           0.72062    0.18771   3.839 0.000221 ***
## x2           0.63731    0.15838   4.024 0.000114 ***
## x1:x2       -0.57731    0.15339  -3.764 0.000288 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1049 on 96 degrees of freedom
## Multiple R-squared:  0.8727, Adjusted R-squared:  0.8688 
## F-statistic: 219.5 on 3 and 96 DF,  p-value: < 2.2e-16
```

What if we don't necessary know the direct interaction, but still have some idea that a combination of variables is likely associated with some response $y$? We can do what is called feature creation (feature is just another word people use to mean variable, most often applied to creating informative predictors). For instance, if we were trying to predict the time until a bottled beer were to go skunky, we could include a measure of bottle opacity (controls the amount of light that can get in and affect the beer), and aspects of bottle dimensions (height, width). What may be important is some combination of these variables, perhaps starting to get at volumn differences or surface area to volume ratio which may influence light attenuation. So we can create a variable that represents total beer volume.

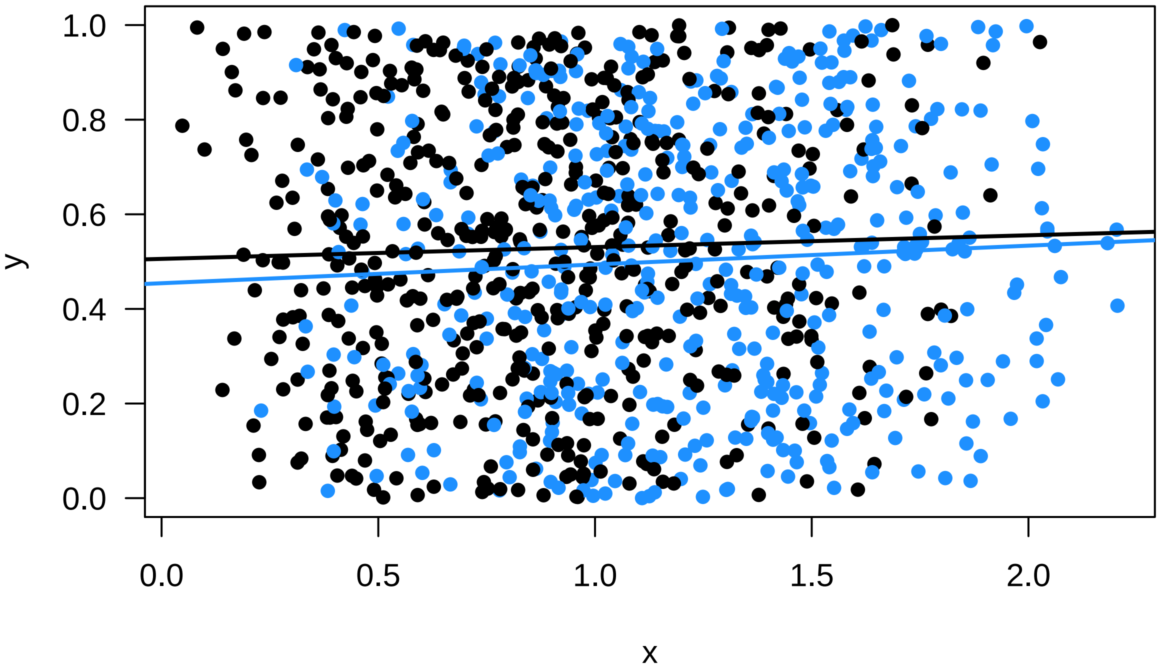
Another example that is often used is to take many many predictors which may be highly collinear and to engineer features from this set. This is often done through what is referred to as 'feature reduction', a form of feature creation that attempts to make a small set of often-independent predictor variables from a large set of collinear variables. Think of climate data, where we have temperature on each day of the year as a set of 364 predictor variables explaining ice cream sales (our response). This model would likely not fit so well with so many predictors. But we could compress the information using feature reduction approaches like principal component analysis (PCA) or just some ad hoc idea about temperature (monthly mean temperature is the most important, reducing our feature set to 12 variables).

In the previous example, we had two continuous predictors ($x1$ and $x2$), but variables can also be categorical or factor variables.

```r
y.1 <- runif(1000)
x.1 <- y * runif(1000, 0.6,1.4) + runif(1000)
x.2 <- c('a', 'b')[1+rbinom(length(x.1), 1, prob=x.1/max(x.1))]
dat2 <- data.frame(y.1,x.1,x.2)

m8 <- lm(y.1~x.1+x.2, data=dat2)

plot(y=y.1, x=x.1, pch=16,
    ylab='y', xlab='x',
    col=c(1, 'dodgerblue')[as.factor(x.2)],
    las=1)

summary(m8)
```

```
##
## Call:
## lm(formula = y.1 ~ x.1 + x.2, data = dat2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.53715 -0.24777  0.00435  0.24349  0.51272
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.49913    0.02311  21.595   <2e-16 ***
## x.1          0.03253    0.02231   1.458   0.1452
## x.2b        -0.03609    0.01960  -1.841   0.0659 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2844 on 997 degrees of freedom
## Multiple R-squared:  0.004017,   Adjusted R-squared:  0.002019
## F-statistic: 2.011 on 2 and 997 DF,  p-value: 0.1344
```

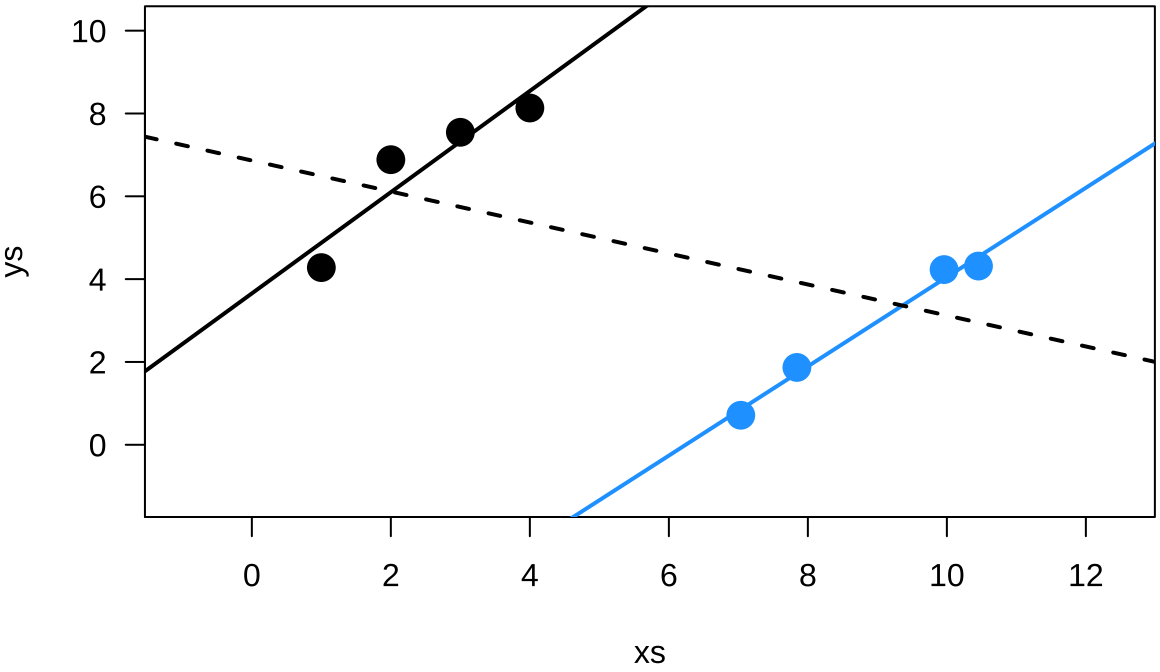
```r
m8_a <- lm(y.1~x.1, data=subset(dat2, dat2$x.2=='a'))
m8_b <- lm(y.1~x.1, data=subset(dat2, dat2$x.2=='b'))
abline(m8_a, lwd=2, col=1)
abline(m8_b, lwd=2, col='dodgerblue')
```



### Simpson's paradox

Here, we run into a cautionary tale of variable inclusion and when model builing takes a bit more knowledge of a system. In this simple example, borrowing most of the code from this user (https://commons.wikimedia.org/wiki/File:Simpson%27s_paradox_continuous.svg), we see that not including a categorical variable can result in a complete misinterpretation of the overall pattern. That is, both categorical groups have a positive relationship between $y$ and $x$, but ignoring the categorical variable results in an overall negative relationship between $y$ and $x$. This is referred to as "Simpson's paradox", and there are plenty of ecological systems where this is definitely a thing.

```
xs1 <- c(1,2,3,4)
ys1 <- xs1 + runif(4, 3, 5)

xs2 <- xs1 + runif(4, 5, 7)
ys2 <- xs2 - runif(4, 5, 7)

xs <- c(xs1,xs2)
ys <- c(ys1,ys2)

plot(xs,ys, cex=2, pch=16, las=1,
     col=rep(c(1, 'dodgerblue'), each=4),
     bg=rep(c("lightblue", "pink"), each=4),
     xlim=range(xs)+c(-2,2), ylim=range(ys)+c(-2,2))
abline(lm(ys1 ~ xs1), col=1, lwd=2)
abline(lm(ys2 ~ xs2), col='dodgerblue', lwd=2)
abline(lm(ys  ~ xs), lwd=2, lty=2)
```



### Measuring model error

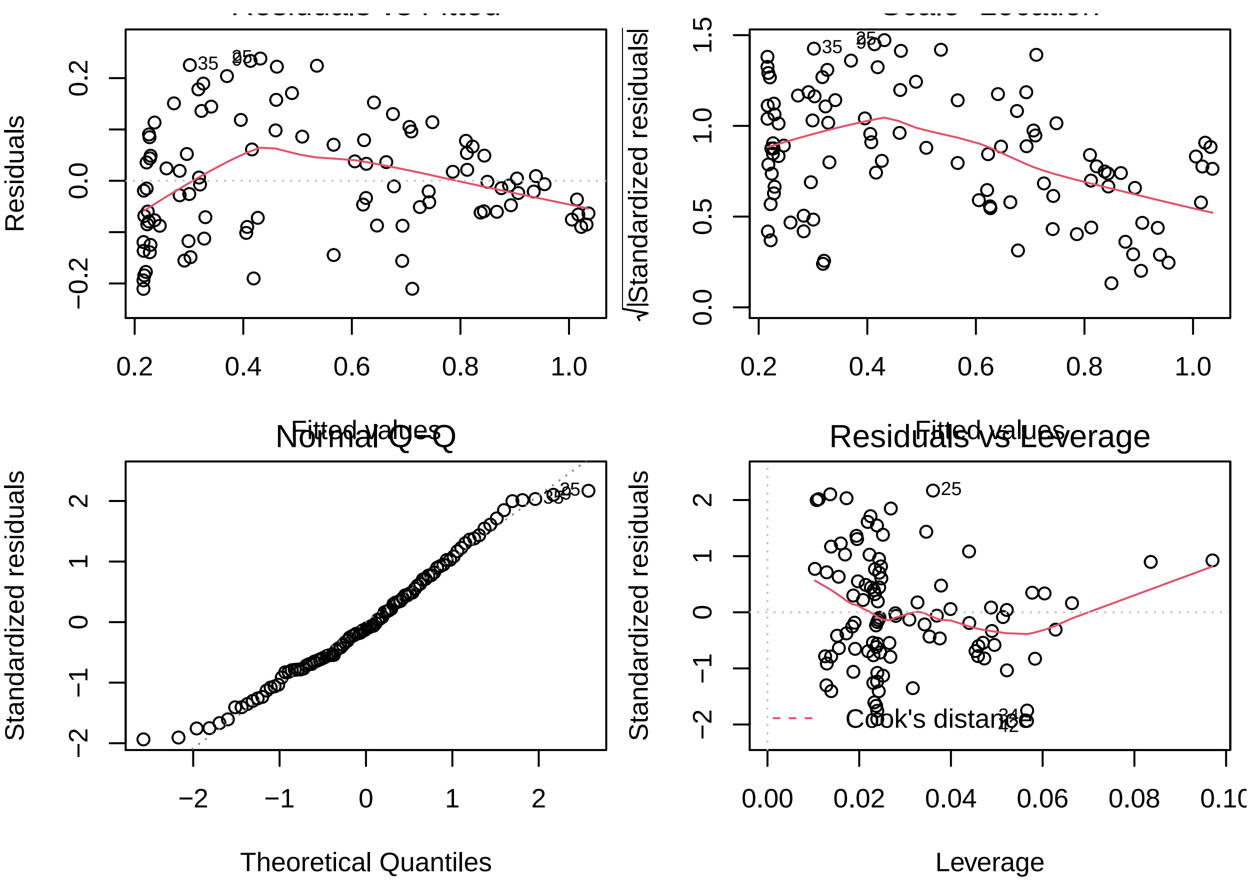
Now that we have the ability to construct models, we need to be able to assess how well they are fitting. We can use the `summary()` function to get information on both individual predictor significance, as well as overall model performance. Overall model performance is assessed in `glm` using R-squared. This measure is bounded between 0 and 1, and corresponds to the proportion of the variance of our `y` that is explained by our predicts $x_1 ... x_n$.

```
summary(m6)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.21061 -0.07744 -0.01261  0.07818  0.23810
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.2162     0.0173  12.493  < 2e-16 ***
## x1            0.3413     0.1688   2.022  0.04590 *
## x2            0.5438     0.1667   3.263  0.00152 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1118 on 97 degrees of freedom
## Multiple R-squared:  0.854,  Adjusted R-squared:  0.851
## F-statistic: 283.6 on 2 and 97 DF,  p-value: < 2.2e-16
```

But R-squared only tells us so much. What we may want to look at is how the model is truly fitting. That is, are there areas of the predictor data that do not capture the variation in `y` equally? This situation would lead to a distribution of residual variation (model residuals are the differences between model-predicted values or fitted values from the true values). To explore that, we can use the base `plot` command on the model object to explore some of these things

```
layout(matrix(1:4, ncol=2))
par(mar=c(4,4,0.5,0.5))
plot(m6)
```



- Residuals vs fitted - there should be no strong patterns and no outliers, residuals should be randomly distributed around zero.
- Normal Q-Q - residuals should go around the diagonal line, i.e., should be normally distributed. This plot helps checking if they are approximately normal.
- Scale-location - as you can see, on Y axis there are also residuals (like in Residuals vs fitted plot), but they are scaled, so it's similar to (1), but in some cases it works better.
- Residuals vs Leverage – it helps to diagnose outlying cases. As in previous plots, outlying cases are numbered, but on this plot if there are any cases that are very different from the rest of the data they are plotted below thin red lines (check wiki on Cook's distance).

## Model predictions

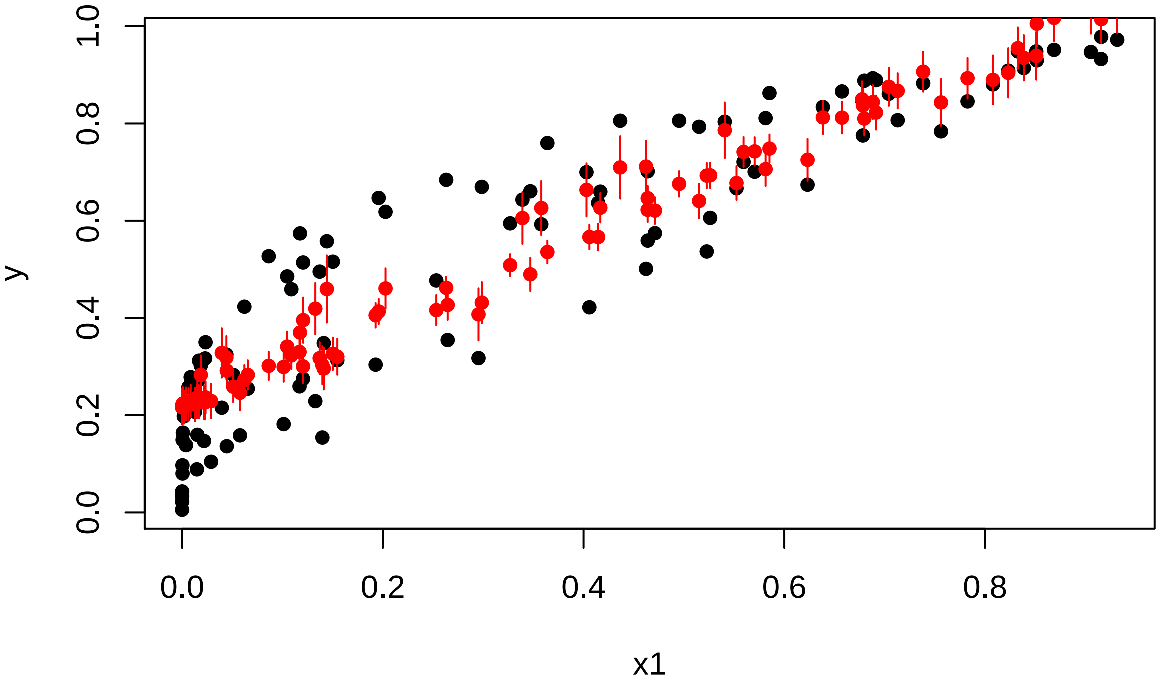
The measures of model performance discussed above are based on how well the predictors can explain the response, but, as we noted in the $n$-order polynomial case, a model could have an R-squared of 1 and be utterly useless. A non-useless model would arguably be able to predict to new data. That is, if more $x$ data were added, the response $y$ could be predicted. We can extract predictions from our model object using the predict function. We can do this for the entire data used to fit the models, or a new data object.

```
newData <- data.frame(x1=x1[1:10], x2=x2[1:10])
```

```
predict(m6, newData, interval='confidence')
```

```
##          fit       lwr       upr
## 1  0.7415221 0.7109335 0.7721107
## 2  0.6224953 0.5972774 0.6477131
## 3  0.2960942 0.2529188 0.3392697
## 4  0.3410096 0.3099920 0.3720273
## 5  0.6933648 0.6672207 0.7195089
## 6  0.8754415 0.8364018 0.9144811
## 7  0.8124819 0.7780548 0.8469089
## 8  0.6634115 0.6088603 0.7179626
## 9  0.4133302 0.3873753 0.4392851
## 10 0.2365338 0.2036559 0.2694117
```

```
preds <- as.data.frame(predict(m6, interval='confidence'))

plot(y=y, x=x1, col=1, pch=16)
points(y=preds$fit, x=x1, pch=16, col='red')
segments(y0=preds$lwr, y1=preds$upr, x0=x1, col='red')
```

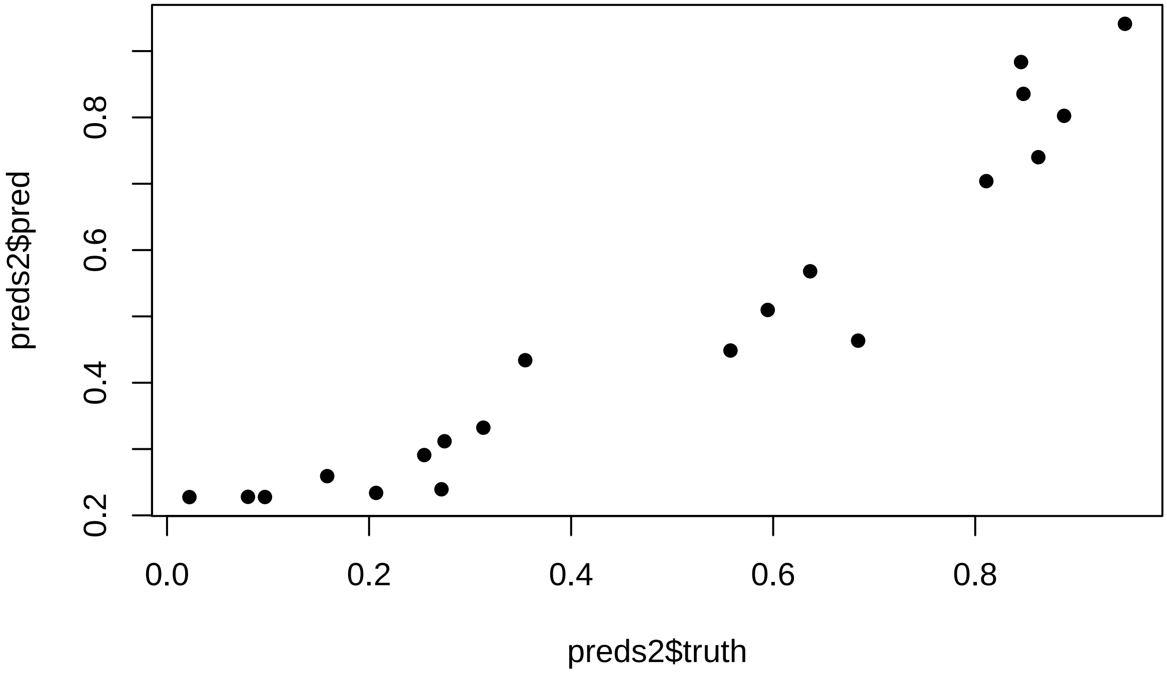


But we may wish to predict on new data or on a held out subset of data not used to train our models. This way, we can assess the performance of the model independent of the trained model conditioned on the existing data.

```
trainMod <- function(dat, holdOut=0.2){
    inds <- sample(1:nrow(dat), round(nrow(dat)*holdOut,0))
    train <- dat[-inds, ]
    test  <- dat[inds, ]
    mod <- glm(y ~ x1+x2, data=train)
    ret <- data.frame(truth=test$y, pred=predict(mod, newdata=test))
    return(ret)
}
```

```
preds2 <- trainMod(dat)
```

```
plot(preds2$truth, preds2$pred,
    pch=16)
```



**Measuring model performance**

So we have our predictions that were made on data that the model hasn't seen…so it reduces the chances that we would overfit. Now we want to assess the performance of the model. We can do so in a whole bunch of ways. A helpful way to think about model performance is to think about how we define model performance. Do we want a model whose predictions most closely match the data? Do we want a model is quite certain in it's predictions (whether the predictions are wrong or right).

**Calibration**: Calibration refers to the statistical consistency between distributional predictions and true values. So are the range of predictions in alignment with the range and distribution of the empirical data.

**Discrimination**: discrimination refers to the ability of the model to rank values correctly. This makes the most sense when thinking about binary data, where the truth is 0/1 and the prediction takes on some continuous value.

**Precision**: precision (also called sharpness) measures the uncertainty in model predictions for a given prediction

**Accuracy**: accuracy estimates the distance between predicted values and true values, typically ignoring the uncertainty in predicted values.

So one measure of accuracy for our simple example above would simply be the squared difference between prediction and truth (in order to make all values positive).

```
acc <- (preds2$truth - preds2$pred)**2
```

This results in a vector that gets at the relative error in each predicted point. We can also calculate a measure of performance for the entire model by measuring root mean squared error. Here, we just take the square root of the average model accuracy as defined above as the squared difference.

```
sqrt(mean(acc))
```

```
## [1] 0.1025586
```

This was obviously a simple example of how we could calculate one aspect of model performance, but there are plenty of others. For binary response data, there is a good discussion of model performance measures in https://doi.org/10.1002/ecm.1370. Otherwise, a good resource for thinking about model building and testing is Hastie, Tibshirani, and Friedman *Elements of Statistical Learning* (https://web.stanford.edu/~hastie/ElemStatLearn/).

### Machine learning

This idea of assessing model performance on a holdout set is not something traditional in ecological model building. This could potentially be that ecologists want to use *all* of their data to try to get at which variables are the most associated or explain the most variance in some response variable. This is fair. But the model will always be conditioned upon on the data available (I will not go into Bayesian modeling here). The book I recommend above sort of lays a lot of the foundation down for learning more about machine learning. Machine learning methods are typically designed to bypass some of the assumptions of more traditional models like those described above. Often, this results in methods that handle NAs well, are flexible to non-linear relationships between response and predictor variables, and that tend to be pretty focused on prediction over inferene (this is not inherently true though). In fact, the book on statistical learning spends the majority of time going over the inner workings of linear models for regression adn classification. It isn't until about halfway through the book where they introduce more "machine learning -esque" methods for classification and regression such as support vector machines and tree-based approaches.

```
# install.packages('gbm')
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
trainMod2 <- function(dat, holdOut=0.2){
    inds <- sample(1:nrow(dat), round(nrow(dat)*holdOut,0))
    train <- dat[-inds, ]
    test  <- dat[inds, ]
    mod <- glm(y ~ x1+x2, data=train)
    mod2 <- gbm(y ~ x1+x2, data=train, n.trees=1000,
        interaction.depth=3, cv.folds=5)
  brt.best.iter <- gbm.perf(mod2, method='OOB', plot=FALSE)
    ret <- data.frame(truth=test$y,
        predGLM=predict(mod, newdata=test),
        predGBM=predict(mod2, newdata=test, type='response', n.trees=brt.best.iter))
    return(ret)
}

preds3 <- trainMod2(dat)
```

```
## Distribution not specified, assuming gaussian ...
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance is reas
```

```
glmAcc <- sqrt(mean((preds3$truth - preds3$predGLM)**2))
gbmAcc <- sqrt(mean((preds3$truth - preds3$predGBM)**2))
```

It's important to note that the model accuracy will change as it uses a random subset of 20% holdout data each time the function is called. We can use that to our advantage when comparing models, as we can train the model $n$ times to see if the distributions of accuracy are distinct between models. It is also important to note here though that we can have lots of control over the outcome of any statistical test for model differences here by increasing the number of times we train the models to minimize the error in the distribution of accuracy values.

```
glmA <- gbmA <- c()
for(i in 1:10){
    set.seed(i)
    preds3 <- trainMod2(dat)

    glmA[i] <- sqrt(mean((preds3$truth - preds3$predGLM)**2))
    gbmA[i] <- sqrt(mean((preds3$truth - preds3$predGBM)**2))

}
```

And we can use a t-test to see if the distributions of accuracy values are different between the glm and the gbm.

```
t.test(glmA, gbmA)
```

```
##
##  Welch Two Sample t-test
##
## data:  glmA and gbmA
## t = -0.39105, df = 15.938, p-value = 0.7009
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.01806593  0.01244033
## sample estimates:
## mean of x mean of y
## 0.1087164 0.1115292
```

On average, using the boosted regression tree resulted in a change in root mean squared error of `mean(gbmA-glmA)` when training 100 models. What happens if we train 2000 models?

```
glmA2 <- gbmA2 <- c()
for(i in 1:2000){
    set.seed(i)
    preds3 <- trainMod2(dat)

    glmA2[i] <- sqrt(mean((preds3$truth - preds3$predGLM)**2))
    gbmA2[i] <- sqrt(mean((preds3$truth - preds3$predGBM)**2))

}
```

This may not really influence the mean difference between the models in terms of root mean squared errors (`mean(gbmA2-glmA2)`), but it will influence the resulting statistical test that we inappropriately apply to prove a point.

```
t.test(glmA2, gbmA2)
```

```
##
##  Welch Two Sample t-test
##
## data:  glmA2 and gbmA2
## t = -9.661, df = 3979.2, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.005251416 -0.003479581
## sample estimates:
```

```
## mean of x mean of y
## 0.1121530 0.1165185
```