Introduction to Computational BioStatistics with R: linear models

Erik Spence

23 October 2025

Today's slides

Today's slides can be found here. Go to the "Introduction to Computational BioStatistics with R" page, under Lectures, "Linear models".

`https://scinet.courses/1391`

Today's class

Today we will continue our adventures in data analysis.

- Initial data exploration.
- Linear models.
- Formulae.
- Quadratic models.
- Fits with categorical independent variables.

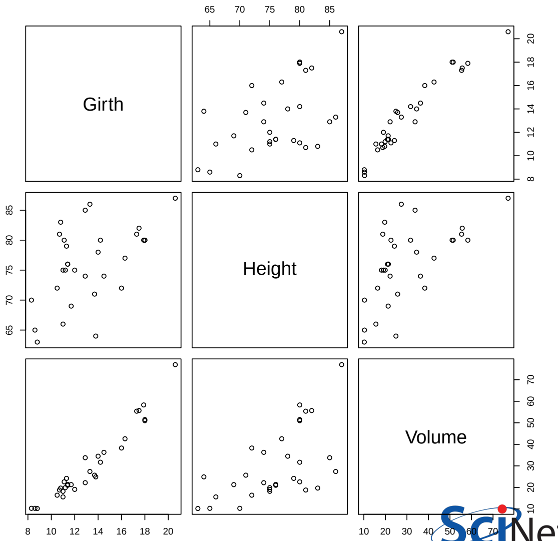
As always, ask questions.

Look at your data

The very first step to dealing with your data is to plot it. Always always always!

The 'pairs' function will do the same as demonstrated here.

```
>
> str(trees)
'data.frame':  31 obs.  of 3 variables:
$Girth :  num 8.3 8.6 8.8 10.5 10.7 ...
$Height:  num 70 65 63 72 81 ...
$Volume:  num 10.3 10.3 10.2 16.4 18.8 ...
>
> plot(trees)
>
```

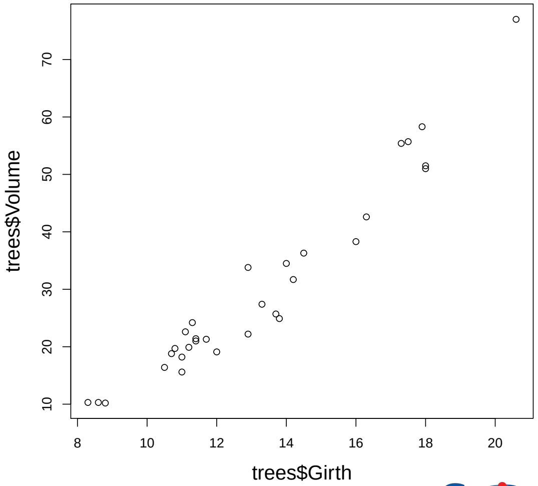


Look at your data, continued

Once you've had a first look, you might want to take a closer look at a particular pair of variables.

```
>
> plot(trees$Girth, trees$Volume)
>
```

Looks like they might be related.



Correlation and covariance

If we suspect that two variables might be related to one another, it's worth our time to look at the correlation and covariance of the variables.

Covariance:

$$\sigma_{XY} = E\left[\left(X - E\left(X\right)\right)\left(Y - E\left(Y\right)\right)\right]$$

Correlation (Pearson's correlation):

$$\rho_{XY} = \frac{E\left[\left(X - E\left(X\right)\right)\left(Y - E\left(Y\right)\right)\right]}{\sigma_X \sigma_Y}$$

Recall that the standard deviation:

$$\sigma_X = \sqrt{E\left[\left(x - E\left(x\right)\right)^2\right]}$$

Where E is the expectation value of the quantity in question.

Correlation and covariance, continued

The "cor" function will produce the correlation from the previous slide.

The "var" function returns the variance or the covariance, depending on the number of arguments.

Recall that the standard deviation is the square root of the variance.

```
>
> cor(trees$Girth, trees$Volume)
[1] 0.9671194
>
> var(trees$Girth, trees$Volume)
[1] 49.88812
>
> var(trees$Girth)
[1] 9.847914
>
> sd(trees$Girth)
[1] 3.138139
>
```

Model fitting

One important application of statistics is the fitting of models to empirical data. There are many ways to do this, but they are all based on the same principles:

- collect some data.
- propose a relationship between the 'features', and the 'target' in your data (if there is a 'target').
  - 'features' are the independent variables in your data ($\mathbf{x}$),
  - the 'target' is the dependent variable ($\boldsymbol{y}$). Not all data sets have dependent variables.
- Fit your model to the data,
- Test and evaluate the quality of the model.

Depending on the field, this is called modelling, fitting, regression.

Linear models

Let's consider the simplest possible case, that the relationship between the independent and dependent variables is linear.

$$y \simeq \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \delta$$

As usual:

- y is the dependent variable,
- $x_1, ..., x_n$ are the independent variables,
- β_0 is the intercept,
- $\beta_1, ..., \beta_n$ are the coefficients, and
- δ is noise.

For example, we might assume a relationship for plant growth:

$$\textbf{growth} \simeq \textbf{water} + \textbf{temp} + \textbf{fertilizer}...$$

The plant growth is linearly related to the temperature, amount of fertilizer and water, etc.

Fitting a linear model

We use the "lm" function to fit a linear model to our data.

The weird thing with the ~ ("tilde") is called a "formula".

Formulae are used, in R, to describe the functional relationship between variables when building models.

```
>
> model <- lm(trees$Volume ~ trees$Girth)
>
> model

Call:
lm(formula = trees$Volume ~ trees$Girth)

Coefficients:
 (Intercept)   trees$Girth
     -36.943         5.066
>
```

Volume $\simeq$ -36.943 + (5.066 x Girth).

Fitting a linear model, continued

The lm function returns an object of class 'lm'.

This is essentially just a very-deep named list.

If we so desire, we can now use the model to calculate the model's prediction for a new tree, with a girth of, say, 15.12.

- Use the 'predict' function.
- Use the coefficients directly.

```
>
> model <- lm(Volume ~ Girth, data = trees)
>
> predict(model, newdata = data.frame(Girth = 15.12))
       1
39.65229
>
> coef(model)
 (Intercept)       Girth
  -36.943459    5.065856
>
> coef(model)[1] + coef(model)[2] * 15.12
(Intercept)
39.65229
>
```

Using formulae, an aside

Formulae show up all over the place in R. There are two ways of building a formula:

- Use vectors of data as the arguments to the formula.
- Specify the names of the columns of a data frame, and then pass the data frame as an argument to the function.
- The entry to the left of the tilde is the dependent variable.
- All the entries to the right of the tilde are the independent variables (there can be more than one).

```
>
> model <- lm(trees$Volume ~ trees$Girth)
>
> model <- lm(Volume ~ Girth, data = trees)
>
> model2 <- lm(Volume ~ Girth + Height,
+      data = trees)
>
```

The second option, specifying column names, is unfortunate due to its syntax being inconsistent with the rest of R, but it is the one more commonly used. It's also somewhat more flexible for making predictions using the trained model.

Using formulae, an aside, continued

There are a few other ways to specify a formula.

- A formula can be assigned to a variable, and used later.
- To specify "all columns which have not yet been mentioned" use the ".".
- To remove an already-specified feature, use the minus sign.
- You can also mix data frame columns and non-column vectors.

```
> f <- Volume ~ .
> class(f)
[1] "formula"
>
> trees.model <- lm(f, data = trees)
>
> library(boot)
> model <- lm(ulcer ~ .  - sex - year, data = melanoma)
>
> model
Call:
lm(formula = ulcer ~ .  - sex - year, data = melanoma)
Coefficients:
 (Intercept)         time       status          age  thickness
   6.146e-01   -5.595e-05   -1.417e-01    4.210e-04  6.045e-02
>
```

Using formulae, an aside, constant values

By default, R will automatically give you an intercept in your model.

On rare occasion, you may want to just have the intercept, and nothing else. To do this use a '1' character.

Note that you must put in a '1'. No other constant works.

```
>
> model <- lm(Volume ~ 1, data = trees)
>
> model
Call:
lm(formula = Volume ~ 1, data = trees)
Coefficients:
(Intercept)
30.17
>
> model <- lm(Volume ~ 2, data = trees)
Error in terms.formula(formula, data = data) :
invalid model formula in ExtractVars
>
```

Using formulae, an aside, interaction terms

Some people like to specify interaction terms in their models. These are terms where features are multiplied together as a combined independent variable.

If you specify "A * B" in a formula you will get "A + B + A:B", where "A:B" is the interaction term.

If you want just the interaction term, put just "A:B" in the formula.

```
> model <- lm(Volume ~ Girth * Height, data = trees)
>
> model
Call:
lm(formula = Volume ~ Girth * Height, data = trees)
Coefficients:
 (Intercept)      Girth     Height   Girth:Height
     69.3963    -5.8558    -1.2971         0.1347
>
> model <- lm(Volume ~ Girth:Height, data = trees)
> model
Call:
lm(formula = Volume ~ Girth:Height, data = trees)
Coefficients:
 (Intercept)   Girth:Height
   -25.24170        0.05449
>
```

Calculating confidence intervals, an aside

Suppose that you've calculated the mean, $\bar{x}$, of some quantity. What is the uncertainty on that quantity?

To answer this question, we estimate the Standard Error

$$\mathbf{SE}\,(x)^2 = \frac{s^2}{n}$$

where $s^2 = \sum (x_i - \bar{x})^2 \,/\, (n-1)$. The 95% confidence interval, in which there is a 95% chance the true mean of the population lies, is given by

$$\mu \pm 1.96\ \mathbf{SE(x)}$$

This is because 1.96 standard deviations is approximately what contains 95% of the normal distribution.

```
>
> x <- trees$Girth
>
> my.mean <- mean(x)
>
> se2 <- var(x) / length(x)
>
> my.mean - 1.96 * sqrt(se2)
[1] 12.14368
>
> my.mean + 1.96 * sqrt(se2)
[1] 14.35309
>
```

Calculating confidence intervals, an aside, continued

We can also use the Standard Errors to perform hypothesis tests on the coefficients of our linear model.

- The Null Hypothesis is that the coefficients in our linear model have a value of zero, meaning that the dependent variable does not depend on the independent variable.
- To test this we need to determine whether our estimate of the coefficient is sufficiently far from zero to reject the Null Hypothesis. How far is far enough?
- We can check in the usual way, by calculating a t-statistic

$$\boldsymbol{t_j = \frac{\beta_j - 0}{\mathrm{SE}(\beta_j)}}$$

- This estimates the number of standard deviations that $\boldsymbol{\beta_j}$ is away from zero.
- The question then becomes: what is the probability of getting a t statistic of value $|\boldsymbol{t}|$, or larger? This is your p value.

Calculating confidence intervals, an aside, more

Given our t statistic (for a Null Hypothesis of zero):

$$t_j = \frac{\beta_j - 0}{\mathrm{SE}(\beta_j)}$$

we need only determine the probability of getting $|t|$ or greater. To determine this we use a 't' distribution, which is very close to the Gaussian CDF for $n > 30$. The second argument is the number of degrees of freedom. The factor of 2 is because it could be left or right tailed.

The p value is small. The probability of committing a Type I error is quite low.

```
>
> coefs <- coef(summary(trees.model))
>
> est <- coefs["Height", "Estimate"]
> std.err <- coefs["Height", "Std. Error"]
>
> my.t <- (est - 0) / std.err
>
> my.t
[1] 2.606594
>
> 2 * (1 - pt(my.t, length(x) - 3))
[1] 0.01449097
>
```

Fitting a linear model, continued more

Important details about the model can be found in the summary:

- The "t value" is the estimate divided by the standard error.
- The p-value is the probability of achieving a value of t, or larger, under the Null Hypothesis (estimate = 0).

```
> model <- lm(Volume ~ Girth + Height, data = trees)
```

```
> summary(model)
Call:
lm(formula = Volume ~ Girth + Height, data = trees)


Residuals:
     Min       1Q    Median      3Q      Max
 -6.4065  -2.6493  -0.2876  2.2003   8.4847
Coefficients:
               Estimate  Std. Error  t value  Pr(>|t|)
 (Intercept)  -57.9877       8.6382   -6.713  2.75e-07  ***
 Girth          4.7082       0.2643   17.816   < 2e-16  ***
 Height         0.3393       0.1302    2.607    0.0145  *
---
Signif.  codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’  0.1 ‘ ’ 1

Residual standard error:  3.882 on 28 degrees of freedom
Multiple R-squared:  0.948, Adjusted R-squared:  0.9442
F-statistic:  255 on 2 and 28 DF, p-value:  < 2.2e-16
```

Fitting a linear model, continued even more

What about that F-statistic at the bottom?

- The fit also gives an analysis of the null hypothesis that $\beta_1 = \beta_2 = ... = \beta_n = 0$.
- This means that there's no dependence on the dependence on the independent variables at all.
- This null hypothesis can be rejected in this case.

```
> model <- lm(Volume ~ Girth + Height, data = trees)
> summary(model)
Call:
lm(formula = Volume   Girth + Height, data = trees)


Residuals:
     Min        1Q    Median       3Q      Max
 -6.4065   -2.6493   -0.2876   2.2003   8.4847
Coefficients:
               Estimate   Std. Error   t value   Pr(>|t|)
 (Intercept)   -57.9877       8.6382    -6.713   2.75e-07   ***
 Girth           4.7082       0.2643    17.816    < 2e-16   ***
 Height          0.3393       0.1302     2.607     0.0145   *
---
Signif.  codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'  0.1 ' ' 1

Residual standard error:  3.882 on 28 degrees of freedom
Multiple R-squared:  0.948, Adjusted R-squared:  0.9442
F-statistic:  255 on 2 and 28 DF, p-value:  < 2.2e-16
```

Fitting a linear model, continued some more

There are some assumptions built into "lm". You need to know the fine print:

- The noise in the data, δ, is normally distributed about the true value (bivariate normality).
- Homoscedasticity: the variance in the noise is constant throughout the data.
- The data is not correlated to the noise.
- The independent variables are independent of each other.

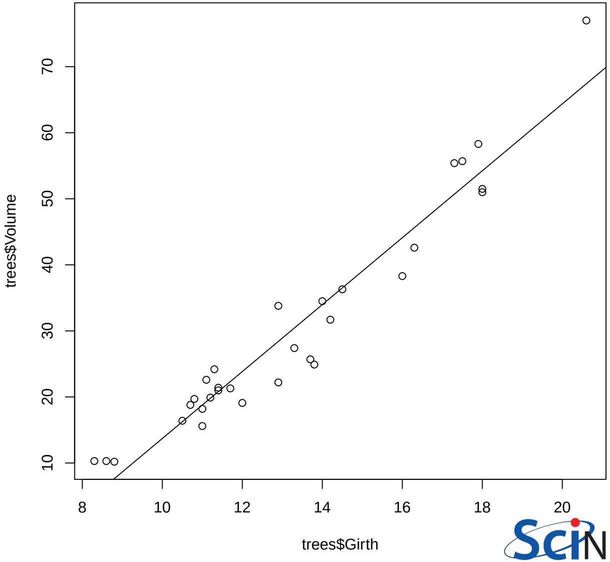
If these conditions are not met your model is not necessarily on a good statistical foundation.

Fitting a linear model, continued even more

It's always good to visualize your model once it's been made.

```
>
> model <- lm(Volume ~ Girth,
+       data = trees)
>
> plot(trees$Girth, trees$Volume)
> abline(model)
>
```

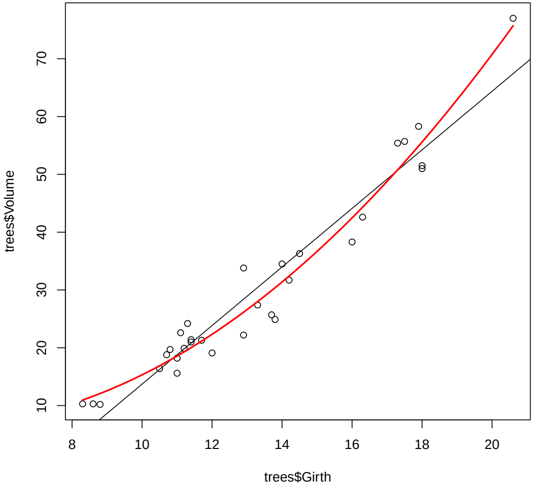
Not bad, but could be better.



Fitting a quadratic model

We can increase the order of the polynomial which we are fitting to the data.

```
> Girth2 <- trees$Girth**2
>
> model2 <- lm(Volume ~ Girth + Girth2,
+       data = trees)
>
> plot(trees$Girth, trees$Volume)
> abline(model)
>
> xx <- seq(min(trees$Girth), max(trees$Girth),
+             len = 100)
> yy <- predict(model2, data.frame(Girth = xx,
+                  Girth2 = xx**2))
>
> lines(xx, yy, lwd = 2, col = "red")
```

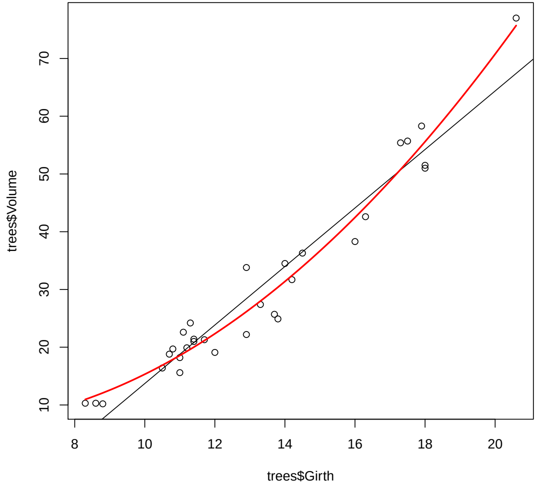


The "abline" command only works with linear fits.

Fitting a quadratic model using curve

The curve function is also a useful option.

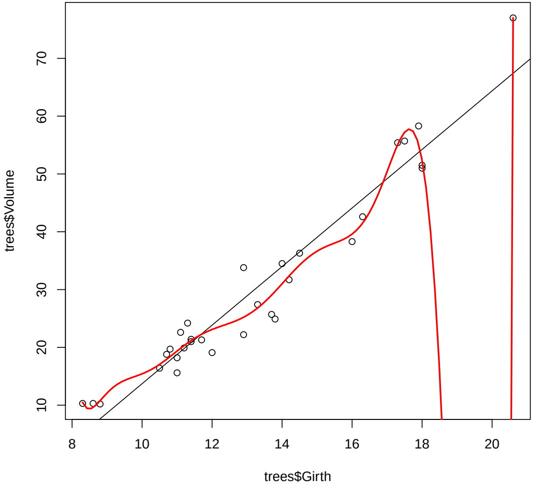
```
> Girth2 <- trees$Girth**2
>
> model2 <- lm(Volume ~ Girth + Girth2,
+       data = trees)
>
> plot(trees$Girth, trees$Volume)
> abline(model)
>
> curve(predict(model2, data.frame(Girth = x,
+                                 Girth2 = x**2)),
+        from = min(trees$Girth),
+        to = max(trees$Girth), add = TRUE,
+        lwd = 2, col = 'red')
>
```



Fitting a higher-order model

There are several ways to have a higher-order fit. The best one is to use 'poly'.

```
>
> model10 <- lm(Volume ~ poly(Girth, 10),
+                 data = trees)
>
> plot(trees$Girth, trees$Volume)
> abline(model)
>
> p.model10 <- predict(model10,
+                         data.frame(Girth = xx))
>
> lines(xx, p.model10, lwd = 2, col = "red")
>
```



What's up with 'poly'?

We've seen that one way to do a non-linear fit is to use the commands

```
> xsq <- x * x
> model3 <- lm(y ~ x + xsq)
```

or it can be written as

```
> model3 <- lm(y ~ x + I(x**2) + I(x**3))
```

Unfortunately, when used this way, x, x^2 and x^3 will be correlated with each other. This can lead to resolution problems, especially at higher orders.

The 'poly' function fixes this by generating at set of orthogonal polynomials evaluated at 'x'.

Linear models with binary categorical variables

What happens when you have independent variables that are not continuous? Can we make a linear model in that case?

The simplest example of this would be the case where x_1 is a binary variable, taking only two values, say 0 or 1.

$$y \simeq \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \delta$$

In this simple case, R will treat β_1 as an offset, like an addition to β_0, which is only used when $x_1 = 1$ and is ignored with $x_1 = 0$.

Models with binary categorical variables, example

The leuk data set contains survival time data for leukemia patients.

- wbc is white blood cell count,
- ag is the binary result of an unspecified test.

The coefficient for the 'ag' variable is given the name 'agpresent', which is the second of the two possible values for that binary variable.

```
> library(MASS)
> str(leuk)
'data.frame':  33 obs.  of 3 variables:
$wbc :  int 2300 750 4300 2600 6000 10500 10000 17000 ...
$ag :  Factor w/ 2 levels "absent","present":  2 2 2 2 ...
$time:  int 65 156 100 134 16 108 121 4 39 143 ...
>
> model <- lm(time ~ wbc + ag, data = leuk)
> summary(model)
.
.
.
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 30.9388802 11.4082897   2.712  0.01096 *
wbc         -0.0004443  0.0002006  -2.215  0.03448 *
agpresent   44.4491283 13.6299888   3.261  0.00277 **
.
.
.
```

Linear models with multiclass categorical variables

What happens when your categorical variable is not binary? How is the variable treated?

In this case, our variable $\boldsymbol{x_1}$ is one of, say, 4 categories.

$$\boldsymbol{y \simeq \beta_0 + \beta_{11} x_{11} + \beta_{12} x_{12} + \beta_{13} x_{13} + \beta_{14} x_{14} + \beta_2 x_2 + \delta}$$

Similar to the binary case, $\boldsymbol{\beta_{11}}$ is absorbed into the intercept, $\boldsymbol{\beta_0}$. Every other value of $\boldsymbol{x_1}$ ($\boldsymbol{x_{12}}$, $\boldsymbol{x_{13}}$, $\boldsymbol{x_{14}}$) takes a value of either 0 or 1, indicating which category the particular data point belongs to. The corresponding coefficients ($\boldsymbol{\beta_{12}}$, $\boldsymbol{\beta_{13}}$, $\boldsymbol{\beta_{14}}$) are active in the model for the data points in their category.

Once again, make sure that feature $\boldsymbol{x_1}$ is a factor if you do this in R.

Models with multiclass categorical variables, example

What happens if there is more than one category for a given categorical variable?

We can use the 'levels' command to find the values in a factor.

When the fit is done,

- the first value of the factor is absorbed into the intercept,
- all other values are given the value of either 0 or 1, depending on whether a given data point is in that level or not.

```
> levels(ChickWeight$Diet)
[1] "1" "2" "3" "4"
>
> model <- lm(weight ~ Time + Diet, data = ChickWeight)
> summary(model)
⋮
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.9244     3.3607   3.251  0.00122 **
Time          8.7505     0.2218  39.451  < 2e-16 ***
Diet2        16.1661     4.0858   3.957 8.56e-05 ***
Diet3        36.4994     4.0858   8.933  < 2e-16 ***
Diet4        30.2335     4.1075   7.361 6.39e-13 ***
⋮
```

Summary

We've started looking at data, and fitting it. Things to remember:

- Plot your data!
- The standard error can be used to estimate confidence intervals for calculated quantities.
- Start with lm, both for linear and other polynomial fits.
- The summary function can give both confidence intervals on coefficient values, as well as information on the null hypothesis that the coefficient is zero.
- Independent variables that are categorical can also be used in linear models. These take a value of 0 or 1, depending on whether the data point is in the given category.

We will examine model fitting further next class.