Introduction to programming in R: linear models

Erik Spence

SciNet HPC Consortium

22 March 2023

Today's slides

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

`https://scinet.courses/1277`

Today's class

Today we will begin our adventures in actual data analysis.

- Initial data exploration.
- Linear models.
- Generalized linear models.

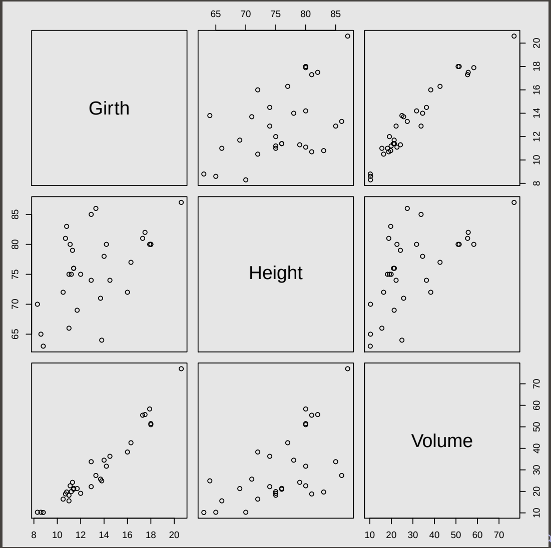
As always, ask questions.

Look at your data

There 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)
>
```

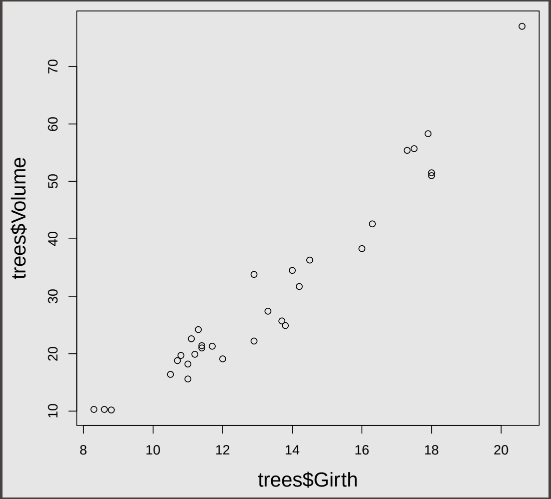


Look at your data, continued

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 (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, is this called: modelling, fitting, regression.

Linear models

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

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

As usual:

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

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

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

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).

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.

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" us 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
>
```

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 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.
- Homoscedasticity: the variance in the noise is constant throughout the data.

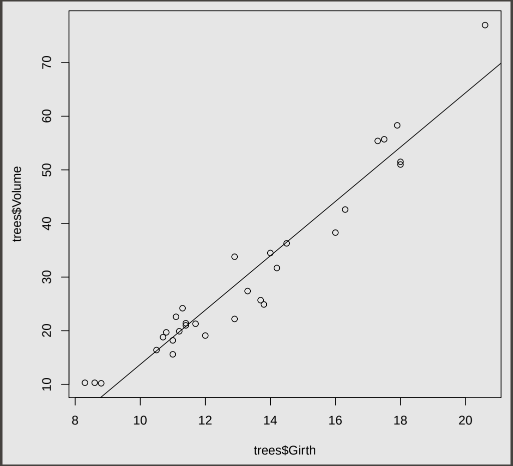
If these conditions are not met your model is not on a good statistical foundation.

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)
>
```

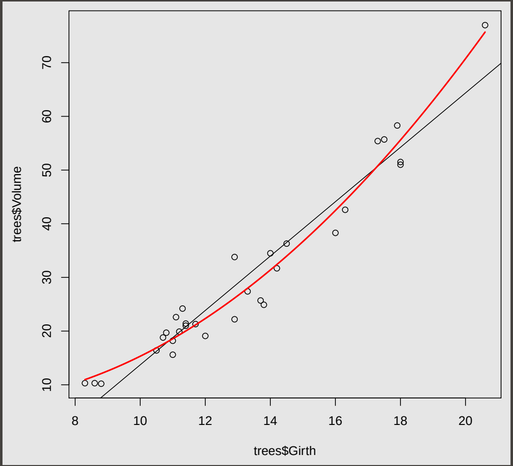
Not bad, but could be better.



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$Girth),
+            len = 100)
> yy <- model2$coef %*% rbind(1, xx, xx * xx)
>
> lines(xx, yy, lwd = 2, col = "red")
```



”abline” only works with linear fits.

matrix-vector multiplication, an aside

What's up with that %*% symbol? That's the matrix-multiplication operator.

- Matrix-scalar multiplication gives element-by-element multiplication.
- Matrix-vector multiplication requires a special operator to be done the usual way.

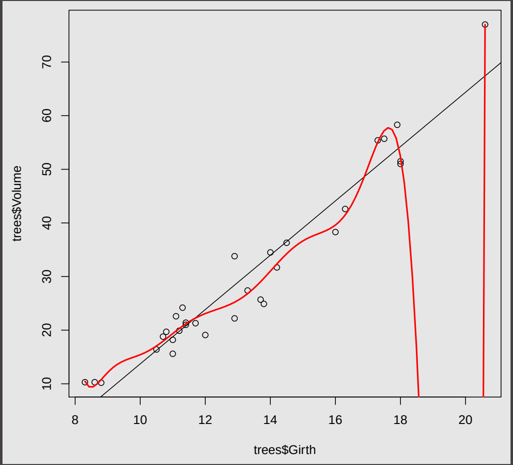
```
> A <- matrix(1:9, nrow = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
>
> 1:3 %*% A
     [,1] [,2] [,3]
[1,]   14   32   50
>
```

$$\begin{bmatrix}\beta_0 & \beta_1 & \beta_2\end{bmatrix} * \begin{bmatrix}1 & 1 & 1\\ xx_1 & xx_2 & xx_3\\ xx_1^2 & xx_2^2 & xx_3^2\end{bmatrix} = \begin{bmatrix}\beta_0 * 1 + \beta_1 * xx_1 + \beta_2 * xx_1^2\\ \beta_0 * 1 + \beta_1 * xx_2 + \beta_2 * xx_2^2\\ \beta_0 * 1 + \beta_1 * xx_3 + \beta_2 * xx_3^2\end{bmatrix}$$

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'.

That's great, but we're not done yet

It's always a good idea to do some further analysis of your model before declaring success. There are a few things in particular that should always be done.

- plot the residuals of the model, in various ways,
- examine the statistics of the residuals,
- examine the statistics of the model.

What are residuals? Residuals are the distance between the actual value, and the value predicted by the model, for each data point:

$$R_i = f(x_i) - y_i$$

where f is the model, evaluated at data point x_i, and y_i is the actual value of the dependent variable.

Step 1: plot the residuals

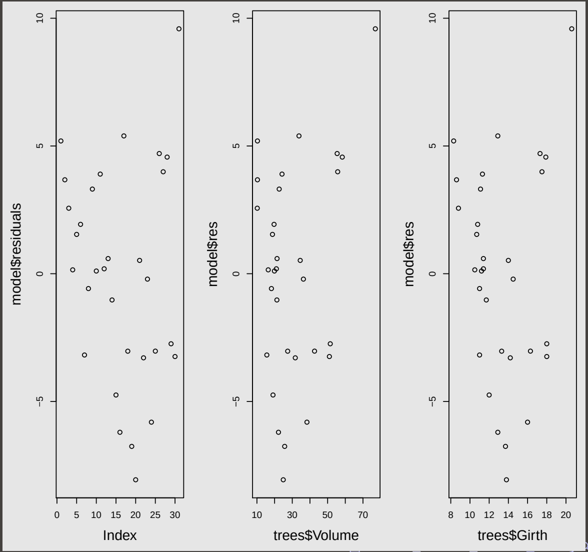
Always plot your residuals. Always.

```
> par(mfrow = c(1, 3))
>
> plot(model$residuals)
> plot(trees$Volume, model$residuals)
> plot(trees$Girth, model$residuals)
>
```

Plot your residuals against everything:

- index,
- against the dependent variables,
- against the independent variables.

You should see a snowstorm. There should be no clumps.

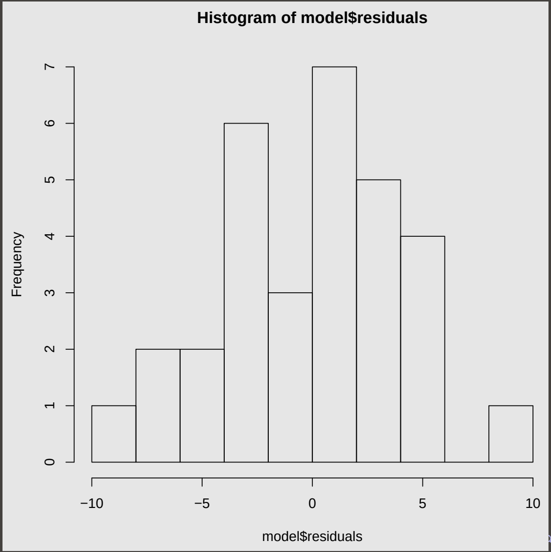


Step 2: plot the residuals via histogram

Always plot a histogram of your residuals. Things to look for:

- The mean should be zero. If your residuals are not centered on zero your model is missing something.
- The distribution should be symmetric. If it's not, it's biased (there 'structure' in the data which has not been captured by the model).
- Distribution should be a Gaussian (an assumption made as part of the fit).

```
> par(mfrow = c(1, 1))
> hist(model$residuals, breaks = 11)
```

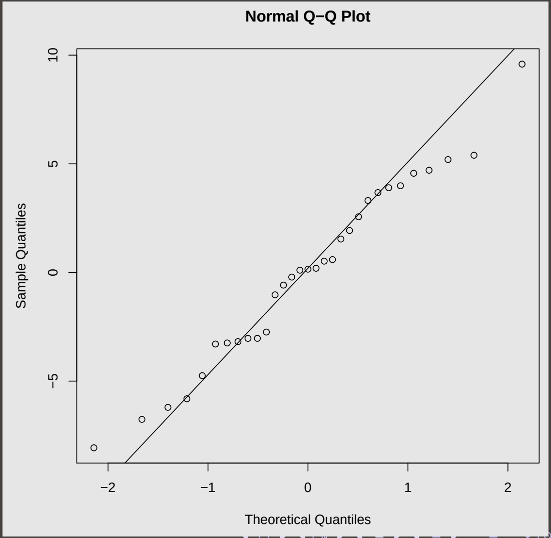


Step 3: plot the residuals via Q-Q plot

Plot your residuals on a Q-Q plot.

- A Q-Q plot graphically demonstrates how normally-distributed the residuals are.
- Ideally the residuals should be normally distributed.

```
>
> qqnorm(model$residuals)
> qqline(model$residuals)
>
```



Using R^2

$R^2 =$ (explained variation) / (total variation).

- Explains how much of the variance in the data can be explained by the model.
- All other variation is caused by shortcomings in the model, or noise.
- A high R^2 value is necessary, but not sufficient, for the model to be satisfactory.

```
> 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
```

Other regression models

The linear model built by "lm" has some built-in assumptions:

- Normally distributed noise.
- Constant variance.

There are situations where these assumptions are dramatically violated. To deal with this, let us examine "Generalized Linear Models". These allow

- Non-normally distributed noise.
- Non-constant variance.

If you find that you have structure in your residuals, it's possible that you need to use a generalized linear model.

Generalized linear models

When should you use a generalized linear model?

- You know that your data should come from a non-linear, non-polynomial distribution (exponential, Poisson, etc).
- You don't know what your distribution should be, and you've got structure in your residuals.

How do generalized linear models work? Let's start with a regular linear model. Assuming the vectors of data are (X, Y), the problem is to find the vector of coefficients β such that

- $E(Y) = X\beta$
- assuming that $Y \sim N(X\beta, \sigma^2)$,

where E is the expectation value, $N(\mu, \sigma^2)$ is the symbol for a normal distribution centred on μ with a standard deviation of σ.

Generalized linear models, continued

As an example, for a log-linked Gaussian GLM, we have

- $\log\left(E(Y)\right) = X\beta$,
- which means that $E(Y) = e^{X\beta}$,
- $Y \sim N(e^{X\beta}, \sigma^2)$.

where E is the expectation value, $N(\mu, \sigma^2)$ is the symbol for a normal distribution centred on μ with a standard deviation of σ.

Generalized linear models consist of 3 parts:

- A "link" function. A function which transforms the data such that it becomes linear.
- A linear predictor $(X\beta)$.
- A probability distribution, which describes the type of noise to be expected in the dependent variable.

Generalized linear models, continued more

There are many possible link functions available. The most common ones are

- Identity: $E(Y) = X\beta$,
- Log: $\log\left(E(Y)\right) = X\beta \quad \rightarrow \quad E(Y) = e^{X\beta}$.
- Logit: $\log\left(\frac{E(Y)}{1-E(Y)}\right) = X\beta \quad \rightarrow \quad E(Y) = \frac{1}{1+e^{-X\beta}}$
- Inverse: $1/E(Y) = X\beta \quad \rightarrow \quad E(Y) = 1/\left(X\beta\right)$

The identity link function results in a standard linear regression. By performing a generalized linear model using this link function, with Gaussian noise, you will get the same result as using the "lm" function.

Generalized linear models, continued even more

Once a link function has been chosen, the type of error in the data must be chosen. The different error families have different default link functions.

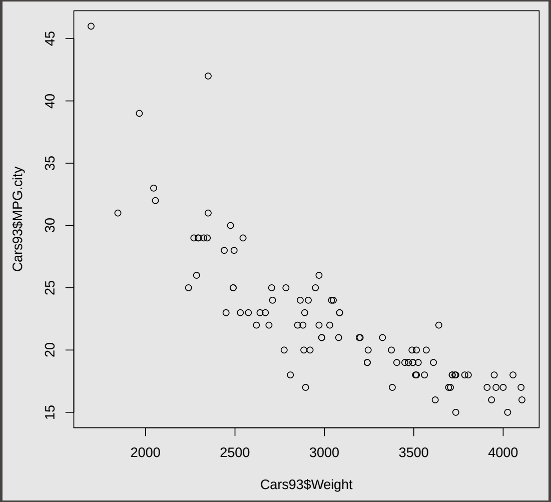
Error family	Default link	Link inverse	Use for:
gaussian	identity	1	Normally distributed error
poisson	log	exp	Counts
binomial	logit	$1/(1+e^{-x})$	Proportions or binary data
Gamma	inverse	$1/x$	Continuous data with non-constant error

```
> glm(formula, family = binomial(link = log))
```

GLM example

Consider the Cars93 data set.
Plotting the MPG in the city, versus Weight, suggests a non-linear relationship.

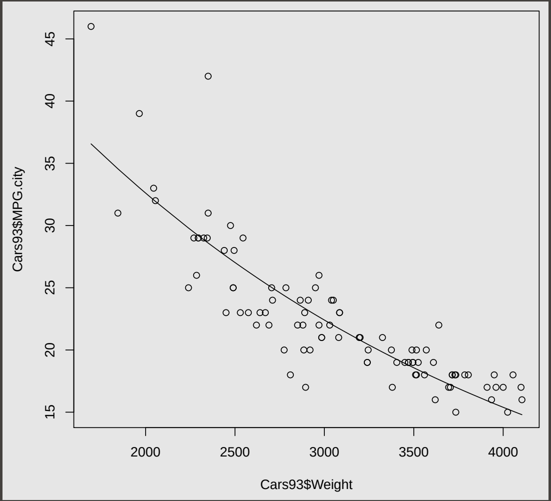
```
>
> library(MASS)
>
> plot(Cars93$Weight, Cars93$MPG.city)
>
```



GLM example, continued

Let's perform a GLM, using Gaussian noise and the log link function.

```
> sorted.weights <- sort(Cars93$Weight)
>
> glm1 <- glm(MPG.city ~ Weight,
+     data = Cars93,
+     family = gaussian(link = "log"))
>
> plot(Cars93$Weight, Cars93$MPG.city)
> lines(sorted.weights,
+     predict(glm1,
+     data.frame(Weight = sorted.weights),
+     type = "response"))
>
```



Summary

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

- Plot your data!
- Start with lm, both for linear and other polynomial fits.
- Plot the residuals! There is important information in there!
- If the data are not polynomial, or the residuals are not normally distributed, you may need to use a Generalized Linear Model.
- You will likely need to play around with the different noise families and link functions to find one that best works with your data.
- Other types of regression include logistic regression, for fitting categories, and multinomial regression, for multiple dependent variables.