

Brain.r: Explanation of code

Goals of code:

- Producing a scatterplot matrix
- Fitting a multiple regression
- Helper functions for multiple regression
- Regression diagnostics

The preparatory code to read the data and compute log transformed variables should be familiar.

Scatterplot matrix: `pairs()`

The `pairs()` function draws a scatter plot matrix. The required argument is a matrix or something that looks like a matrix. To `pairs()`, a data frame is something that looks like a matrix. Each variable is a column of this matrix. If you want to plot all variables in a matrix (or data frame), just provide the name (e.g. `pairs(brain)`).

If you try that with the brain data frame, `pairs()` will fuss because `species` is not a number. Even if you deleted `species`, the resulting plot is an 8 x 8 array of plots, each of which is quite small. Although 8 x 8 is not impossible to read, I find anything larger than 12 x 12, and sometimes 10 x 10, is very hard to read. The solution is to plot only a subset of the variables.

The 5 versions of `pairs()` illustrate different ways to extract a subset of the variables in a data frame. All of these are very general and can be used anytime you want to pull out a subset of variables. You can:

- Select the desired columns by number. `names()` will give you the names of each column, in order. Remember that `1:4` is shorthand for 1, 2, 3, 4. Remember that column indices go after the comma. `brain[1:4,]` is all columns but only rows 1 through 4; very different and probably not wanted.
- Omit a column(or columns) by number. Negative indices omit the specified column, so `brain[, -5]` omits the fifth column (`species`), so `pairs(brain[, -5])` will plot columns 1-4, 6-9.
- Select desired columns by name. You can select columns using a vector of character strings. All matching columns will be included, in the order specified in the vector. If you wanted to rearrange the order of the variables, you can specify your desired order, either as a names or numbers (e.g. `brain[, c(3,1,2,4)]`).

Note: You can't omit using names (at least I don't know how to) because `-'species'` isn't valid; you can only negate a number.

- Assemble a matrix with the desired columns. `cbind()` “binds” columns (the `c` in `cbind`). You specify the columns. This can be useful when you want to collect columns of information not stored in one object (e.g. predicted values, residuals, and X variables).

Note that `cbind()` loses the names of the variables. You can specify labels for each row/column using the `labels=` argument. If you use labels, you can specify anything you want, e.g. `labels=c('gestation length (m)', 'brain size (gm)', etc.)`

In all, the `main=` specifies the main title.

The last `pairs()` selects and plots the log transformed variables.

Estimating correlation coefficients: `cor()`

`cor()` calculates the correlation between all pairs of columns of the input. Like `pairs()`, the input can be a data frame or a matrix.

Fitting a multiple regression: `lm()`

The `lm()` function will fit multiple regressions: just specify each X variable separated by + signs.

Helper functions after fitting a multiple regression: various

The most useful information about the multiple regression is obtained by helper functions that manipulate or extract something you want from a fitted regression. There are many. Some of the ones that are relevant now are:

- `summary()`: Print a summary of the regression. This includes the model that was fit (“Call”), a five number summary of the residuals (which I find less useful), a table of information about the coefficients, and overall information about the fit.

The coefficient information includes the estimate, standard error, T statistic testing the hypothesis that the parameter = 0, and the associated p-value. The line labeled (Intercept) is for the intercept; the other lines are the regression slopes, labeled by the name of the X variable.

The first value in the overall information about the fit (labelled as residual standard error) is the root Mean Square Error with its df. (No, I don’t know why R calls this a standard error; I would call it a standard deviation). I de-emphasize R^2 and adjusted R^2 , but they are printed if you want them. The last line of information concerns the overall F test of all slopes = 0, with the F statistic, its df, and the p-value.

- `confint()`: If you want 95% confidence intervals for each regression parameter, use `confint()`. Add `, level=` to change the coverage.
- `drop1()`: If you want type III (partial) F tests of each parameter, you get them using the `drop1()` function. The default is to provide the information, but not do any test. Adding `test='F'`, with F in quotes, tells `drop1()` to do the F test and give you the p-value.
- `anova()`: If you want type I (sequential) SS and tests, you get these using `anova()`.

Lecture will discuss sequential (Type I) and partial (Type III) tests.

The two three-variable models (`brain.lm2` and `brain.lm2b`) illustrate that the type I (sequential) tests depend on the order of variables in the model while the type III (partial) tests do not.

All Type III tests: `Anova()`

The `car` library includes a function, `Anova()`, that computes type III tests for each model term. This is the quickest way to get type III tests and gives conclusions identical to `drop1()`.

Regression diagnostics:

These are provided by still more helper functions. See `?influence.measures` for a full list. All accept the result from `lm()` and return a vector with the requested measure. The result has one value for each observation. The functions we need and a few more that I'll mention are:

- `rstandard()`: Standardized residuals. Residual divided by its standard error
- `rstudent()`: externally standardized residuals. Residual divided by its standard error, where s is computed without the i 'th observation.
- `cooks.distance()`: Cook's Distance
- `dffits()`: DFFITS statistic

The usual way of plotting quantities like Cook's Distance is to use the observation number as the X variable. The line `nobs <- dim(brain)[1]` is one of various ways to extract the number of observations in the data set and save it as `nobs`. `dim()` returns a vector of two numbers (the "DIMensions") of the data set, i.e. the number of rows and number of columns. `[1]` extracts the first, the number of rows.

You could also use `length(d)`, which returns the length of a vector, i.e. the number of values in the vector. This is the number of observations.

Note: `length(brain)` does not work as you might expect. If `brain` is a data frame, this gives you the number of variables. If `brain` is a matrix, it gives you the total number of values (rows times columns).

Once you have the total number of observations, plotting values against the observation number is easy. Just use `1:nobs` as the X value, because `1:nobs` is the sequence: 1, 2, 3, \dots `nobs`.

The easiest way to get VIF, variance inflation factor, values for each variable is to use the `vif()` function in the `car` library. Note: there is a `VIF` library, but that does something different.