

Corn.r: Explanation of code

Goals of code:

- Using `with()`
- labeling data plots
- Fitting a polynomial regression using `proc reg`
- Creating groups from a continuous variable
- labeling data plots using coded symbols
- “Trellis” plot, using groups

Using `with()`: `with(something)`

It can get tedious to repeat `dataframe$` when you need to refer to many variables in the same data frame. `with()` allows you to omit that data frame name. The first argument is the name of a data frame in which to “look up” variables. The second argument is what you want to do. Usually this will be a function. Every variable in that second function call is interpreted as having a `dataframe$` preceding it. So, `with(corn, plot(rainfall, yield))` is equivalent to `plot(corn$rainfall, corn$yield)`.

labeling data points: `text()`

The `text()` function adds text to a pre-existing plot. The three required arguments are, in order, the X coordinate, the Y coordinate, and the text to place at that position. So `text(rainfall, yield, year)` will print each year value at the appropriate (rainfall, yield) location. By default, the text is placed on top of the point. Adding the `pos=` argument shifts the text to one side of the point. `pos=1` is below, `2` is to the left, `3` is above, and `4` is to the right of the point.

The size of the character string can be altered by adding the `cex=` argument (to any function that plots text). `cex=1` is the default size; all other values are proportions or multiples of that default size.

Note: in all cases, you need to draw the plot first. `Text()` adds to a pre-existing plot.

fitting a polynomial regression using `lm()`:

There are two common ways to fit a polynomial regression: one is to create the quadratic term in the data set, then use that variable in the `lm()` model. The quadratic term can be created either by multiplication (`corn2`) or by raising X to the power of 2 (`corn2a`). Those two variables have the same values.

The quadratic regression is fit by specifying both the linear and quadratic terms on the right hand side of the `~`, separated by `+`.

All the usual helper functions (see `brain.r` explanation) can be used. `summary()` gives you estimates and supporting information. The sequential SS, from `anova()`, are more useful than the partial SS from `drop1()`.

The second way to fit a quadratic model is to create the quadratic term “on the fly”. It would make sense to specify `rainfall*rainfall` to get the quadratic term. Unfortunately, the part of R that interprets formulae considers `rainfall*rainfall` to be the same as `rainfall`. There is a logic here, but the explanation is more than we need. The solution is the “protect” the arithmetic operation from the interpreter. That’s the role of the `I()`. Put the piece to be evaluated as an arithmetic operation inside `I()` and it will be evaluated as you intended (or at least as you wrote). If this gives you trouble or doesn’t work, just create the variable outside of `lm()` and add it to the model.

Creating groups: `cut()`

The `cut()` function breaks a numeric variable into groups. The first argument is the numeric variable. The second is the vector of breakpoints. These need to be extensive: the first breakpoint should be below the smallest data value; the last should be above the largest. R interprets the breaks as above the first value, up to and including the second.

The result is a factor with levels named from the breaks. You can provide your own names for the groups by adding the `labels=` argument and a vector of character strings.

A potentially useful detail: a factor is stored as a vector of integers with an additional piece that specifies the level corresponding to each integer. You can see this by `unclass()`, which “unclassifies” any object. You can see the effects of this by comparing the output when you print the corn data frame (printed as a data frame) and when you print `unclass(corn)`, which prints the list, which how that data frame is actually stored. If you `unclass` just the `yrgroup` variable, created by `cut()`, you get a vector of integers and a list of level names. The `attr()` piece is an attribute that can be attached to any object and specifies something additional about that object. In this case, the attribute is the list of levels for that factor.

labeling data points: `scatter / group=` ;

Instead of labeling points by their year, you can tell R to use some attribute of the point to indicate groups.

The `col=` argument specifies the color of the points. If this is a single number, that color is used for all points. If this is a vector with a value for each observation, each observation is plotted in the corresponding color. Specifying `col=yrgroup` takes advantage of the “integer-ness” of a factor to plot points in the 1890’s with `col=1` (black), points in the 1900’s with `col=2` (red), etc.

The `pch=` argument specifies the plotting character. 19 is a largish solid dot. The numbers 1 through 24 correspond to different symbols. You can also provide a single character (e.g. `pch='='`) will plot all points with an equals sign.

If you want to see all 24 plotting symbols, type the following into the R console:

```
plot(1:24, rep(1,24), pch=1:24)
```

This plots points at (1,1), (2,1), ..., (24,1) with symbols 1 through 24.

The `legend()` adds a legend indicating plot color. The arguments, in order of appearance, are the location, suppress the bounding box, show points, of specified colors, with labels grabbed from the levels of the factor variable.

You could use the plot symbol to mark the groups. For some reason, `pch=` requires the explicitly integer version of the factor, which is why you need `pch=unclass(corn$yrgroup)`. All points are plotted in blue (`col=4`). The `legend()` adds the appropriate key to the symbols.

Trellis plot: `xyplot()` in the lattice library

A “Trellis plot” plots Y vs X for subsets of the data. The example code plots the yield against rainfall separately for each decade. `proc sgpanel` requires three pieces of information to draw that plot: 1) the name of the variable that defines each subset. This goes in the `panelby` statement. The `/novarname` option suppresses the “decade=” information from the panel header. 2 & 3): the names of the X and Y variable, which go in the scatter statement.

You can ask for many other types of plots instead of a scatter plot in `proc sgpanel; panelby .` Just replace `scatter` ; by the plot you want (e.g., `series` ; to connect the dots).

Note: If you ask for `panelby year`, you will get one plot for each year; each plot will have one observation. Probably not a useful plot.