# Many Models in R

## Transcript video 2

Full resource: <https://www.ncrm.ac.uk/resources/online/all/?id=20840>

Okay, so at the end of the last video, we had a data frame called modspecs, which looked like this as it's printed to the console. It has 96 rows and five columns, and each row gives us the information that we need to specify a model that we want to run.

Now we have this specification grid, the next step is to take one of these rows and use it to create a regression model and then extract the results that we need. We're only really interested in the coefficient for the association between cognitive ability and BMI, so we don't need to return the full regression model object, as this would include things like the data that was used to run the model.

We're going to write the code out for one model first to be as general as possible so that it would work regardless of which row of the specification grid we fed into it. Once we've written the code once, we're going to turn it into a function, and that function will take a single input spec ID so that it can extract the appropriate row from the specification grid, and then run the appropriate model and get the appropriate results.

Again, it's easier to use spec ID than to pass all the information in separate arguments to the function, as this can get cumbersome, particularly where the number of columns in the specification grid is large. It is also more adaptable if we want to add further columns to the specification grid down the line. Let's start by setting spec ID to one, so that we have a particular row of data to work from. There we go.

Next, we want to use spec ID to obtain the relevant row from the specification grid. Here, I'm just going to use the filter function to return rows where a particular condition is true. So, we're going to take mod specs, and then we're going to filter to where the column spec ID is equal to spec ID. Now, this bit of code that I've just run isn't going to work, and the reason for this is that there's an external object called spec ID, which contains the value one, but then there's also a column within mod specs, which has the numbers, sorry, there's a column within mod specs with the column called spec ID, which goes from one to 96.

By default, the functions within the tidyverse are going to use the column within the data frame rather than an external object. So, this statement just says, give me back those rows where the column spec ID is equal to the column spec ID. If we want to refer to an external object, we can do so using the bang bang operator, which is this exclamation exclamation mark.

This explicitly tells R to look for the object in the external environment. So, now when we run this, we just get one row back where spec ID is equal to one. Let's save this object to spec, the specification that we want to use so that we can refer to it easily later. Okay, this is what spec looks like. We can extract columns from spec using the dollar sign operator. So, to get the FUP, the agent follow-up, we can just do spec dollar FUP.

To get the cognitive ability variable, we can do spec dollar sign COGVAR. Now, we're going to take the information contained in this single row of data to run a regression model. There's a few things we need to do.

First, we need to subset the DF data frame to get back just those observations we need to run the model, that is the data from a particular cohort and follow-up age. Second, we need to specify the model formula. That's the names of the outcome and the exposure variables and the sets of the covariance control variables that we want to add. Third, we want to run the model and extract the results that we need.

In this case, the coefficient for the cognitive ability variable. Let's first subset the DF data frame from the spec. The observations that we need are from the age 15 follow-up for the baby boomer cohort. We can do this subsetting using the filter function.

So, we'll take DF and then we'll pipe it into filter. We want to keep where cohort is equal to spec cohort. This has given us back too much information. It's given us 30,000 rows, but it's given us observations at age 15, 25, 35, and so on. We can add multiple filter statements within one function call just by separating them with columns, sorry, with commas.

These are combined with ands, so we're going to keep the rows that have cohort equal to baby boomer and FUP equal to 15. Great, so now we've got back the 5,000 rows that we need. They're just from this age 15 follow-up. Let's save this data frame as DF mod so that we can refer to it later. Okay, so the next thing we need to do is specify the formula for the model that we want to run. I'm going to use the glue package to do this because it's got a very straightforward syntax for constructing strings.

Glue is part of the tidyverse, but it's not a core tidyverse package, so it needs loading separately. So, we can just use library and then glue to load it. The main function within glue is itself called glue. I'm going to create an object called x, which is going to be a vector 1 to 7, yet containing the integers 1 to 7. I'm going to quickly use this just to demonstrate how glue works. So, that's x saved, and then we've got 1, 2, 3, 4, 5, 6, 7.

To create a string using glue, we just write out some text, and then within curly brackets, we provide the name of an object or we write a function. Glue will run R in the background and place the results that are within this curly brackets directly into the string. So, for instance, if we use glue to create a string containing the mean of 1 to 7, we can do that using curly brackets, mean x, and we can add some explanatory text to that. So, the mean of 1 to 7 is 4 because within curly brackets, this mean x, the mean of the object x is the mean of the vector 1 to 7, which is 4.

We can use this glue function to create model formulas. So, we want to regress BMI on a measure of cognitive ability plus some control variables.

Let's just have a quick look at spec again. So, the first thing we can do is add in the COGVAR. So, let's just do BMI curly brackets spec COGVAR. Let's just run that. Now, you can see we've got BMI tilde COGNONVERBAL. So, we've got the right variable specified there. We also want to add some control variables and we can specify another set of curly brackets, do MODCOVARS, double square brackets, and then SPECCOVARS.

So, this will replace SPECCOVARS with BASIC. MODCOVARS BASIC will just be the set of control variables for the basic model. So, now when we run this, we've got BMI COGNONVERBAL plus age plus female plus social class. And this gives us the correct formula string. We need to do one more thing though, which is to change this string from a string to a formula type object. And we can do this using the ASFORMULA function. Regression functions like LM, which you use for linear regression, require a formula object to specify the model.

So, let's just quickly pipe this into the ASFORMULA function. You can see it has an environment now, which shows you that it's a formula type object. Let's just quickly save that as MODFORMULA. Okay.

So, we now have the data in DFMOD and a model formula in MODFORMULA, which is all we need to run the regression model. Let's do that and save the result as an object called MOD. So, we have MOD, LM, MODFORMULA, and data equals DFMOD. Let's quickly just call SUMMARYMOD to see the results.

So, we've got an intercept, a coefficient for COGNONVERBAL, and coefficients for the other control variables. Now, this MOD object is quite complicated. It's got the data that we use to run the regression model and lots of other extraneous information. Really, as I said before, we just need the coefficient for COGNONVERBAL and some of the related estimates, like the standard error for that coefficient, the p-value, and the confidence intervals. To extract just this information, we can use the BROOM package. BROOM is another tidyverse package, but it's not a core one, so it needs loading explicitly.

So, LIBRARY BROOM. BROOM contains a set of functions that extract model information and put the results into data frames, which makes it particularly useful to work with. So, we're going to use the tidy function from BROOM, which takes a model object and returns a data frame with the coefficients and related estimates. So, that's TIDYMOD. I'm also just going to specify the confidence argument to return the confidence intervals. So, when we run that back, you'll see it returns a table with a row for each of the different coefficients of the model. So, we've got one for the intercept, COGNONVERBAL, and the three different control variables. This is still a bit more information than we need, so let's first use the FILTER function to get just the row for the COGVAR that we're interested in. So, we can pipe this into FILTER, and we can do TERM equals SPEC COGVAR. There we go.

It's also got a bit too much information in terms of the number of columns, so let's just use the SELECT function to return just those columns that we need. So, that's the estimate, the standard error, the p-value, the confidence intervals, CONF FLOW, and CONF HIGH. Great, that's everything we want.

So, in a few lines of code, we've been able to take a SPEC ID, use it to prepare data, and run a regression model, and then extract the appropriate results. Let's turn the code above into a function now, so that it can be reused. Let's call this function GETLM, and it will take a single input SPEC ID.

So, we do GETLM function, and then we'll just name the particular argument that we want, a single argument called SPEC ID, and I'm just going to copy and paste this stuff from above, and put it straight into the GETLM function. I'm going to tidy this up a slight bit, so I'm going to rename the result of the tidy function as RET. I'll put a return statement in as well. I'll remove SUMMARY MOD, because we no longer need it.

I don't need this bit about loading library glue and using the X's, so I can get rid of that, and also the stuff where I've just repeatedly called on SPEC. Okay, so let's just save that as GETLM, and then we can have a quick look at whether this works. So, let's just run GETLM1, and we've got the same result as we had before, so an estimate of 0.1969. We can also just change the input to 15 to get the result of the 15th row, so now we've got 0.48.

Great, and there you have it. That's the results for the first specification in the 15th. In the next video, we'll see how we concisely run this code for all the specifications in our specification grid.

Thank you.

National Centre for Research Methods (NCRM)  
Social Sciences  
Murray Building (Bldg 58)  
University of Southampton  
Southampton SO17 1BJ  
United Kingdom

**Web** www.ncrm.ac.uk   
**Email** info@ncrm.ac.uk  
**Tel** +44 23 8059 4539  
**Twitter** @NCRMUK