sat.r: Explanation

Goals of code:

- Defining a function
- Regression model selection
- Regression diagnostics

Note: Many things in this week's code require libraries that extend base R's capabilities. Remember that you need to download and install a library before you can use it. That is done with install.packages(), with the library name in quotes, or with the install packages menu item. I have commented out those commands, because they only need to be done once (until you install a new version of R).

You do need to access the library each time you start R. That is the library() command. If you get errors such as could not find function: regsubsets(), you have not yet linked the necessary library.

The beginning of the code in sat.r defines a function, called print.regsub, that prints model selection results in a nice way. Use of that function is explained in the model selection section. The ability to define a function to perform frequent tasks is one of the real strengths of R.

Reading an excel worksheet: (reminder) `read_excel( )`
The read.excel() function in the readxl package provides the ability to read a specific sheet from an excel workbook (.xls or .xlsx format file). The required argument is the name of the file (or you can use choose.files() to browse to the desired file). You can add more arguments to read a specific sheet, or specific parts of a specific sheet.

Regression model selection: `regsubsets( )`
The regsubsets() function in the leaps package provides all subsets regression. The arguments are the "fullest" model to be considered. This has the Y variable on the left-hand side and all variables to consider in the model on the right-hand side. The data= argument specifies the data set containing all the variables. The first two arguments are just like you were running lm().

You can use . on the right-hand side of the formula to indicate "all other variables". For the SAT analysis, you probably don't want to use this, because then both takers and ltakers are included in the potential variables. But, . can often be useful, especially if you create a new data frame with the subset of variables of interest.

`method='exhaustive'` requests all subsets regression. There are other methods, but I'm not emphasizing them.

`nbest=` specifies how many models to request **for each number of variables** in the model. Specifying nbest=3 will give you the three best 1 variable models, the three best 2 variable models, etc. through the one "all variables" model. I usually use nbest=3 or omit that argument. I omit it when I use the print.regsub() function to select the best models no matter how many variables they have.

The summary() function, when used on the output from regsubsets() prints a synopsis of the variables in each model. I don't find the default output very helpful.

Printing model selection results: `print.regsub( )`
I wrote a function to print the model selection results nicely, with more control over selection of "interesting" models. That is the print.regsub( ) function defined at the top of the sat.r file. The function definition starts with the `print.regsub() <- function( ) {` code and ends with the matching `}` on line 25 of sat.r.

The arguments to `print.regsub()` are:

- the object containing the results produced by summary() of a regsubsets result. In the example code, that is saved by `sat.sub2 <- summary(sat.sub)`.
- `sort=` (optional, default is 'BIC'): indicates which statistic should be used to sort results from smallest (best model) to largest (worst model). The name is in quotes and MUST match one of the names in the printed output from print.regsub(). Common choices will be 'Cp', 'AIC', or 'BIC'.
- `best=` (optional, default is now one model per number of X variables.): how many models to report on. best=5 will tell you about the best 5 models, without regard to the number of variables in them.

Note: If the `print.regsub(sat.sub2, best=5)` command gives you an error: could not find function "print.regsub", you have not defined the function (or there was an error when you tried). Rerun the code at the top of the file and see me if there any problems.

Fitting a large collection of models and finding the best 5-10:
My preferred way to use R for model selection is to get information about most or all possible models from regsubsets (i.e., set `nbest=` to a large number), then only report the best 5 or so using print.regsub(). Then when I have chosen a model, I refit that model using lm() with the appropriate variables and get diagnostics. The strategy here is to fit many 1 variable models, many 2 variable models, ... up to the one "all-variable" model. This is done by specifying `nbest=` a large number. The way regsubsets() fits models, `nbest=30` will give you the 30 best 1 variable models, the 30 best two variable models, $\cdots$. This isn't quite all possible models, but it is essentially that. If you really wanted all possible, set `nbest=` to a really large number. If you ask for a lot of models, R double checks that you really wanted a lot of models because that might be slow. Specifying `really.big=T` says that you really do. The rest of the code then prints a small number of best models according to the criterion you specify.

Regression diagnostics:
The last few lines of code show how to get a variety of multiple regression diagnostics. The four variable model with `ltakers + years + rank + expend` is one possible result from model selection. We then want to check there are no issues using this model. The model is fit using lm(). The stored result is used for all diagnostic functions.

- VIF: the `vif()` function in the car library. Argument is the lm() model result.
- externally standardized residuals: `rstudent()`. Returns residuals divided by its standard error. s is computed without the i'th observation.
- Cook's Distance: `cooks.distance()` Returns the vector of D values for each observation.
- PRESS statistic: This has to be computed "by hand". The formula is given in the code that computes the `pres` variable. Note that the lm() result is referenced in two places: in the resid() function and in the lm.influence() function. The values in pres are the "out of sample" residuals. The PRESS statistic is then computed as the sum of squared pres values. The root-mean-squared-error of prediction (out of sample prediction sd) is the square root of PRESS/N.