How to code for repeat analysis

This file demonstrates how to organise your code in a way which facilitates repeating your analysis with slightly different variables. The analysis here is basically a repeat of the Lecture 5 analysis. There is really no new analysis here, just a slight re-organisation of the code to help you for your project.

Preparing your workfile

We add the basic libraries needed for this week's work:

```r
library(tidyverse)    # for almost all data handling tasks
library(ggplot2)      # to produce nice graphiscs
library(stargazer)    # to produce nice results tables
library(haven)        # to import stata file
library(AER)          # access to HS robust standard errors
source("stargazer_HC.r")  # includes the robust regression display
```

Introduction

First we will go through the analysis we did previously. The data are an extract from the Understanding Society Survey (formerly the British Household Survey Panel).

```r
data_USoc <- read_dta("20222_USoc_extract.dta")
data_USoc <- as.data.frame(data_USoc)    # ensure data frame structure

# create factor variables
data_USoc$region <- as_factor(data_USoc$region)
data_USoc$male <- as_factor(data_USoc$male)
data_USoc$degree <- as_factor(data_USoc$degree)
data_USoc$race <- as_factor(data_USoc$race)

# create real hourly pay variable
data_USoc <- data_USoc %>%
              mutate(hrpay = paygu/(jbhrs*4)/(cpi/100)) %>%
              mutate(lnhrpay = log(hrpay))
```

The hourly pay variable is the dependent variable in our analysis later. In any project analysis you may have an alternative dependent variable. This may be because you are genuinly running two analyses, or in the calculation of the dependent variable there may be alternative ways of doing it and it is not immediately obvious that one is right and the other wrong. On this occasion we will calculate an hourly pay variable, but not adjusted for inflation.

```r
# create nominal hourly pay variable
data_USoc <- data_USoc %>%
              mutate(hrpay_nom = paygu/(jbhrs*4)) %>%
              mutate(lnhrpay_nom = log(hrpay_nom))
```

As we wanted to save these additional variables we assign the result of the operation to `data_USoc`.

In this lecture we will use the firmsize variable `mfsize9`. Let us first repeat the work we did with this variable previously.

```
# Name change
names(data_USoc)[names(data_USoc) == "mfsize9"] <- "fsize"
# Create factor
data_USoc$fsize_f <- as.factor(data_USoc$fsize)
```

At this stage we will create an alternative variable for the firm size which we will use later to change the analysis. In particular we want to use an alternative explanatory variable. Let's look at the different categories in `fsize_f`

```
table(data_USoc$fsize_f)
```

```
##
##    1    6   17   37   75  150  350  750 1500
## 2269 7722 9599 9095 6766 5814 6788 3768 7168
```

We will create a new variable `fsize_f2` which collects a few of these categories into new categories creating only 4 categories. To do this we use the `recode_factor`function when we create (`mutate`) a new variable.

```
data_USoc <- data_USoc %>% mutate(fsize_f2 = recode_factor(fsize_f, `1` = "<10",
                                              `6` = "<10",
                                              `17` = "10-100",
                                              `37` = "10-100",
                                              `75` = "10-100",
                                              `150` = "100-1000",
                                              `350` = "100-1000",
                                              `750` = "100-1000",
                                              `1500` = ">1000",
                                              .default = "NA"))
table(data_USoc$fsize_f2)
```

```
##
##      <10   10-100 100-1000    >1000
##     9991    25460    16370     7168
```

As you can see, we have now created fewer categories but with more observations in them.

Cleaning the dataset

As in the original analysis we will delete some observations with missing data.

```
data_USoc <- data_USoc %>% filter(!is.na(lnhrpay)) %>%
                            filter(!is.na(fsize))
```

Estimating the model

Let's estimate the regression model with a set of dummies (but not for bas category - firm size = 1) in `mod1`.

```
mod1 <- lm(lnhrpay~fsize_f, data = data_USoc)
```

Then we re-estimate the model but with the numerical `fsize` variable and we save this model as `mod4`.

```r
mod4 <- lm(lnhrpay~log(fsize), data = data_USoc)
stargazer_HC(mod1,mod4)
```

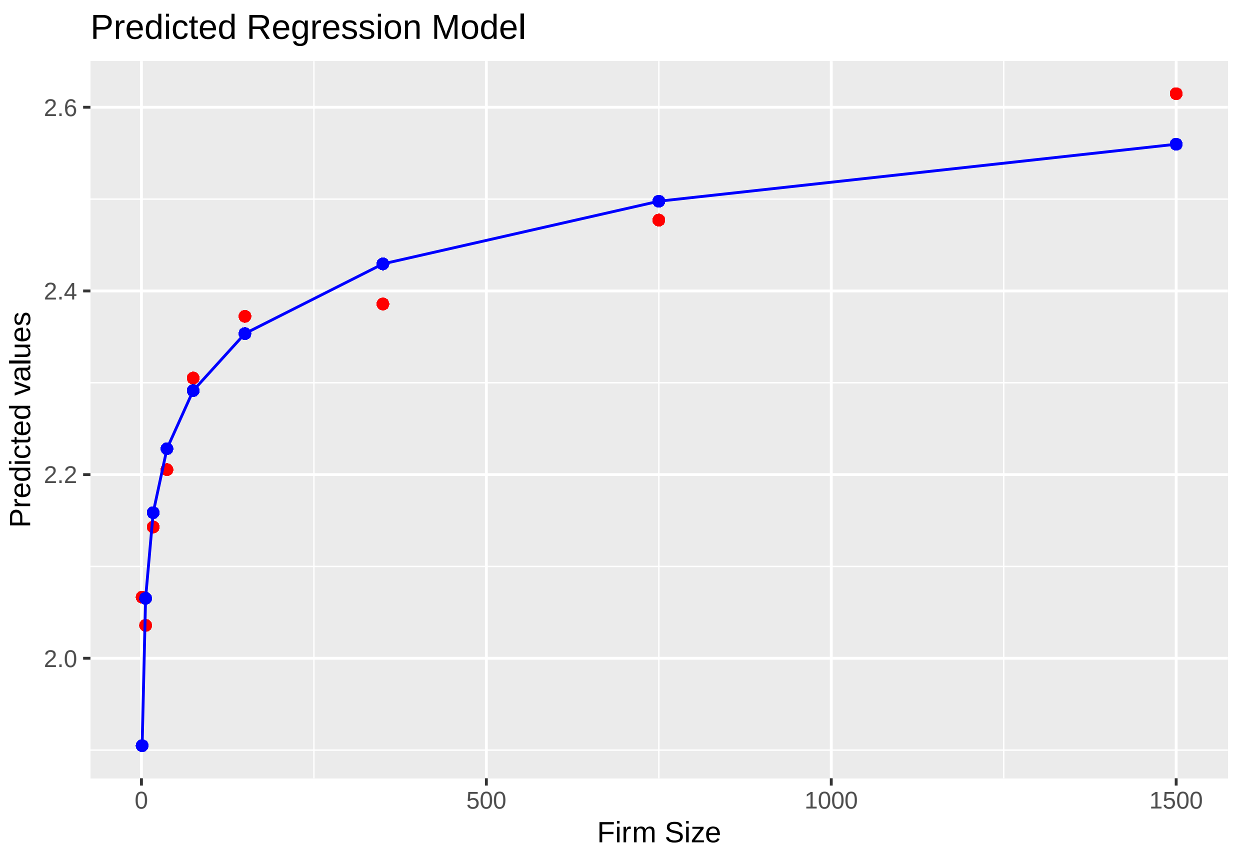
```
##
## ===========================================================================
##                                       Dependent variable:
##                        ----------------------------------------------------
##                                             lnhrpay
##                                  (1)                         (2)
## ---------------------------------------------------------------------------
## fsize_f6                       -0.031
##                               (0.020)
##
## fsize_f17                     0.076***
##                               (0.019)
##
## fsize_f37                     0.139***
##                               (0.020)
##
## fsize_f75                     0.239***
##                               (0.020)
##
## fsize_f150                    0.306***
##                               (0.020)
##
## fsize_f350                    0.319***
##                               (0.020)
##
## fsize_f750                    0.411***
##                               (0.021)
##
## fsize_f1500                   0.548***
##                               (0.020)
##
## log(fsize)                                                 0.090***
##                                                            (0.001)
##
## Constant                      2.067***                     1.905***
##                               (0.018)                      (0.007)
##
## ---------------------------------------------------------------------------
## Observations                   58,789                       58,789
## R2                             0.080                        0.075
## Adjusted R2                    0.080                        0.075
## Residual Std. Error      0.606 (df = 58780)          0.607 (df = 58787)
## F Statistic         636.491*** (df = 8; 58780) 4,763.519*** (df = 1; 58787)
## ===========================================================================
## Note:                                           *p<0.1; **p<0.05; ***p<0.01
##                                      Robust standard errors in parenthesis
```

Then we create the predicted values:

```r
data_USoc$pred_mod1 <- mod1$fitted.values
data_USoc$pred_mod4 <- mod4$fitted.values
```

Now we plot the predicted values for the two specifications. Recall that we only have 9 different values for the firm size.

```r
ggplot(data_USoc, aes(x=fsize,y=pred_mod1)) +
  geom_point(color = "red") +
  geom_point(aes(y=pred_mod4),color = "blue") +
  geom_line(aes(y=pred_mod4),color = "blue") +
  ggtitle("Predicted Regression Model") +
  ylab("Predicted values") +
  xlab("Firm Size")
```



So this was the analysis with `lnhrpay` as the dependent variable and `fsize_f` as the explanatory variable. In bigger projects there may be way more steps involved but this little example (2 regresison models, calculating predicted values and then graphing) serves to demonstrate the point.

We now have an alternative dependent variable `lnhrpay_nom` and an alternative firm size variable, `fsize_f2`. You could just copy and past the code and change the respective variables, but there are better ways to do this.

Estimating the model multiple times

I will actually present two ways of doing this.

Version A

First we shall replicate the above code in just one code block (no need to actually run this code here as we have already done all of this, also I commented out the regression output just to make this file a little shorter but you could of course keep it):

```r
mod1 <- lm(lnhrpay~fsize_f, data = data_USoc)
mod4 <- lm(lnhrpay~log(fsize), data = data_USoc)

# stargazer_HC(mod1,mod4)

data_USoc$pred_mod1 <- mod1$fitted.values
data_USoc$pred_mod4 <- mod4$fitted.values

ggplot(data_USoc, aes(x=fsize,y=pred_mod1)) +
  geom_point(color = "red") +
  geom_point(aes(y=pred_mod4),color = "blue") +
  geom_line(aes(y=pred_mod4),color = "blue") +
  ggtitle("Predicted Regression Model") +
  ylab("Predicted values") +
  xlab("Firm Size")
```

Here is the first way you could write your code to make it easy to repeat analysis. In essence you define two new variables (`depvar` and `expvar`) at the beginning of your code. In the first instance you could set them as being exactly the same variables we used before, but you could then also just change the definitions of the new variables `depvar` and `expvar` and just re-run the code below with the changed definitions. Importantly, as you can see below, we then have to use `depvar` and `expvar` in the actual commands for our analysis.

```r
# Define the dependent and explanatory variabl es
data_USoc$depvar <- data_USoc$lnhrpay_nom   # use either lnhrpay or lnhrpay_nom
data_USoc$expvar <- data_USoc$fsize_f2   # use either fsize_f or fsize_f2

mod1 <- lm(depvar~expvar, data = data_USoc)
mod4 <- lm(depvar~log(fsize), data = data_USoc)

# stargazer_HC(mod1,mod4)

data_USoc$pred_mod1 <- mod1$fitted.values
data_USoc$pred_mod4 <- mod4$fitted.values

ggplot(data_USoc, aes(x=fsize,y=pred_mod1)) +
  geom_point(color = "red") +
  geom_point(aes(y=pred_mod4),color = "blue") +
  geom_line(aes(y=pred_mod4),color = "blue") +
  ggtitle("Predicted Regression Model") +
  ylab("Predicted values") +
  xlab("Firm Size")
```

Version B - using a function

The above way of writing your code was actually very easy, it was really just a little renaming "trick". What I now demonstrate is a technique which is actually much more generic and as you become a more experienced coder you will often write your own functions. You can learn more about how to write functions from one of the excellent Software Carpentry tutorials.

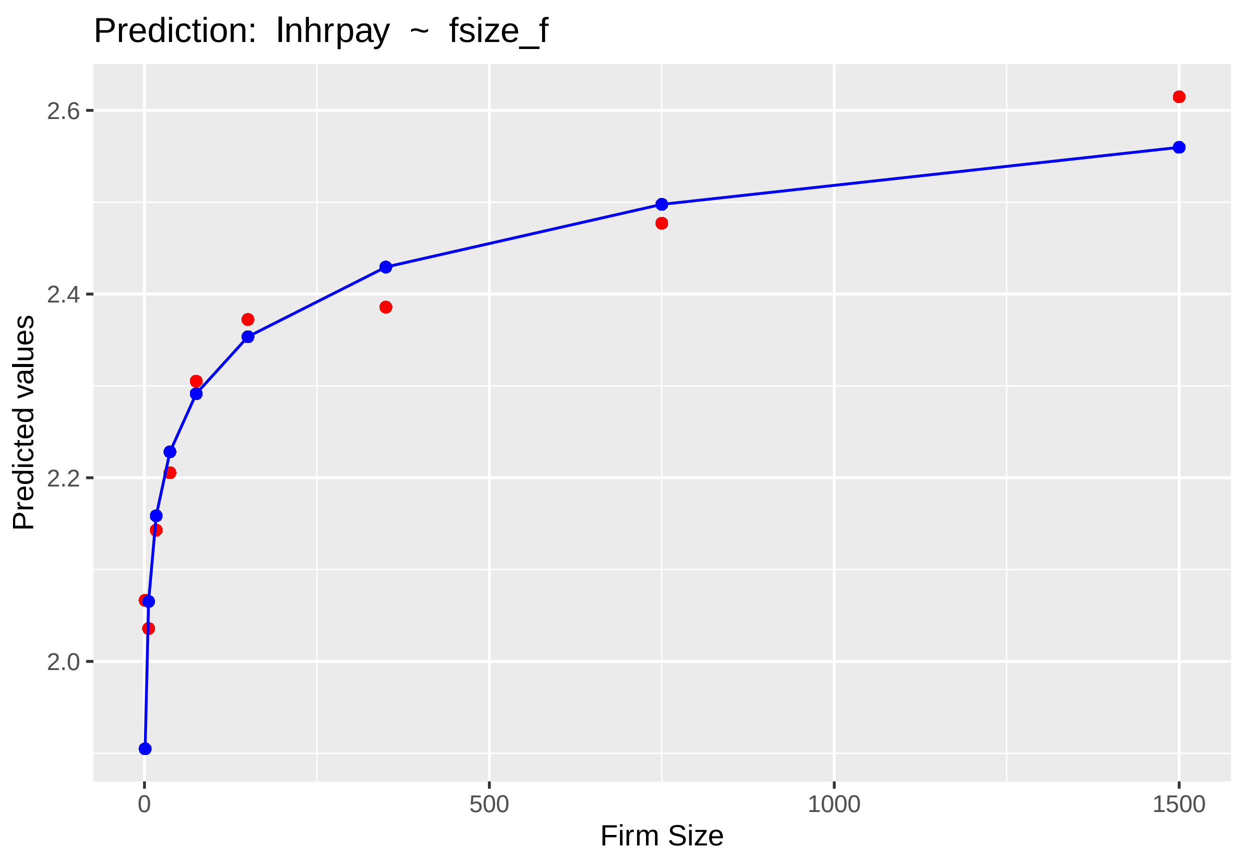
This is going to be a bespoke function for your own purpose.

```r
your_analysis <- function(depvar_name,expvar_name) {
# depvar_name: text with name of dependent variable
# expvar_name: text with name of explanatory variable
# requires data_USoc from which to draw the data
# Set the dependent and explanatory variabl

data_USoc$depvar <- data_USoc[,depvar_name]
data_USoc$expvar <- data_USoc[,expvar_name]

mod1 <- lm(depvar~expvar, data = data_USoc)
mod4 <- lm(depvar~log(fsize), data = data_USoc)

# stargazer_HC(mod1,mod4)

data_USoc$pred_mod1 <- mod1$fitted.values
data_USoc$pred_mod4 <- mod4$fitted.values

title_text <- paste("Prediction: ",depvar_name, " ~ ", expvar_name )
# we save the graph to an object p1
p1 <- ggplot(data_USoc, aes(x=fsize,y=pred_mod1)) +
  geom_point(color = "red") +
  geom_point(aes(y=pred_mod4),color = "blue") +
  geom_line(aes(y=pred_mod4),color = "blue") +
  ggtitle(title_text) +
  ylab("Predicted values") +
  xlab("Firm Size")

# we return the graphing object
return(p1)
}
```

By running this code you have saved the function, called 'your_analysis' (you could really call it anything you want), in your workspace. This function requires two inputs, which are called 'depvar_name' and 'expvar_name'. These are expected to be text variable names. Inside the function these names are then used to determine the 'depvar' and `expvar` used in the regressions. Then everything works as in Method A just that at the end we are saving the output in a variable 'p1' which is then returned by the function ('return(p1)'). Here we are chosing to return (what that means you will see in a minute) the graph. YOu could equally chose to return the estimated models or both the model and the graph. Details on how to achieve that you can find in the above link or if you google "R function returning multiple objects".

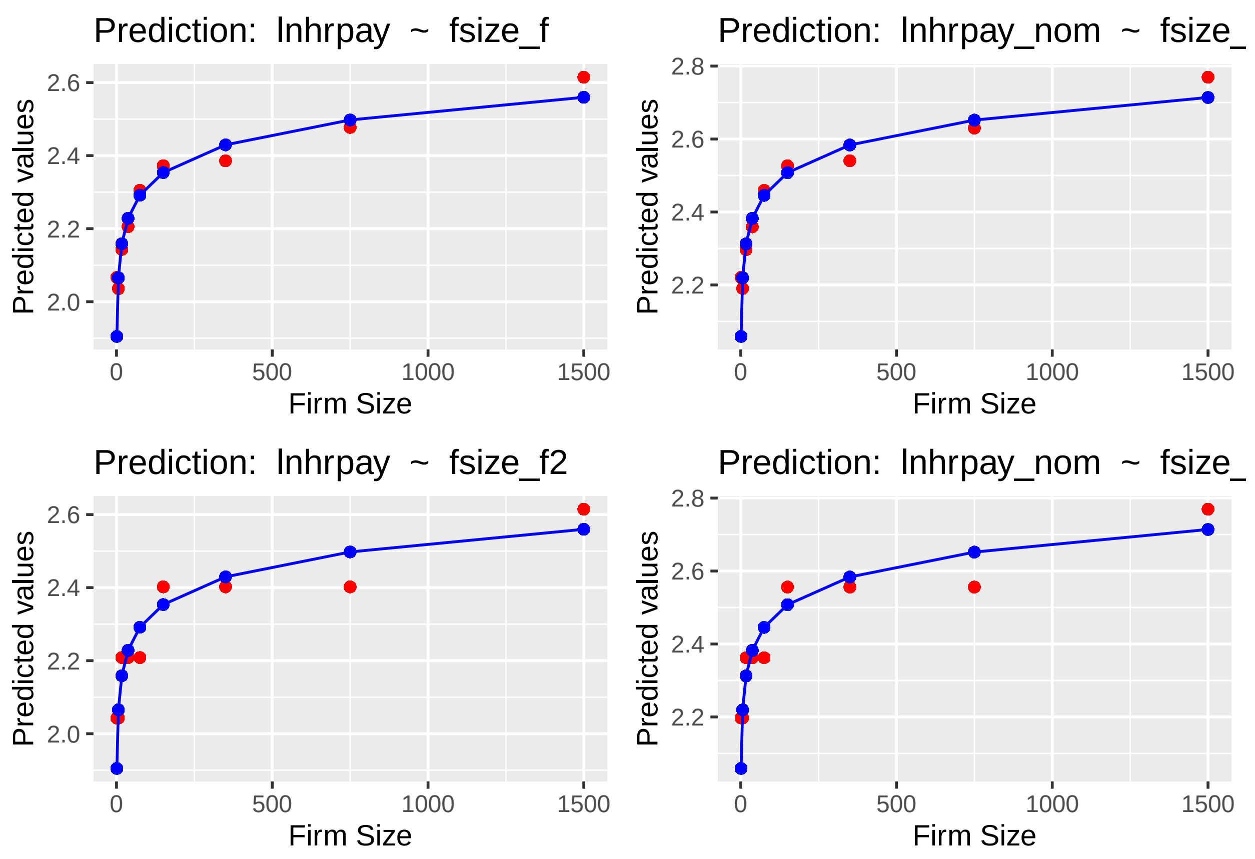
Just by running this code, you havn't actually estimated anything. You have basically laid down the recipe to be used to do some analysis. The two key ingredients, the dependent and the explanatory variable, are, as of now, not determined. So how do you do that and how do you now apply the analysis. Say you wanted to replicate the original analysis, then you would do that by using ' "lnhrpay" ' as the input for 'depvar_name' and'"fsize_f"'as 'expvar_name'.

```r
graph1 <- your_analysis("lnhrpay","fsize_f")  # this saves the analysis in graph1
graph1    # this plots graph1
```



So why is this a great way of working? Because it is now very straightforward to repeat the analysis for different variable combinations:

```
graph2 <- your_analysis("lnhrpay_nom","fsize_f")
graph3 <- your_analysis("lnhrpay","fsize_f2")
graph4 <- your_analysis("lnhrpay_nom","fsize_f2")

library(gridExtra) # Package required to arrange grids via grid.arrange
grid.arrange(graph1, graph2, graph3, graph4, nrow = 2)
```



Essentially what you want to achieve is that you write the actual code which does the analysis only once but then call that code several times. Method A did that by re-running the same code and just making sure that you only had to change as few lines as possible. Method B delivered a more generic way of achieving that.

You should note a couple of things here. For the function to work you also need to ensure that `data_USoc` is in your environment as you call the function as the function uses that dataframe. This makes it also obvious that this function is very specialised to your particular problem. The programmers which have written all the useful functions you have been using, like the `grid.arrange` function, usually do make sure that everything that is needed by the function is handed it via function inputs. But then they do write functions for more general use.