This is the variable description:

1. inlf =1 if in labor force, 1975

2. hours hours worked, 1975

3. kidslt6 # kids < 6 years

4. kidsge6 # kids 6-18

5. age woman's age in yrs

6. educ years of schooling

7. wage estimated wage from earns., hours

8. repwage reported wage at interview in 1976

9. hushrs hours worked by husband, 1975

10. husage husband's age

11. huseduc husband's years of schooling

12. huswage husband's hourly wage, 1975

13. faminc family income, 1975

14. mtr fed. marginal tax rate facing woman

15. motheduc mother's years of schooling

16. fatheduc father's years of schooling

17. unem unem. rate in county of resid.

18. city =1 if live in SMSA

19. exper actual labor mkt exper

20. nwifeinc (faminc - wage\*hours)/1000

21. lwage log(wage)

22. expersq exper^2

The goal is to first impute missing data in the data set, and then determine the impact of one added  
year of education on wages. If one simply ignores missing values, bias can be introduced depending on  
the missingness mechanism. The second problem here is that education is likely to be endogeneous  
(and thus be correlated to the error term), as it is not randomly assigned. This causes biased estimates  
and may lead to seriously wrong conclusions. So missingness and endogeneity should be dealt with, but  
dealing with both issues is more of a programming challenge than an econometrics challenge.  
Thankfully, the packages contained in the {tidyverse} as well as {mice} will save the day!

If you inspect the data, you will see that there are no missing values. So I will use the {mice}  
package to first *ampute* the data (which means adding missing values). This, of course, is done  
for education purposes. If you’re lucky enough to not have missing values in your data, you shouldn’t  
add them!

Let’s load all the packages needed:

library(tidyverse)

library(AER)

library(naniar)

library(mice)

So first, let’s read in the data, and ampute it:

wages\_data <- read\_csv("http://eclr.humanities.manchester.ac.uk/images/5/5f/Mroz.csv")

## Parsed with column specification:

## cols(

## .default = col\_integer(),

## wage = col\_character(),

## repwage = col\_double(),

## huswage = col\_double(),

## mtr = col\_double(),

## unem = col\_double(),

## nwifeinc = col\_double(),

## lwage = col\_character()

## )

## See spec(...) for full column specifications.

First, I only select the variables I want to use and convert them to the correct class:

wages\_data <- wages\_data %>%

select(wage, educ, fatheduc, motheduc, inlf, hours,

kidslt6, kidsge6, age, huswage,

mtr, unem, city, exper) %>%

mutate\_at(vars(kidslt6, kidsge6, hours, educ, age, wage, huswage, mtr,

motheduc, fatheduc, unem, exper), as.numeric) %>%

mutate\_at(vars(city, inlf), as.character)

## Warning in evalq(as.numeric(wage), ): NAs introduced by

## coercion

In the data, some women are not in the labour force, and thus do not have any wages; meaning they  
should have a 0 there. Instead, this is represented with the following symbol: “.”. So I convert  
these dots to 0. One could argue that the wages should not be 0, but that they’re truly missing.  
This is true, and there are ways to deal with such questions (Heckman’s selection model for instance),  
but this is not the point here.

wages\_data <- wages\_data %>%

mutate(wage = ifelse(is.na(wage), 0, wage))

Let’s double check if there are any missing values in the data, using naniar::vis\_miss():

vis\_miss(wages\_data)

Nope! Let’s ampute it:

wages\_mis <- ampute(wages\_data)$amp

## Warning: Data is made numeric because the calculation of weights requires

## numeric data

ampute() returns an object where the amp element is the amputed data. This is what I save into  
the new variable wages\_mis.

Let’s take a look:

vis\_miss(wages\_mis)

Ok, so now we have missing values. Let’s use the recently added mice::parlmice() function to  
impute the dataset, in parallel:

imp\_wages <- parlmice(data = wages\_mis, m = 10, maxit = 20, cl.type = "FORK")

For reproducibility, I save these objects to disk:

write\_csv(wages\_mis, "wages\_mis.csv")

saveRDS(imp\_wages, "imp\_wages.rds")

As a sanity check, let’s look at the missingness pattern for the first completed dataset:

vis\_miss(complete(imp\_wages))

mice::parlmice() was able to impute the dataset. I imputed it 10 times, so now I have 10 imputed  
datasets. If I want to estimate a model using this data, I will need to do so 10 times.  
This is where the tidyverse comes into play. First, let’s combine all the 10 imputed datasets into  
one long dataset, with an index to differentiate them. This is done easily with mice::complete():

imp\_wages\_df <- mice::complete(imp\_wages, "long")

Let’s take a look at the data:

head(imp\_wages\_df)

## .imp .id wage educ fatheduc motheduc inlf hours kidslt6 kidsge6 age

## 1 1 1 3.3540 12 7 12 1 1610 1 0 32

## 2 1 2 1.3889 12 7 7 1 1656 0 2 30

## 3 1 3 4.5455 12 7 12 1 1980 0 3 35

## 4 1 4 1.0965 12 7 7 1 456 0 3 34

## 5 1 5 4.5918 14 14 12 1 1568 1 2 31

## 6 1 6 4.7421 12 7 14 1 2032 0 0 54

## huswage mtr unem city exper

## 1 4.0288 0.7215 5.0 0 14

## 2 8.4416 0.6615 11.0 1 5

## 3 3.5807 0.6915 5.0 0 15

## 4 3.5417 0.7815 5.0 0 6

## 5 10.0000 0.6215 9.5 1 14

## 6 4.7364 0.6915 7.5 1 33

As you can see, there are two new columns, .id and .imp. .imp equals i means that it is the  
ith imputed dataset.

Because I have 0’s in my dependent variable, I will not log the wages but instead use the Inverse  
Hyperbolic Sine transformation.

ihs <- function(x){

log(x + sqrt(x\*\*2 + 1))

}

I can now apply this function, but first I have to group by .imp. Remember, these are 10 separated  
datasets. I also create the experience squared:

imp\_wages\_df <- imp\_wages\_df %>%

group\_by(.imp) %>%

mutate(ihs\_wage = ihs(wage),

exper2 = exper\*\*2)

Now comes some tidyverse magic. I will create a new dataset by using the nest() function from tidyr.

(imp\_wages <- imp\_wages\_df %>%

group\_by(.imp) %>%

nest())

## # A tibble: 10 x 2

## .imp data

##

## 1 1

## 2 2

## 3 3

## 4 4

## 5 5

## 6 6

## 7 7

## 8 8

## 9 9

## 10 10

As you can see, imp\_wages is now a dataset with two columns: .imp, indexing the imputed datasets,  
and a column called data, where each element is itself a tibble! data is a so-called list-column.  
You can read more about it on the purrr package.

Estimating a model now is easy, if you’re familiar with purrr. This is how you do it:

imp\_wages\_reg = imp\_wages %>%

mutate(lin\_reg = map(data,

~lm(ihs\_wage ~ educ + inlf + hours +

kidslt6 + kidsge6 + age + huswage +

mtr + unem + city + exper + exper2,

data = .)))

Ok, so what happened here? imp\_wages is a data frame, so it’s possible to add a column to it  
with mutate(). I call that column lin\_reg and use map() on the column called data (remember,  
this column is actually a list of data frame objects, and map() takes a list as an argument, and then a  
function or formula) with the following formula:

~lm(ihs\_wage ~ educ + inlf + hours +

kidslt6 + kidsge6 + age + huswage +

mtr + unem + city + exper + exper2,

data = .)

This formula is nothing more that a good old linear regression. The last line data = . means that  
the data to be used inside lm() should be coming from the list called data, which is the second  
column of imp\_wages. As I’m writing these lines, I realize it is confusing as hell. But I promise  
you that learning to use purrr is a bit like learning how to use a bicycle. Very difficult to explain,  
but once you know how to do it, it feels super natural. Take some time to play with the lines above  
to really understand what happened.

Now, let’s take a look at the result:

imp\_wages\_reg

## # A tibble: 10 x 3

## .imp data lin\_reg

##

## 1 1

## 2 2

## 3 3

## 4 4

## 5 5

## 6 6

## 7 7

## 8 8

## 9 9

## 10 10

imp\_wages\_reg now has a third column called lin\_reg where each element is a linear model, estimated  
on the data from the data column! We can now pool the results of these 10 regressions using  
mice::pool():

pool\_lin\_reg <- pool(imp\_wages\_reg$lin\_reg)

summary(pool\_lin\_reg)

## estimate std.error statistic df p.value

## (Intercept) 1.2868701172 3.214473e-01 4.0033628 737.9337 6.876133e-05

## educ 0.0385310276 8.231906e-03 4.6806931 737.9337 3.401935e-06

## inlf 1.8845418354 5.078235e-02 37.1101707 737.9337 0.000000e+00

## hours -0.0001164143 3.011378e-05 -3.8658143 737.9337 1.204773e-04

## kidslt6 -0.0438925013 3.793152e-02 -1.1571510 737.9337 2.475851e-01

## kidsge6 -0.0117978229 1.405226e-02 -0.8395678 737.9337 4.014227e-01

## age -0.0030084595 2.666614e-03 -1.1281946 737.9337 2.596044e-01

## huswage -0.0231736955 5.607364e-03 -4.1327255 737.9337 3.995866e-05

## mtr -2.2109176781 3.188827e-01 -6.9333267 737.9337 8.982592e-12

## unem 0.0028775444 5.462973e-03 0.5267360 737.9337 5.985352e-01

## city 0.0157414671 3.633755e-02 0.4332011 737.9337 6.649953e-01

## exper 0.0164364027 6.118875e-03 2.6861806 737.9337 7.389936e-03

## exper2 -0.0002022602 1.916146e-04 -1.0555575 737.9337 2.915159e-01

This function averages the results from the 10 regressions and computes correct standard errors. This  
is based on Rubin’s rules (Rubin, 1987, p. 76). As you can see, the linear regression indicates that  
one year of added education has a positive, significant effect of log wages (they’re not log wages,  
I used the IHS transformation, but *log wages* just sounds better than *inverted hyperbolic sined wages*).  
This effect is almost 4%.

But education is not randomly assigned, and as such might be endogenous. This is where instrumental  
variables come into play. An instrument is a variables that impacts the dependent variable only through  
the endogenous variable (here, education). For example, the education of the parents do not have  
a direct impact over one’s wage, but having college-educated parents means that you are likely  
college-educated yourself, and thus have a higher wage that if you only have a high school diploma.

I am thus going to instrument education with both parents’ education:

imp\_wages\_reg = imp\_wages\_reg %>%

mutate(iv\_reg = map(data,

~ivreg(ihs\_wage ~ educ + inlf + hours +

kidslt6 + kidsge6 + age + huswage +

mtr + unem + city + exper + exper2 |.-educ + fatheduc + motheduc,

data = .)))

The only difference from before is the formula:

~ivreg(ihs\_wage ~ educ + inlf + hours +

kidslt6 + kidsge6 + age + huswage +

mtr + unem + city + exper + exper2 |.-educ + fatheduc + motheduc,

data = .)

## ~ivreg(ihs\_wage ~ educ + inlf + hours + kidslt6 + kidsge6 + age +

## huswage + mtr + unem + city + exper + exper2 | . - educ +

## fatheduc + motheduc, data = .)

Instead of lm() I use AER::ivreg() and the formula has a second part, after the | symbol. This  
is where I specify that I instrument education with the parents’ education.

imp\_wages\_reg now looks like this:

imp\_wages\_reg

## # A tibble: 10 x 4

## .imp data lin\_reg iv\_reg

##

## 1 1

## 2 2

## 3 3

## 4 4

## 5 5

## 6 6

## 7 7

## 8 8

## 9 9

## 10 10

Let’s take a look at the results:

pool\_iv\_reg <- pool(imp\_wages\_reg$iv\_reg)

summary(pool\_iv\_reg)

## estimate std.error statistic df p.value

## (Intercept) 2.0091904157 5.146812e-01 3.9037568 737.9337 1.033832e-04

## educ 0.0038859137 2.086592e-02 0.1862326 737.9337 8.523136e-01

## inlf 1.9200207113 5.499457e-02 34.9129122 737.9337 0.000000e+00

## hours -0.0001313866 3.157375e-05 -4.1612608 737.9337 3.537881e-05

## kidslt6 -0.0234593391 4.000689e-02 -0.5863824 737.9337 5.577979e-01

## kidsge6 -0.0123239220 1.422241e-02 -0.8665145 737.9337 3.864897e-01

## age -0.0040874625 2.763340e-03 -1.4791748 737.9337 1.395203e-01

## huswage -0.0242737100 5.706497e-03 -4.2536970 737.9337 2.373189e-05

## mtr -2.6385172445 3.998419e-01 -6.5989008 737.9337 7.907430e-11

## unem 0.0047331976 5.622137e-03 0.8418859 737.9337 4.001246e-01

## city 0.0255647706 3.716783e-02 0.6878197 737.9337 4.917824e-01

## exper 0.0180917073 6.258779e-03 2.8906127 737.9337 3.957817e-03

## exper2 -0.0002291007 1.944599e-04 -1.1781381 737.9337 2.391213e-01