B2 - Econometrics - HWA2 Example

Lukas Arnroth

12 juni 2017

Getting Started

It's good practise to begin your code file with the packages you will use in your code. If you want to install a package the function you use is *install.package*(). Within the paranthesis you supply a string with the package name. For example *install.package*("*fmsb*") will install the package fmsb. Below you find the packages I will use in this example that you can use as an outline for your own assignment.

```
library(normtest) # for Jarque-Bera test of normality
library(ggplot2) # for plots
library(fmsb) # for VIF
```

Loading the Data

Next we look at loading the data. The data used in this example is emigration and measures of poverty(tenant farming), population pressure (farm size) and information (gov't workers in Hawaii) for 45 prefectures in Japan. [1]. Below you find a description of the variables

Variable	Description
X_1	prefecture
Y	emigrants per 1 million residents
X_2	% land cultivated by tenant farmers
X_3	% change in ratio of tenant farmlands
X_4	avarage area of arable land per farm
X_5	number of government labors workin in Hawaii in 100s
X_6	existence of pioneer immigrants 1=y, 0=n

Loading the .csv file is done using the *read.csv*() function in R. The data is then assigned to an object conveniently named "data" which will be stored in the global environment. The environments of R are simply places to store variables and objects. The global environment is located on the upper right hand side of your RStudio window and is the only environment you will have to consider on the B-level courses.

```
data <- read.csv("japanese_emmigration.txt", sep = ";")
```

Note two things within the parenthesis in the read.csv call. Firstly we find *sep* = ";" in the function call. Looking at the help documentation of the *read.csv*() function is the best way of understanding what the parameter does

[1] Y. Murayama (1991). "Information and Emigrants: Interprefectural Differences of Japanese Emigration to the Pacific Northwest, 1880-1915", The Journal of Economic History, Vol. 51

```r
?read.csv()
```

The R documentation states that the parameter specifices the character which seperates the values in .csv file. If wrongly stated R will read the whole data as one long character string. If you're uncertain about what character seperates the values in your data, you can simply open the .csv file in your notepad to see what seperates the values. Also note that in the *read.csv*() call the location of the data file is not specified. In order to see your working directory (where R looks for files if you dont explicitly state their location) you can use the *getwd*() function

```r
getwd()
```

```
## [1] "C:/Users/lukas/Desktop/B2/B-courses_R/B2_HWA2"
```

This is the correct working directory for me. The working directory can be set manually on windows as

```r
setwd("C:/Users/lukas/Desktop/B-course-R/B2_HWA2")
```

Changing the Variable Names

So now that the data has been read we can begin by looking at the variable names

```r
colnames(data)
```

```
## [1] "X1" "Y"  "X2" "X3" "X4" "X5" "X6"
```

Not very informative. We can change this by assigning a concatonate object (something often refered to as a vector although not strictly true in a linear algebra sense) with new, more informative, variable names

```r
colnames(data) <- c("prefecture","emigrants","cultivated","farmland",
                    "arable","labors","pioneer")
colnames(data)
```

```
## [1] "prefecture" "emigrants"  "cultivated" "farmland"   "arable"
## [6] "labors"     "pioneer"
```

Section 2.1 of the R-introduction covers the vector type object.

Quick and Dirty Investigation of the Data

Two useful functions once the data has been read and stored in the global environment is the structure function, *str*(), and the summary function, *summary*(). The structure function list the variables of your data frame along with what type (character, integer etc.) each variable is and the first 10 observations

```r
str(data)
```

```
## 'data.frame':    45 obs. of  7 variables:
##  $ prefecture: Factor w/ 45 levels "Aichi","Akita",..: 3 2 15 23 43 8 37 13 10 33 ...
##  $ emigrants : int  59 8 55 51 32 173 52 38 53 15 ...
##  $ cultivated: int  49 50 29 39 43 27 34 42 39 46 ...
##  $ farmland  : num  16.6 14.8 14.7 17 8.4 17.3 10.9 16.2 12.8 0.1 ...
##  $ arable    : num  165 163 159 131 145 ...
##  $ labors    : num  0 0 0 0.01 0 0 0.02 0 0.1 0 ...
##  $ pioneer   : int  0 0 0 0 0 1 0 0 0 0 ...
```

It's good practise to already here think of whether the level of some variable is wrongly specified. If you don't change this by hand, somewhere down the line R might make an implicit type conversion which sometimes can be catastrophic. Say for example that i don't want prefecture to be a factor, but rather a character type. I would then change it in the following way

```r
data$prefecture <- as.character(data$prefecture)
str(data)
```

```
## 'data.frame':    45 obs. of  7 variables:
##  $ prefecture: chr  "Aomori" "Akita" "Iwate" "Miyagi" ...
##  $ emigrants : int  59 8 55 51 32 173 52 38 53 15 ...
##  $ cultivated: int  49 50 29 39 43 27 34 42 39 46 ...
##  $ farmland  : num  16.6 14.8 14.7 17 8.4 17.3 10.9 16.2 12.8 0.1 ...
##  $ arable    : num  165 163 159 131 145 ...
##  $ labors    : num  0 0 0 0.01 0 0 0.02 0 0.1 0 ...
##  $ pioneer   : int  0 0 0 0 0 1 0 0 0 0 ...
```

In the first line of code, on the left hand side of the get-symbol ("<-") I tell R that my result should be applied to the prefecture variable of data. The variables are specified using "$" after the name of the data frame. The *as.character*() function tells R to convert the variable to a character type object. If you want to understand how the assignment operator works in words it can often be most informative to read it from right to left: *take the prefecture variable in the object data and make it into character type then assign it to the prefecture variable in the object data.*

Next we have the summary function which gives some summary statistic of each variable.

```r
summary(data)
```

```
##   prefecture          emigrants       cultivated       farmland     
##  Length:45          Min.   :  8.0   Min.   :27.00   Min.   :-5.200  
##  Class :character   1st Qu.: 53.0   1st Qu.:40.00   1st Qu.: 4.100  
##  Mode  :character   Median : 84.0   Median :44.00   Median : 8.300  
##                     Mean   :178.7   Mean   :44.87   Mean   : 7.924  
##                     3rd Qu.:233.0   3rd Qu.:49.00   3rd Qu.:10.900  
##                     Max.   :895.0   Max.   :70.00   Max.   :23.200  
##      arable           labors           pioneer      
##  Min.   : 53.90   Min.   :  0.000   Min.   :0.0000  
##  1st Qu.: 74.20   1st Qu.:  0.000   1st Qu.:0.0000  
##  Median : 90.50   Median :  0.000   Median :0.0000  
##  Mean   : 95.48   Mean   :  6.464   Mean   :0.2889  
##  3rd Qu.:111.50   3rd Qu.:  0.200   3rd Qu.:1.0000  
##  Max.   :164.80   Max.   :111.240   Max.   :1.0000
```

Always take an extra second here and investigate the summary statistics. Does anything look out of place? In my data I notice nothing out of order.

Model Fit

Now I will move forward estimating the regression functions. I will look at estimating three functions

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + u_i \tag{1}$$

$$Y_i = \theta_0 + \theta_1 X_{1i} + \theta_2 X_{2i} + \theta_3 X_{3i} + u_i \tag{2}$$

$$Y_i = \gamma_0 + \gamma_1 X_{1i} + \gamma_2 X_{2i} + \gamma_3 X_{3i} + \gamma_4 X_{4i} + u_i \tag{3}$$

As was mentioned in the help document for the first assignment, estimating a linear regression model in R is done using the linear model function, $lm()$. Section 11.1-2 of the R-introduction covers the topic of linear models more closely. In this example we supply the function with the data frame we're using, which we named "data", aswell as the formula. Remember that the default setting is to include an intercept.

```
# model 1
m1 <- lm(data = data, emigrants ~ cultivated + farmland)
# model 2
m2 <- lm(data = data, emigrants ~ cultivated + farmland + arable)
# model 3
m3 <- lm(data = data, emigrants ~ cultivated + farmland + arable + labors)
```

Using Transformed Variables

It is straightforward to use transformed variables. If I wanted to estimate model 1 using logarithms of each variable it is done as follows

```
m1_log <- lm(data = data, log(emigrants) ~ log(cultivated) + log(farmland))
```

Model Evaluation

Let's look at how we would move forward evaluating the first model. This is done using the summary function (the summary function is a base function which has many unique interactions with different types of objects, section 11.3 of the R-introduction covers such functions and their specific interaction with a lm object)

```
summary(m1)
```

```
##
## Call:
## lm(formula = emigrants ~ cultivated + farmland, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -198.18 -126.87  -86.56   51.09  724.00
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 170.4049   176.2774   0.967    0.339
## cultivated    0.9391     3.7530   0.250    0.804
## farmland     -4.2691     5.2297  -0.816    0.419
```

```
##
## Residual standard error: 209.3 on 42 degrees of freedom
## Multiple R-squared:  0.01708,    Adjusted R-squared:  -0.02973
## F-statistic: 0.3649 on 2 and 42 DF,  p-value: 0.6964
```

Here we find the estimates along with some useful statistics such as R^2 (Multiple R-squared) and the F-statistic. You should also be able to see that no variable is significantly different from 0 on the 5%-level.

Let's check the assumption of normality of the residuals. This is done using the function *jb.norm.test* from the norm test package. This is the Jarque-Bera test of normality. In order to do this I will have to find some way to extract the residuals from the m1 object which we stored the first model in. This can be done in two ways

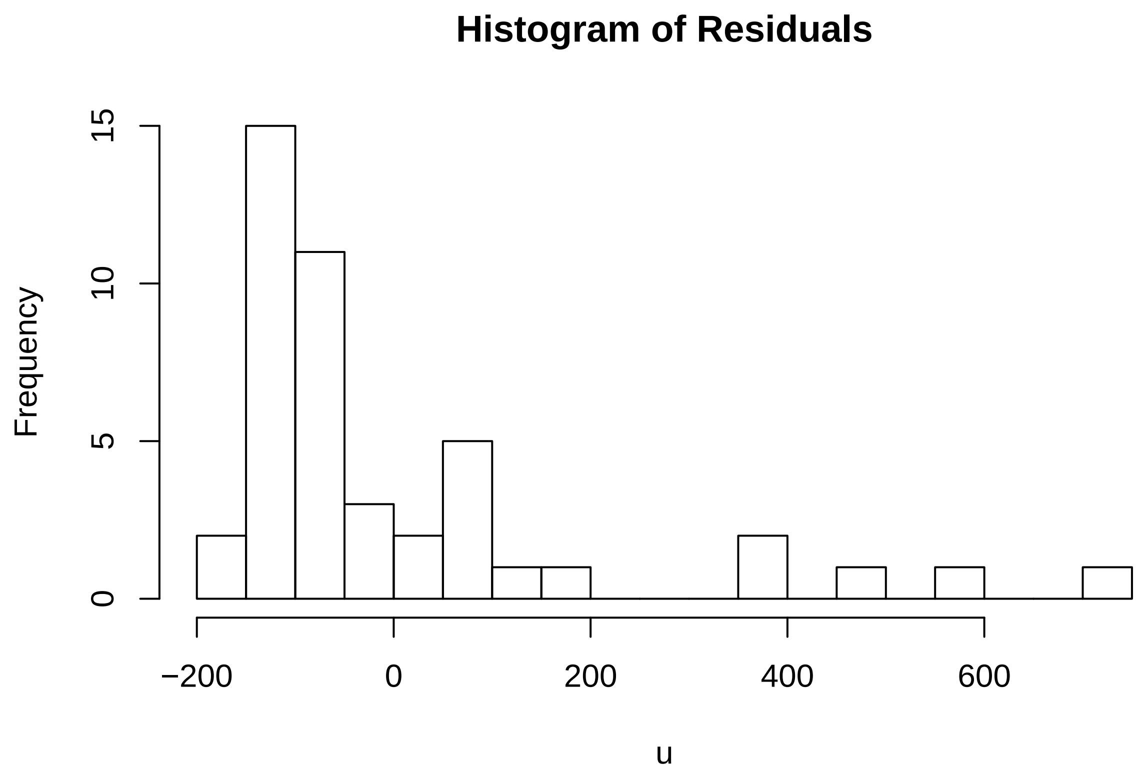
```r
u <- m1$residuals
# or
u <- residuals(m1)
# performing the test
jb.norm.test(u)
```

```
##
##  Jarque-Bera test for normality
##
## data:  u
## JB = 53.768, p-value < 2.2e-16
```

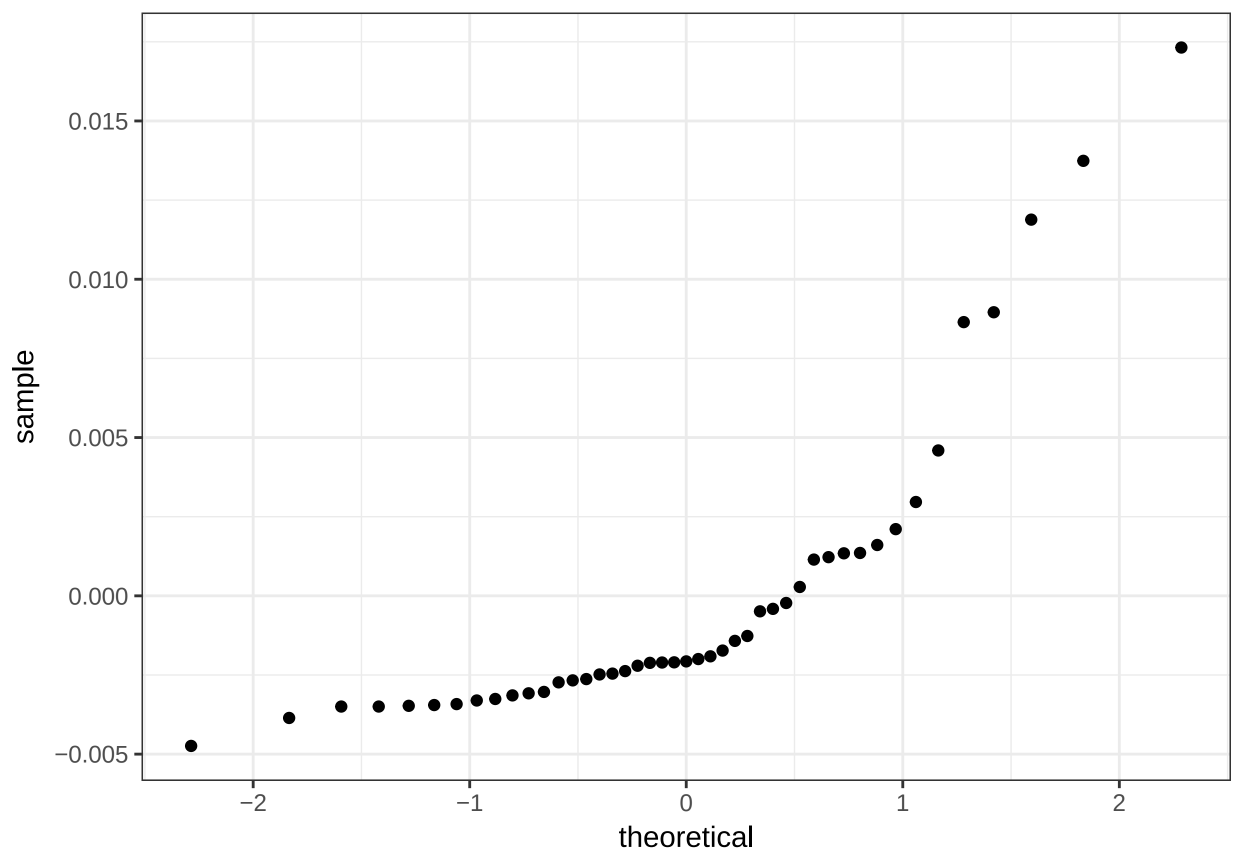
In the first line of code we extract the residuals much the same way as specifying a variable in a data frame, using the "$". The second approach uses the residuals function (see section 11.3 of the R-introduction).

It's often a good idea to use plots in determining normality of a random variable.

```r
# nclass specifies the number of bins to use in the histogram
hist(u, main = "Histogram of Residuals", nclass = 15)
```



```
ggplot() +
  stat_qq(aes(sample =(u-mean(u))/var(u))) +
  theme_bw()
```

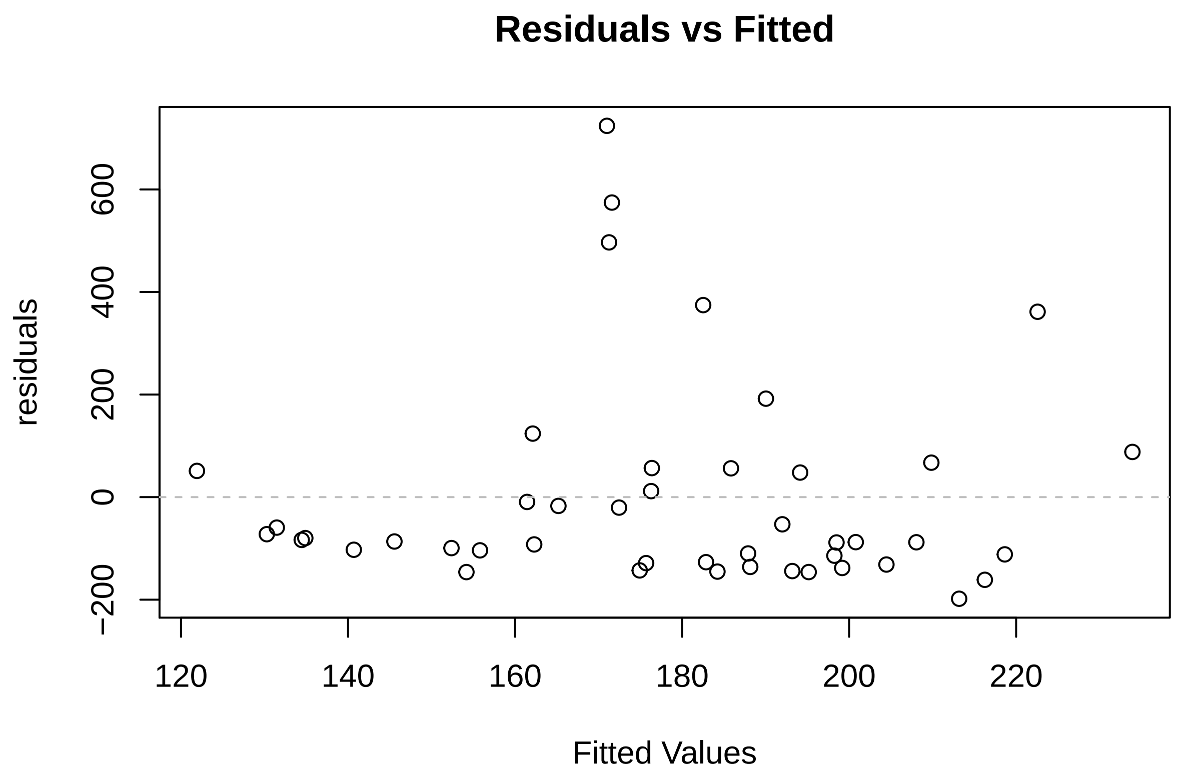


```
# if it were normal, we should have that the sample quantiles coincide with the
# theoretical quantiles of the normal distribution (dots should be along a diagonal line).
# This is not the case
```

hist() is a base function in R. For more information on these you should consult section 12 of the R introduction. The second plot is done using the ggplot2 package. This is outside the scope of this course and should you use this type of plot you can feel free to copy my syntax and replace the object with your own.

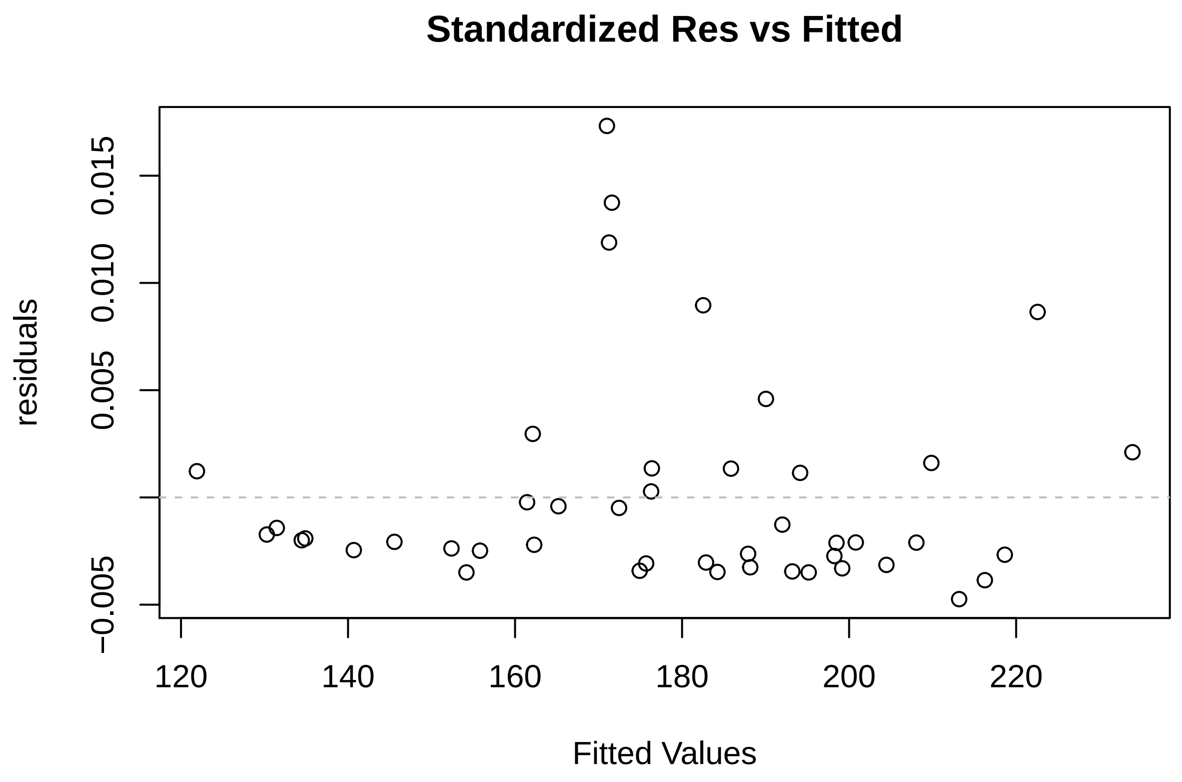
If you want to investigate the residuals and fitted values, it is done as follows

```
fitted <- m1$fitted.values

plot(y = u, x = fitted, main = "Residuals vs Fitted",
     xlab = "Fitted Values", ylab = "residuals")
abline(a = 0, b = 0, lty = 2, col = "gray")
```



Often you see these type of plots with the residuals standardized.

```
fitted <- m1$fitted.values
u_std <- (u - mean(u))/var(u)

plot(y = u_std, x = fitted, main = "Standardized Res vs Fitted",
     xlab = "Fitted Values", ylab = "residuals")
abline(a = 0, b = 0, lty = 2, col = "gray")
```



You can inspect the difference by looking at the values of the y-axis on each plot.

Next we evaluate the model in terms of multicollinearity. This is done using the $VIF()$ function from the fmsb package. The rule of thumb is that VIF < 10 is ok. So let's investigate the *cultivated* variable. To check VIF of *cultivated* you fit a linear model with this variable as dependent of the other independent variables

```
m_vif <- lm(data = data, cultivated ~ farmland)
VIF(m_vif)
```

```
## [1] 1.000005
```

```
# Less than 10, suggesting it's not a problem
```

Lastly we will evaluate whether the residuals are heteroskedastic or not. This you will do using White's test. However since there is no usable function in R that performs this test you will have to use a function written by myself. This is one of the drawbacks of R but also one of its greatest strengths. Some functions and tests are not available in R but the only thing that holds you back from writing them yourselves is your statistical knowledge and programming ability. Below you find the function along with code on how to use them for your model.

```
#### White's test for heteroskedasticity function ####
white_test <- function(dat, names_independent, model){
  # dat - data frame
  # names_independent - concatonate object with all variable names of regressors
  #                     as they appear in the data frame
  # model - lm object
  # how many variables have the user submitted?
  n_vars <- length(names_independent)

  #subset supplied data
  sub_data <- dat[, which(colnames(dat) %in% names_independent)]

   # set up dataframe that will be used
  out_dat <- data.frame(V1 = rep(NA, nrow(dat)))

  # how many columns the final data set should contain
  # number of crossproducts = (n_vars^2-n_vars)/2, see quadratic form
  n_cols <- n_vars*2 + (n_vars^2-n_vars)/2

  ### Squares
  for(i in 1:n_vars) {
    out_dat[, i] <- (sub_data[, i])^2
  }
  ### non squares
  for(i in (n_vars+1):(2*n_vars)){
    out_dat[, i] <- sub_data[, (i-n_vars)]
  }
  ### Crossproducts
  ind_vec <- (2*n_vars+1):n_cols
  ind <- 0
  for(j in 1:(n_vars-1)){
    for(k in (j+1):n_vars){
      # if(k == j) next # squares are already dealt with
      ind <- ind + 1
      out_dat[, ind_vec[ind]] <- sub_data[, j]*sub_data[, k]
    }
  }
  # add new dependent variable as outlined by white's test
  out_dat$u2 <- (model$residuals)^2

  # what we want to return
  result <- summary(lm(data = out_dat, u2 ~ .))
  # returns list with F-statistic and it's corresponding p-value
  return( list("F-statistic" = result$fstatistic[[1]],
               "p-value" = pf(result$fstatistic[1],
                              result$fstatistic[2],
                              result$fstatistic[3],
                              lower.tail = F)[[1]]))
}
```

Note in the first line of code that there are three parameters within *function*(). These you have to specify in order for the function to work. The first, *dat*, is the data frame that you are using. The second, $names_independent$, is a concatonate object consisting of the names of the regressors. Lastly we have *model* which is lm object. Below I use the function on my three models.

```
# m1
white_test(dat = data, names_independent = c("cultivated", "farmland"), model = m1)
```

```
## $`F-statistic`
## [1] 0.3835528
```

```
##
## $`p-value`
## [1] 0.8570043
```

```
# m2
white_test(dat = data, names_independent = c("cultivated", "farmland", "arable"), model = m2)
```

```
## $`F-statistic`
## [1] 1.174756
##
## $`p-value`
## [1] 0.3408406
```

```
# m3
white_test(dat = data, names_independent = c("cultivated", "farmland", "arable", "labors"), model = m3)
```

```
## $`F-statistic`
## [1] 0.5102587
##
## $`p-value`
## [1] 0.9082279
```