# Introduction to R

Prof. Dr. Derya Uysal
Winter 2019/20

LMU Munich
Department of Economics
Email: derya.uysal@econ.lmu.de

# Introduction

- Our aim is to introduce the basics of R
- Remember this is an econometrics course/tutorial, it is NOT an R lecture.
- Most of these slides are based on the following:
    - Rodrigues, B. (2014) "Introduction to programming Econometrics with R" https://www.brodrigues.co/blog/2015-01-12-introduction-to-programming-econometrics-with-r/
    - Kleiber, C. and Zeileis, A. (2017) "Applied Econometrics with R" https://eeecon.uibk.ac.at/ zeileis/teaching/AER/
    - Heiss, F. (2016) "Using R for Introductory Econometrics" http://www.urfie.net/

Other references

- Hanck, Arnold, Gerber, Schmelzer (2018). Introduction to Econometrics with R. GitHub/bookdown. https://www.econometrics-with-r.org/
- W. N. Venables, D. M. Smith and the R Core Team (2019) An Introdcution to R https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf
- Tutorials in RStudio https://education.rstudio.com/learn/
- Cheat sheets in RStudio https://rstudio.com/resources/cheatsheets/
- Stackoverflow is a good resource for specific questions and answers. https://stackoverflow.com/questions/tagged/r
- Rapidly growing list of books on R or on statistics using R.

## Why use R?

- Runs on any modern operating system
- Very rapid and active development. There are yearly releases, and minor releases in between to fix bugs
- Very nice graphs (especially with `ggplot2`, a package that makes beautiful graphs)
- Huge user community, getting help is easy
- R is free software

## Installation

- We will install two things: R itself, and Rstudio, an IDE for R.
- An IDE (Integrated Development Environment) is an interface that allows the user to program more efficiently.
- Go to the following url http://cran.r-project.org/bin/windows/base/ and download the latest version of R. Since you're probably using a modern computer, install the 64-bit version.
- Once the installation is complete, you can download Rstudio here: http://www.rstudio.com/ide/ download/desktop.

# File management

Working directory:

- query with `getwd()`
- change with `setwd()`
- if available, `.RData` and/or `.Rhistory` are loaded upon startup,
- `dir()` lists available files

More generally:

- directories can be listed with `dir()`
- saved workspaces can be loaded using `load()`,
- R objects can be saved by `save()`.

## Packages

Packages are a very neat way to extend R's functionality

- packages can contain R code, source code (e.g., C, Fortran), data, manual pages, further documentation, examples, demos, . . .
- package can depend on other packages (that need to be available for using the package),
- "base" packages: contained in the R sources,
- "recommended" packages: included in every binary distribution,
- "contributed" packages: available from the CRAN servers (currently more than 10,000) at https://CRAN.R-project.org/web/packages/.

# Packages

Installing and loading packages:

- if connected to the internet, simply type install.packages("nameofthepackage") for installing a package,
- packages are installed in libraries (= collections of packages),
- library paths can be specified (see `?library`),
- packages are loaded by the command `library()`, e.g., `library("AER")`,
- `library()` lists all currently installed packages.

## R Basics: Vocabulary

- Source code: the source code is the file in which you write the instructions. In R, these files have a `.R` extension.
- Command prompt: In Rstudio, you have a pane where you write your script, and another pane that is the command prompt.
- Object: An object is a location in memory with a value and an identifier. An object can be a variable, a data structure (such as a matrix) or a function. An object has generally a type or a class.
- Class: determines the nature of an object. For example, if A is a matrix, then A would be of class matrix.
- Identifier: the name of an object. In the example above, A is the identifier.
- Comments: in your script file, you can also add comments. Comments begin with a # symbol and are not executed by R

# R Basics: Data types and objects

- Integers: Integers are numbers that can be written without a fractional or decimal component

```
> p <- as.integer(3)
> class(p)
[1] "integer"
```

- Floating point numbers: Floating point numbers are representations of real numbers.

```
> p <- 3
> class(p)
[1] "numeric"
```

- Strings: Strings are chain of characters:

```
> a <- "this is a string"
> class(a)
[1] "character"
```

## R Basics: Vectors and matrices

- In most programming languages a vector is nothing more than a list of things, i.e. numbers (either integers or floats), strings, or even other vectors.
- The c command: A very important command that allows you to build a vector:

```
> a <- c(1,2,3,4,5)
> print(a)
[1] 1 2 3 4 5
> class(a)
[1] "numeric"
```

- Note that c doesn't build a vector in the mathematical sense, but rather a list with numbers.

```
> dim(a)
NULL
```

- The `cbind` command and `rbind` command

```
> a <- cbind(1,2,3,4,5)
> print(a)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
> class(a)
[1] "matrix"
> dim(a)
[1] 1 5
```

- Let's create a bigger matrix:

```
>  b <- cbind(6,7,8,9,10)
```

- Now let's put vector `a` and `b` into a matrix called `c` using `rbind`

```
> c <- rbind(a,b)
> print(c)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

- You can create a matrix of dimension (5,5) filled with 0's with the following command:

  ```
  > A <- matrix(0, nrow = 5, ncol = 5)
  ```

- If you want to create the following matrix:

  $$B = \begin{pmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{pmatrix}$$

  you would do it like this:

  ```
  >  B <- matrix(c(2, 4, 3, 1, 5, 7), nrow = 2, byrow = TRUE)
  ```

  The option `byrow = TRUE` means that the rows of the matrix will be filled first

# R Basics: Matrix class

- Access elements of a matrix or vector
- Access the element at the 2nd row, 3rd column of A

```
> A[2, 3]
[1] 0
```

- We can assign a new value to this element

```
> A[2, 3] <- 7
> print(A)
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    7    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0
[5,]    0    0    0    0    0
```

- This class is the result of logical comparisons, for example, if you type:

```
> 4 > 3
[1] TRUE
```

- If we save this in a variable `l` and check `l`'s class::

```
> l <- 4 > 3
> class(l)
[1] "logical"
```

  R returns "logical".[1]

- A logical variable can only have two values, either `TRUE` or `FALSE`.

[1] In other programming languages, logicals are often called `bools`.

# R Basics: Logical Operators

- Logical operators: $<$, $<=$, $>$, $>=$, $==$ (for exact equality) and $!=$ (for "not equal").
- If `expr1` and `expr2` are logical expressions,
- `expr1 & expr2` is their intersection (logical "and"),
- `expr1 | expr2` is their union (logical "or"), and
- `!expr1` is the negation of `expr1`?.

```
>  x <- c(1.8, 3.14, 4, 88.169, 13)
>  x > 3 & x <= 4
[1] FALSE  TRUE  TRUE FALSE FALSE
```

- Assess which elements are TRUE:

```
> which(x > 3 & x <= 4)
[1] 2 3
```

- Specialized functions `which.min()` and `which.max()` for computing the position of the minimum and the maximum.

**Figure 1:** Ada Lovelace, an English mathematician, discovered the notion of looping in 1843 and is often credited as being the first computer programmer in history.

# R Basics: If-Else

- If $a > b$ then $c$ should be equal to 20, else $c$ should be equal to 10.

```
> a <- 4
> b <- 5
> if (a > b) {
+     c <- 20
+ } else {
+         c <- 10
+         }
> print(c)
[1] 10
```

- It is also possible to add multiple statements. For example:

```
> if (10 %% 3 == 0) {
+ print("10 is divisible by 3")
+ } else if (10 %% 2 == 0) {
+ print("10 is divisible by 2")
+ }
[1] "10 is divisible by 2"
>
```

# R Basics: Looping

- For loops

```
> result = 0
> for (i in 1:100){
+ result <- result + i
+ }
> print(result)
[1] 5050
```

- While loops

```
>  result = 0
>  i = 1
>  while (i<=100){
+ result <- result + i
+ i <- i + 1
+ }
>  print(result)
[1] 5050
```

# R Basics: Functions

Some examples of preprogrammed functions available in R

- Numeric functions

  `abs(x)`: returns the absolute value of x
  `sqrt(x)`: returns the square root of x
  `round(x, digits = n)`: rounds a number to the $n^{th}$ place
  `exp(x)`: returns the exponential of x
  `log(x)`: returns the natural log of x
  `log10(x)`: returns the common log of x
  `cos(x), sin(x), tan(x)`: trigonometric functions
  `factorial(x)`: returns the factorial of x
  `sum(x)`: For a vector x, returns the sum of its elements
  `min(x)`: For a vector x, returns the smallest of its elements
  `max(x)`: For a vector x, returns the largest of its elements

- Statistical and probability functions

    `dnorm(x)`: returns the normal density function

    `pnorm(q)`: returns the cumulative normal probability for quantile $q$

    `qnorm(p)`: returns the quantile at percentile $p$

    `rnorm(n, mean = 0, sd = 1)`: returns $n$ random numbers from the standard normal distribution

    `mean(x)`: For a vector x, returns the mean

    `sd(x)`: For a vector x, returns its standard deviation

    `cor(x)`: gives the linear correlation coefficient

    `median(x)`: For a vector x, returns its median

    `table(x)`: For a vector x, makes a table of all values of x with frequencies

    `summary(x)`: For a vector x, returns a number of summary statistics for x

# R Basics: Functions

- Matrix manipulation

  `A*B`: returns the element-wise multiplication of A and B

  `A %*% B`: returns the cumulative normal probability for quantile $q$

  `A %x% B` or `kronecker(A, B)`: returns the Kronecker product of A and B

  `t(A)`: returns the transpose of A

  `diag(A)`: returns the diagonal of A

  `eigen(A)`: returns the eigenvalues and eigenvectors of A

  `chol(A)`: Choleski factorization of A

- Other useful commands

  `rep(a, n)`: repeat a n times

  `seq(a,b,k)`: rcreates a sequence of numbers from a to b, by step k

  `cbind(n1, n2, n3,...)` creates a vector of numbers

  `c(n1, n2, n3, ...)`: similar to cbind, but the resulting object doesn't have a dimension

  `dim(a)`: check dimension of a

  `length(a)`: returns length of a vector

  `ls()`: lists memory contents (doesn't take an argument)

  `sort(x)`: sort the values of vector x

  `?keyword`: looks up help for keyword. keyword must be an existing command

  `??keyword`: looks up help for keyword, even if the user is not sure the command exists

- Suppose you want to create the following function: $f(x) = \frac{1}{\sqrt{x}}$. This is the syntax you would use:

```
> MyFunction <- function(x){
+ # This function takes one argument, x,
+ # and return the inverse of its square root.
+ return(1/sqrt(x))
+ }
>  MyFunction(4)
[1] 0.5
```

Creation from scratch

- Data frames: Basic data structure in R. (In other programs such structures are often called data matrix or data set.)
- Typically: An array consisting of a list of vectors and/or factors of identical length, i.e., a rectangular format where columns correspond to variables and rows to observations.
- Example: Artificial data with variables named "one", "two", "three".

  ```
  > mydata <- data.frame(one = 1:10, two = 11:20, three = 21:30)
  ```

  Alternatively:

  ```
  >  mydata <- as.data.frame(matrix(1:30, ncol = 3))
  >  names(mydata) <- c("one", "two", "three")
  ```

- Technically: This data frame is internally represented as a list of vectors (not a matrix).

Subset selection

- Select columns: Subsets of variables can be selected via [ or $ (for a single variable).

```
> mydata$two
 [1] 11 12 13 14 15 16 17 18 19 20
> mydata[, "two"]
 [1] 11 12 13 14 15 16 17 18 19 20
> mydata[, 2]
 [1] 11 12 13 14 15 16 17 18 19 20
>
```

In all cases: The data frame attributes are dropped (by default).

Subset selection

- Select rows: Subsets of observations (and variables) can be selected again via [ or (more conveniently) via `subset()`.

```
> subset(mydata, two <= 16, select = -two)
  one three
1   1    21
2   2    22
3   3    23
4   4    24
5   5    25
6   6    26
```

Import and export

- Export as plain text: `write.table()`

  ```
  > write.table(mydata, file = "mydata.txt", col.names = TRUE)
  ```

  This creates a text file mydata.txt in the current working directory.
- To read again, use:

  ```
  > newdata <- read.table("mydata.txt", header = TRUE)
  ```

Details:

- `read.table()` returns a "data.frame" object
- By setting `col.names = TRUE`, `mydata.txt` contains variable names in the first row. Hence, it should be read with `header = TRUE`.
- `write.table()` allows specification of: separation symbol, decimal separator, quotes, and many more. Thus, it can create tab- or comma-separated values etc.

Import and export

- CSV: Comma-separated values
- `read.csv()` and `write.csv()` are available.
- CSV is useful format for exchanging data between R and Microsoft Excel.
- More elementary: scan() is useful for reading more complex structures.
- See the manual pages and the "R Data Import/Export" manual for further details.

Reading and writing foreign binary formats

- Package foreign: R can also read and write a number of proprietary binary formats, including S-PLUS, SPSS, SAS, Stata, Minitab, Systat, and dBase files.
- Example: Stata files

  Export

  ```
  > library("foreign")
  > write.dta(mydata, file = "mydata.dta")
  ```

  Import

  ```
  > mydata <- read.dta("mydata.dta")
  ```

- CPS1985 from Berndt (1991) (comes with the package "AER")

```
> library(AER)
> data("CPS1985")
> str(CPS1985)
'data.frame':       534 obs. of  11 variables:
 $ wage      : num  5.1 4.95 6.67 4 7.5 ...
 $ education : num  8 9 12 12 12 13 10 12 16 12 ...
 $ experience: num  21 42 1 4 17 9 27 9 11 9 ...
 $ age       : num  35 57 19 22 35 28 43 27 33 27 ...
 $ ethnicity : Factor w/ 3 levels "cauc","hispanic",..: 2 1 1 1 1 1
 $ region    : Factor w/ 2 levels "south","other": 2 2 2 2 2 2 1 2
 $ gender    : Factor w/ 2 levels "male","female": 2 2 1 1 1 1 1 1
 $ occupation: Factor w/ 6 levels "worker","technical",..: 1 1 1 1
 $ sector    : Factor w/ 3 levels "manufacturing",..: 1 1 1 3 3 3 3
 $ union     : Factor w/ 2 levels "no","yes": 1 1 1 1 1 2 1 1 1 1 .
 $ married   : Factor w/ 2 levels "no","yes": 2 2 1 1 2 1 1 1 2 1 .
```

- Overview: Summary by variable.

```
> summary(CPS1985)
      wage          education       experience         age       
 Min.   : 1.000   Min.   : 2.00   Min.   : 0.00   Min.   :18.00  
 1st Qu.: 5.250   1st Qu.:12.00   1st Qu.: 8.00   1st Qu.:28.00  
 Median : 7.780   Median :12.00   Median :15.00   Median :35.00  
 Mean   : 9.024   Mean   :13.02   Mean   :17.82   Mean   :36.83  
 3rd Qu.:11.250   3rd Qu.:15.00   3rd Qu.:26.00   3rd Qu.:44.00  
 Max.   :44.500   Max.   :18.00   Max.   :55.00   Max.   :64.00  
    ethnicity     region        gender          occupation        
 cauc     :440   south:156   male  :289   worker    :156   manufactu
 hispanic: 27   other:378   female:245   technical :105   construct
 other    : 67                           services  : 83   other
                                          office    : 97  
                                          sales     : 38  
                                          management: 55  
 union      married  
 no :438   no :184  
 yes: 96   yes:350  
```

For simplifying input and output:

```
> levels(CPS1985$occupation)[c(2, 6)] <- c("techn", "mgmt")
> attach(CPS1985)
```

In the following:

- Exploratory analysis of a single numerical/categorical variable.
- Exploratory analysis of pairs of variables.

One numerical variable

- Distribution of wages:

```
> summary(wage)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.000   5.250   7.780   9.024  11.250  44.500 
```

- Standalone functions: `mean()`, `median()`, `min()`, `max()`, `fivenum()`.

```
> mean(wage)
[1] 9.024064
```

- Arbitrary quantiles: `quantile()`.
- Measures of spread: variance and standard deviation.

```
> var(wage)
[1] 26.41032
> sd(wage)
[1] 5.139097
```

- Conditional summary statistics

```
> mean(wage[gender == "male"])
[1] 9.994913
```

One numerical variable

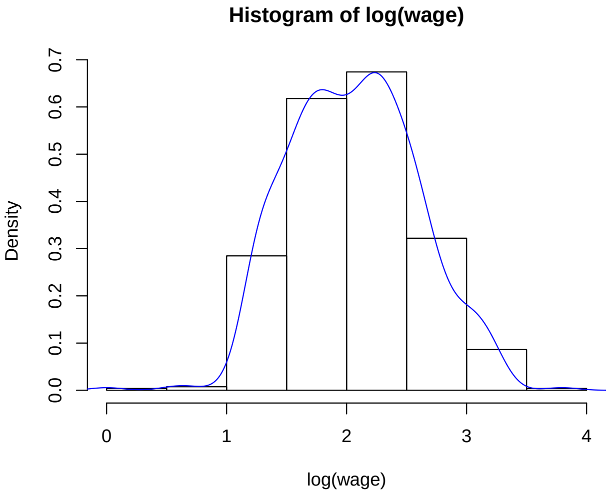
Graphical summary: Density visualizations (via histograms or kernel smoothing) and boxplots.

```
> hist(log(wage), freq = FALSE)
> lines(density(log(wage)), col = 4)
```

Details:

- Density of logarithm of wage (i.e., area under curve equals 1).
- Default: absolute frequencies, changed to density via `freq = FALSE`.
- Further fine tuning possible via selection of `breaks`.
- Added kernel density estimate.

One numerical variable



One categorical variable

- Appropriate summary chosen automatically for "factor" variables.

```
> summary(occupation)
  worker    techn services   office    sales     mgmt
     156      105       83       97       38       55
```

- Alternatively: Use `table()` and also compute relative frequencies.

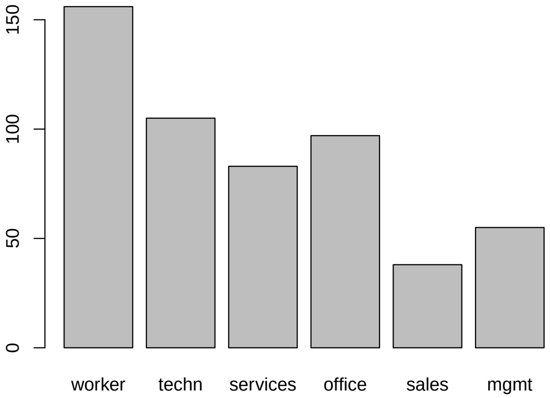
```
> tab <- table(occupation)
> prop.table(tab)
occupation
    worker      techn   services     office      sales       mgmt
0.29213483 0.19662921 0.15543071 0.18164794 0.07116105 0.10299625
```

- Visualization: `barplot()`. If majorities are to be brought out, `pie()` charts might be useful. Both expect tabulated frequencies as input.

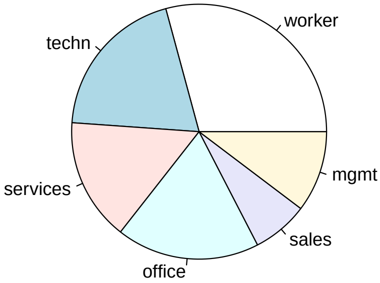
```
> barplot(tab)
> pie(tab)
```

- `plot(occupation)` is equivalent to `barplot(table(occupation))`.

One categorical variable



One categorical variable



Two categorical variables

- Relationship between two categorical variables:
  - Numerical summary: Contingency table(s) via `xtabs()` or `table()`.
  - Use `table(gender, occupation)` or

```
> xtabs(~ gender + occupation, data = CPS1985)
```

```
        occupation
gender   worker techn services office sales mgmt
  male      126    53       34     21    21   34
  female     30    52       49     76    17   21
```

  - Graphical summary: Mosaic plot, a generalization of stacked barplots. The following variant is also called "spine plot":

```
> plot(gender~occupation, data = CPS1985)
```

Bar heights correspond to the conditional distribution of gender given occupation. Bar widths visualize the marginal distribution of occupation.

Two categorical variables

Two numerical variables

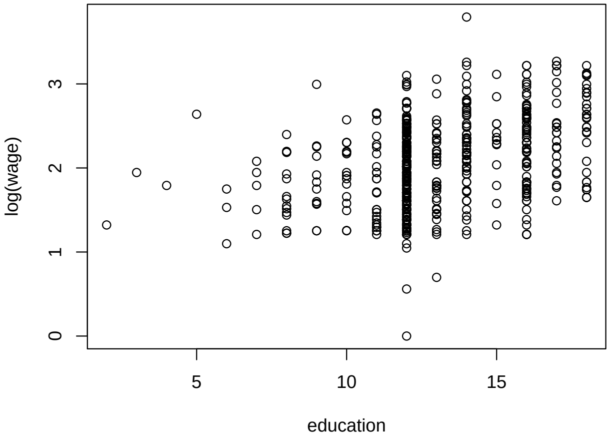
- Numerical summary: Correlation coefficient(s) via cor(). Default is the standard Pearson correlation coefficient.

```
> cor(log(wage), education)
[1] 0.3803983
```

- Graphical summary: Scatterplot.

```
> plot(log(wage) ~ education)
```

Two numerical variables



One numerical and one categorical variable

- Numerical summary: Grouped numerical summaries (for the numerical variable given the categorical variable)
- `tapply()` applies functions grouped by a (list of) categorical variable(s).
- Mean wages conditional on gender are available using:

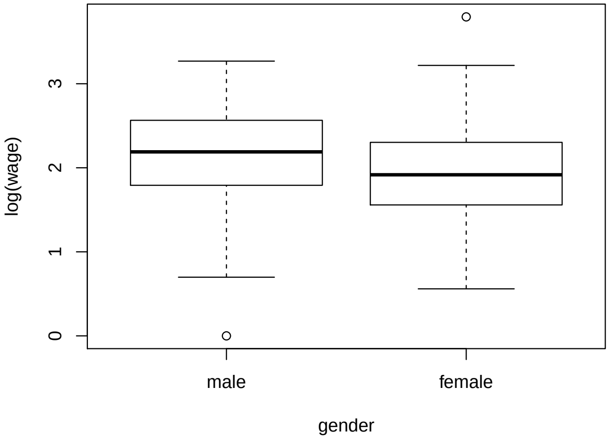
```
> tapply(log(wage), gender, mean)
    male   female
2.165286 1.934037
```

- Other measures: Replace mean by other function, e.g., summary
- Graphical summary: Parallel boxplots

```
> plot(log(wage) ~ gender)
```

The commands `plot(y ~x)` and `boxplot(y ~x)` both yield the same parallel boxplot if x is a "factor".

One numerical and one categorical variable



One numerical and one categorical variable
Boxplots:

- Coarse graphical summary of an empirical distribution.
- Box indicates "hinges" (approximately the lower and upper quartiles) and the median.
- "Whiskers" indicate the largest and smallest observations falling within a distance of 1.5 times the box size from the nearest hinge.
- Observations outside this range are outliers (in an approximately normal sample).

- Let us suppose we want to estimate the parameters of the following model:

  $$wage_i = \beta_0 + \beta_1 * ethnicity_i + \beta_2 * education_i + \beta_3 * gender_i + \varepsilon_i \quad i = 1, \dots, n$$

- Remember the OLS estimator:

  $$\hat{\beta} = \left(X'X\right)^{-1} X'y$$

  where

  $$X = \begin{pmatrix} 1 & eth.._1 & education_1 & gender_1 \\ 1 & eth.._2 & education_2 & gender_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & eth.._n & education_n & gender_n \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} \quad y = \begin{pmatrix} wage_1 \\ wage_2 \\ \vdots \\ wage_n \end{pmatrix}$$

- Create the matrix $X$

```
> X <- cbind(1, ethnicity, education, gender)
> dim(X)
[1] 534   4
> class(X)
[1] "matrix"
```

- Define the transpose of $X$

```
> tX <- t(X)
```

- Compute $\hat{\beta}$

```
> beta_hat <- solve(tX %*% X) %*% tX %*% wage
```

- What about standard errors?

```
> res_hat <- wage- X%*%beta_hat
> sigma_hat <- (sum(res_hat^2)/(nrow(X)-ncol(X)))
> invxx <- solve(tX %*% X)
> Vbeta_hat <- sigma_hat*invxx
> se_beta_hat <- as.matrix(sqrt(diag(Vbeta_hat)))
> cbind(beta_hat,se_beta_hat)
                [,1]       [,2]
           3.1565646 1.27557173
ethnicity -0.4850776 0.29648014
education  0.7391806 0.07705734
gender    -2.1417333 0.40234273
```

- Another method you can use to obtain the same result is to use the command `lm()`

```
> ethnN <- as.numeric(ethnicity)
> genderN <- as.numeric(gender)
> lm(wage ~ ethnN + education + genderN)
Call:
lm(formula = wage ~ ethnN + education + genderN)

Coefficients:
(Intercept)        ethnN    education      genderN
     3.1566      -0.4851       0.7392      -2.1417
```

- If you want more details you can use `summary()` with `lm()`:

```
> model <-lm(wage ~ ethnN + education + genderN)
> summary(model)
Call:
lm(formula = wage ~ ethnN + education + genderN)

Residuals:
   Min     1Q Median     3Q    Max
-9.007 -3.054 -0.602  2.230 35.763

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.15656    1.27557   2.475   0.0136 *
ethnN       -0.48508    0.29648  -1.636   0.1024
education    0.73918    0.07706   9.593  < 2e-16 ***
genderN     -2.14173    0.40234  -5.323 1.51e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```