Introduction to R Programming

Ozan Bakış[1]

[1]Bahcesehir University, Department of Economics and BETAM

Outline

What is R? I

- R home page: `https://www.r-project.org`
- R is a programming language
- R is an environment for statistical computing and graphics
- R is free and open source (since 1995)
- R has lots of packages (more than 20k on CRAN, Bioconductor and Github)
- R is used in industry: Google, IBM, HP, Microsoft, Oracle etc. See `https://www.r-consortium.org/members`
- R plays well with other programming languages

What is RStudio? I

- RStudio is an integrated development environment (IDE) for R
- There are many IDEs (and graphical user interfaces (GUIs)) for R such as Rcmdr, Emacs+ESS, Revolution-R, Tinn-R, Eclipse, JGR, ...
- RStudio is also a company :(. It is the company providing Rstudio IDE
- RStudio is available in two versions: open source and commercial
- RStudio has good community and commercial support

R resources I

- Official (and difficult): `http://cran.r-project.org/manuals.html`
- Google ”learning r”
- Contributed documentation (customized): `http://cran.r-project.org/other-docs.html`
- Books: `http://www.r-project.org/doc/bib/R-books.html`
- Stackoverflow
- And a large number of mailing lists `https://www.r-project.org/mail.html`

Before getting started I

Getting help: Google it! But also:

```
help("solve")
?"solve"
```

Installing packages:

- to find a package: use http://cran.r-project.org/web/views/ and https://cran.r-project.org/web/packages/
- to install/remove a package:

```
install.packages("dplyr")
remove.packages("dplyr")
#install.packages(c("dplyr","tidyr")) # multiple packages
#remove.packages(c("dplyr","tidyr"))
```

Before getting started II

Getting and setting working directory:

```r
getwd()
setwd("c:\\my work\\R") # windows
setwd("path/to/my work/R") # windows/linux/macOS
```

In RStudio:
Working directory: Session $\longrightarrow$ Set Working Directory $\longrightarrow$ Choose directory

Before getting started III

R Markdown vs R Script:

- R Script is for live code.
- R Markdown is for live documents (HTML, PDF, WORD). For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

Outline

Working with R I

Standard arithmetic operators: `+`, `-`, `*`, `/`, and `^` :

```
(2 + 3*5 - 3^2 )/5
```

```
## [1] 1.6
```

```
2^4
```

```
## [1] 16
```

Use `Ctrl-Enter` to run selected lines or current line
Mathematical functions: R has `log()`, `exp()`, `sqrt()`, `log()`, `abs()`, `min()`, `max()`, `sin()`, `cos()`, `tan()`, `sign()`, ...

```
# log() # by default base = e
log(10, base=2) +  sin(pi/4)
```

```
## [1] 4.03
```

Calling an R script: If R commands are stored in a file, say `do.R` in `c:/my work/R`, the command to use is

```
source("c:/my work/R/do.R") # from anywhere
source("do.R") # If I am already in "c:/my work/R"
```

Assignments I

Assignment operators:

- `<-` and `=` are both OK. But the former is somehow more popular!

```
x <- 5
y = 6
```

- An object's name can not begin with a number. Names are case sensitive.
- The following names are used by R; they shoud not be used as object name: **Inf, NA, NaN, NULL, TRUE, FALSE, break, else, for, function, if, in, next, repeat, return, while**.
- The following ones can be used but it is not recommended **c, q, t, C, D, I, diff, length, mean, pi, range, var**.

Saving and loading objects I

- Basic save and load process:

```
# history() # default is max.show=25
# history(max.show=Inf) # all history
# savehistory(file="myRsession.R") # default is ".Rhistory"
### save some variables
x=5; y=10
save(x, y, file = "xy.RData")
### save all
#save(list=ls(all=TRUE), file = "myRsession.RData")
save.image(file = "myRsession.RData") #  default is ".RData"
load("myRsession.RData") # for loading back
```

- One problem with **`save()`** is it saves the objects and their names together. When **`load()`** loads a file saved by **`save()`** this may overwrite objects in memory already. Use **`saveRDS()`** and **`readRDS()`** for avoiding this danger.
- To see the current objects the command is

Saving and loading objects II

```r
objects()
ls()
```

- To remove objects, say **`x,y,z, foo`** and **`bar`**, from the workspace we use the rm() command

```r
rm(x, y)
```

Data structures I

- 2 factors determine the type of the data structures in R
- dimension (1d, 2d or 3d+) and content (homogenous or heterogenous)

```
      Homogeneous     Heterogeneous
      ------------    ------------
1d    Vector          List
2d    Matrix          Data frame
nd    Array
```

Vectors I

- Generation of vectors: Using combine function, `c()`.

```
x <- c(1.8, 3.14, 4)
```

- Generating vectors with a given pattern:

```
z=1:4
seq(from = 0, to = 0.7, by = 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6
```

```
seq(0, 0.7, 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6
```

```
# seq(from = 0, to = 1, length.out = 6)
# seq(0, 1, 6) ## !!!
s2 = c("a","b","c")
(s3 <- rep(s2, times=2))
```

```
## [1] "a" "b" "c" "a" "b" "c"
```

```
(s4 <- rep(s2, each=2))
```

```
## [1] "a" "a" "b" "b" "c" "c"
```

Vectors II

```
# (s5 <- 3:10)
# (s6=seq_along(s5))
```

- Generating random vectors:

```
set.seed(127) # To make below reproducible
v1 = rnorm(n=5,mean=3, sd=1) # default m=0,sd=1
v1
```

```
## [1] 2.43 2.19 2.51 3.00 3.82
```

```
#v2 = runif(n=5,min=1,max=4) # default min=0,max=1
## random numbers from a given set
sample(1:10,size=10,replace=FALSE)
```

```
##  [1]  6  5  7 10  8  2  1  3  4  9
```

```
sample(1:10,size=10,replace=TRUE)
```

```
##  [1]  5  7  5  3  5  6  5  8 10  9
```

- Basic vector operations:

```
(v1 = 1:4)
```

```
## [1] 1 2 3 4
```

Vectors III

```
(v2 = 4:1)

## [1] 4 3 2 1

v1+v2

## [1] 5 5 5 5

v1*v2

## [1] 4 6 6 4

sum(v1*v2)

## [1] 20

v1 %*% v2# dot product: sum_i^n (v1_i*v2_i)

##      [,1]
## [1,]   20

(v3 <- 1:2)

## [1] 1 2

v1+v3 ##Attention !!!

## [1] 2 4 4 6
```

Missing values:

Vectors IV

```
vv = c(2,NA,3,4)
vv
```

```
## [1]  2 NA  3  4
```

```
mean(vv)
```

```
## [1] NA
```

```
mean(vv, na.rm=TRUE)
```

```
## [1] 3
```

```
is.na(vv) #returns o logical vector
```

```
## [1] FALSE  TRUE FALSE FALSE
```

```
!is.na(vv) #returns o logical vector
```

```
## [1]  TRUE FALSE  TRUE  TRUE
```

NULL values:

```
x = c(2,NULL,3,4)
x
```

```
## [1] 2 3 4
```

```
mean(x)
```

```
## [1] 3
```

Vectors V

```r
is.na(x) #returns o logical vector
## [1] FALSE FALSE FALSE
is.null(x) #returns TRUE or FALSE
## [1] FALSE
```

Matrices I

- Matrices have two dimensions, rows and columns. A 2×3 matrix containing the elements `1:6`, by column, is generated via

```
A <- matrix(1:6, nrow = 2) # tray also: matrix(1:6, ncol = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
A2 <- matrix(1:6, nrow = 2, byrow=T)
A2
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Matrices II

- Solving a system of linear equations: Let's say, we want to solve the following linear system

$$\begin{aligned} 5x + 7y &= 1 \\ 4x + 3y + 2z &= 2 \\ 6x - 2y - z &= 3 \end{aligned}$$

```
A = matrix(c(5,7,0,4,3,2,6,-2,-1),nrow=3, byrow=T)
A
```

```
##      [,1] [,2] [,3]
## [1,]    5    7    0
## [2,]    4    3    2
## [3,]    6   -2   -1
```

```
b=1:3
x=solve(A,b)# solution of A*x=b
x
```

Matrices III

```
## [1]  0.487 -0.205  0.333
```

```r
Ainv = solve(A) # A^(-1) = solve(A)
Ainv %*% b # A^(-1)*b should be equal to x
```

```
##        [,1]
## [1,]  0.487
## [2,] -0.205
## [3,]  0.333
```

Indexing - subsetting I

Vectors:

- Extract elements by their index.
- Exclude elements with negative index.

```
x
```

```
## [1]  0.487 -0.205  0.333
```

```
x[c(1, 4)]
```

```
## [1] 0.487    NA
```

```
x[-c(1, 4)]
```

```
## [1] -0.205  0.333
```

- Conditional subsetting: subsetting by a logical vector

```
set.seed(2762)
x2 = sample(1:100, size = 5)
x2
```

```
## [1] 74 47 35 38 27
```

```
x2 > 40
```

Indexing - subsetting II

```
## [1]  TRUE  TRUE FALSE FALSE FALSE
x2[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
## [1] 74 38 27
x2[x2 > 40] # same as above
## [1] 74 47
```

`TRUE` selects the element with the same index, while `FALSE` does not.

Matrices are vectors with an additional dimension attribute enabling row/column-type indexing

- `A[i,j]` extracts element a_{ij} of matrix A.
- `A[i,]` extracts ith row.
- `A[,j]` extracts jth column.

Indexing - subsetting III

- Results of these operations are vectors, i.e., dimension attribute is dropped (by default).
- **`A[i, j, drop = FALSE]`** avoids dropping and returns a matrix.

```
A[1:2, c(1, 3)]
```

```
##      [,1] [,2]
## [1,]    5    0
## [2,]    4    2
```

```
A[-(1:2), c(1, 3)]
```

```
## [1]  6 -1
```

Data frame I

- A data frame is a 2-dimensional list with the following properties:
  1. The components are vectors of the same length but they can have different data types, i.e. numeric, character, or logical.
  2. Each column has a title by which the whole vector may be addressed.
  3. Numeric vectors, logicals and factors are included as is, and character vectors are coerced to be factors.
- To create a data frame and entering some values manually for variables:

```
dz <- data.frame(v1= 1:4,v2 = c("B","C","A","D"),
     v3=c(33,43,61,91))
dz
```

```
##   v1 v2 v3
## 1  1  B 33
## 2  2  C 43
## 3  3  A 61
## 4  4  D 91
```

Data frame II

```
dz[order(dz$v2, dz$v3), ]
```

```
##   v1 v2 v3
## 3  3  A 61
## 1  1  B 33
## 2  2  C 43
## 4  4  D 91
```

- In order delete a column permanently:

```
dz$log_v3 = log(dz$v3) # creates new var
dz$v3 <- NULL # deletes v3
dz
```

```
##   v1 v2 log_v3
## 1  1  B   3.50
## 2  2  C   3.76
## 3  3  A   4.11
## 4  4  D   4.51
```

Outline

Reading-writing an R data file I

R has its own data file format, RDS. It is saved using the `.rds` extension by default. `readRDS()` and `saveRDS()` can be used to read and write R data files.

```
#### Saving as RDS file:
saveRDS(dz, "my_data.rds") #content of "dz" saved in "my_data.rds"
dat <- readRDS("my_data.rds") #content of "my_data.rds" loaded as "dat"
```

Both saveRDS and readRDS functions use `gzip` compression by default. This is benefit, as saved files have smalles storage space but when a file is read from a website we need to decompress it before loading it. When loading from local disk this is not needed.

```
dat_url = "https://github.com/obakis/econ_data/raw/master/wage1.rds"
download.file(url = dat_url, destfile = "wage1.rds", mode="wb")
wage1 = readRDS("wage1.rds")

## or in 1 step
wage1 = readRDS(gzcon( url( dat_url )))
```

Reading-writing Excel and CSV files I

```r
library(openxlsx)
#### simple Excel file
f_url = "https://github.com/obakis/econ_data/raw/master/tur_gdp-un-inf.xlsx"
download.file(url = f_url, destfile = "tur_gdp-un-inf.xlsx", mode="wb")
datTUR = read.xlsx("tur_gdp-un-inf.xlsx")
#### complicated Excel file
f_url = "https://github.com/obakis/econ_data/raw/master/illere_gore_ihracat.xlsx"
download.file(url = f_url, destfile = "il_ihracat.xlsx", mode="wb")
dat = read.xlsx("il_ihracat.xlsx",
                cols = 1:16, rows=5:1458) #colNames = TRUE by default

##### CSV: no need for extra package
v1 = read.csv("ver1.csv") # sep = "," and dec = "."
v2 = read.csv2("ver2.csv") # sep = ";" and dec = ","

#### Writing to Excel or CSV
write.xlsx(datTUR,"TUR_data.xlsx")
write.csv(datTUR,"TUR_data.csv") # sep = "," and  dec = "."
write.csv2(datTUR,"TUR_data.csv") # sep = ";" and dec = ","
```

Reading-writing other data files I

- **`read.table()`** for any type of delimited ASCII file (both numeric and character values). We can specify optional arguments for decimals, missing values, headers, separator etc.

- **`foreign`** package can be used to read and write data in 'Minitab', 'SAS', 'SPSS', 'Stata' etc. format.

```
#Ex.
read.dta    #Read Stata binary files
read.spss   #Read an SPSS data file
read.xport  #Read a SAS XPORT Format Library
read.dbf    #Read a DBF File
read.octave #Read Octave Text Data Files
```

Outline

dplyr verbs I

We will use dplyr package for data manipulation and descriptive statistics

- `select()`: select columns
- `filter()`: select rows
- `arrange()`: order or arrange rows
- `mutate()`: create new columns
- `summarise()`: summarize values
- `group_by()`: group observations

```r
f_url = "https://github.com/obakis/econ_data/raw/master/hls2011.rds"
download.file(url = f_url, destfile = "hls2011.rds", mode="wb")
hls = readRDS("hls2011.rds")
library("dplyr")
```

dplyr verbs II

```
hls |>
  select(hwage, educ,female,exper,emp_sect) |>
  head(n=3)
```

```
##   hwage educ female exper emp_sect
## 1  8.75    2      0    33      pub
## 2  2.92    2      1     2     priv
## 3  2.53    5      1    22     priv
```

```
hls |>
  select(hwage, educ,female,exper,emp_sect) |>
  summary()
```

```
##      hwage             educ             female           exper     
##  Min.   : 1.2   Min.   : 0.00   Min.   :0.00   Min.   : 0.0  
##  1st Qu.: 2.9   1st Qu.: 5.00   1st Qu.:0.00   1st Qu.:10.0  
##  Median : 3.9   Median : 8.00   Median :0.00   Median :18.0  
##  Mean   : 6.2   Mean   : 9.26   Mean   :0.22   Mean   :18.9  
##  3rd Qu.: 8.2   3rd Qu.:15.00   3rd Qu.:0.00   3rd Qu.:27.0  
##  Max.   :58.3   Max.   :15.00   Max.   :1.00   Max.   :72.0  
##   emp_sect
```

dplyr verbs III

```
##  other:  9
##  priv :557
##  pub  :196
##
##
##
```

```
hls |>
  select(hwage, educ,female,exper,emp_sect) |>
  mutate(
    exper_gr = cut(exper, breaks = c(0,10,20,30,40,99),
                    right=FALSE, include.lowest = TRUE)
  ) |>
  count(exper_gr)
```

```
##   exper_gr   n
## 1   [0,10) 184
## 2  [10,20) 234
## 3  [20,30) 200
## 4  [30,40) 108
## 5  [40,99]  36
```

dplyr verbs IV

```r
hls2 = hls |>
  select(hwage, educ,female,exper,emp_sect) |>
  mutate(
    exper_gr = cut(exper, breaks = c(0,10,20,30,40,99),
                            right=FALSE, include.lowest = TRUE)
  )
```

Creating frequency tables with dplyr I

```
tab=hls2 |>
  count(exper_gr)
tab
```

```
##   exper_gr   n
## 1   [0,10) 184
## 2  [10,20) 234
## 3  [20,30) 200
## 4  [30,40) 108
## 5  [40,99]  36
```

```
tab |>
  mutate(
    rat=prop.table(n)
    )
```

Creating frequency tables with dplyr II

```
##   exper_gr   n    rat
## 1   [0,10) 184 0.2415
## 2  [10,20) 234 0.3071
## 3  [20,30) 200 0.2625
## 4  [30,40) 108 0.1417
## 5  [40,99]  36 0.0472
```

```
tab2=hls2 |>
  count(exper_gr,female) |>
  arrange(female, desc(n))
tab2
```

```
##    exper_gr female   n
## 1   [10,20)      0 177
## 2   [20,30)      0 169
## 3    [0,10)      0 125
## 4   [30,40)      0  89
## 5   [40,99]      0  34
## 6    [0,10)      1  59
## 7   [10,20)      1  57
## 8   [20,30)      1  31
```

Creating frequency tables with dplyr III

```
## 9   [30,40)      1  19
## 10  [40,99]      1   2
```

Adding proportions with dplyr I

```
tab2 |>
  mutate(
    rat = n/sum(n)
    )
```

```
##    exper_gr female   n     rat
## 1   [10,20)      0 177 0.23228
## 2   [20,30)      0 169 0.22178
## 3    [0,10)      0 125 0.16404
## 4   [30,40)      0  89 0.11680
## 5   [40,99]      0  34 0.04462
## 6    [0,10)      1  59 0.07743
## 7   [10,20)      1  57 0.07480
## 8   [20,30)      1  31 0.04068
## 9   [30,40)      1  19 0.02493
## 10  [40,99]      1   2 0.00262
```

Adding proportions with dplyr II

```
tab2 |>
  group_by(female) |>
  mutate(
    rat = n/sum(n)
    )
```

```
## # A tibble: 10 x 4
## # Groups:   female [2]
##    exper_gr female     n    rat
##    <fct>     <int> <int>  <dbl>
##  1 [10,20)       0   177 0.298
##  2 [20,30)       0   169 0.285
##  3 [0,10)        0   125 0.210
##  4 [30,40)       0    89 0.150
##  5 [40,99]       0    34 0.0572
##  6 [0,10)        1    59 0.351
##  7 [10,20)       1    57 0.339
##  8 [20,30)       1    31 0.185
##  9 [30,40)       1    19 0.113
## 10 [40,99]       1     2 0.0119
```

Creating a summary table with dplyr I

```
hls |>
  filter(female==0) |>
  group_by(educ) |>
  summarise(
    ave_wage = mean(hwage),
    ave_exper = mean(exper),
    min_wage = min(hwage),
    sd_wage = sd(hwage),
    nobs = n()
  )
```

```
## # A tibble: 6 x 6
##    educ ave_wage ave_exper min_wage sd_wage  nobs
##   <int>    <dbl>     <dbl>    <dbl>   <dbl> <int>
## 1     0     3.02      31       1.5     1.07     7
## 2     2     3.56      21.4     1.69    1.85    17
## 3     5     4.02      24.7     1.33    2.26   217
## 4     8     3.69      16.3     1.30    1.82   126
## 5    11     5.26      17.2     1.67    3.03    78
## 6    15    11.6       16.6     1.56    8.72   149
```

Analyzing Turkish exports using dplyr I

```r
library(dplyr)
f_url = "https://github.com/obakis/econ_data/raw/master/tur_x.rds"
download.file(url = f_url, destfile = "tur_x.rds", mode="wb")
dat_x = readRDS("tur_x.rds")


#group_by(): How to group data (to facilitate "split-apply-combine")
#slice_max()/slice_min(): to select the top entries in each group

### least exporting 2 provinces in 2017 in each month
dat_x %>%
  filter (!is.na(export) & year == 2017) %>%
  group_by(year,month) %>%
  slice_min(export, n =2) %>%
  print(n=5)
```

Analyzing Turkish exports using dplyr II

```
## # A tibble: 24 x 4
## # Groups:   year, month [12]
##    year province month    export
##   <dbl>    <dbl> <fct>     <dbl>
## 1  2017       29 January    4.62
## 2  2017       13 January    6.38
## 3  2017       62 February   2.41
## 4  2017       36 February   2.73
## 5  2017       62 March     18.2
## # i 19 more rows
```

```
## most exporting 2 provinces in 2017 in each month
dat_x %>%
  filter (!is.na(export) & year == 2017) %>%
  group_by(year,month) %>%
  mutate(sh_x = 100*export/sum(export)) %>%
  slice_max(sh_x, n =2) %>%
  print(n=4)
```

Analyzing Turkish exports using dplyr III

```
## # A tibble: 24 x 5
## # Groups:   year, month [12]
##    year province month      export  sh_x
##   <dbl>    <dbl> <fct>       <dbl> <dbl>
## 1  2017       34 January  5571150. 49.5
## 2  2017       16 January   745933.  6.63
## 3  2017       34 February 6182411. 51.1
## 4  2017       16 February  881377.  7.29
## # i 20 more rows
```

```
## annual exports by province
dat_x %>%
  group_by(province, year) %>%
  summarise(
    export = sum(export, na.rm=TRUE)
  ) -> dat_x2
```

```
## `summarise()` has grouped output by 'province'. You can override
## using the `.groups` argument.
```

```
head(dat_x2)
```

Analyzing Turkish exports using dplyr IV

```
## # A tibble: 6 x 3
## # Groups:   province [1]
##   province  year   export
##      <dbl> <dbl>    <dbl>
## 1        1  2002  461040.
## 2        1  2003  565281.
## 3        1  2004  816249.
## 4        1  2005  883833.
## 5        1  2006  958987.
## 6        1  2007 1166028.
```

```
## provinces with highest annual exports after 2014 (except ist)
dat_x2 %>%
  filter (province !=34  & year > 2014) %>%
  group_by(year) %>%
  slice_max(export,n=1)
```

Analyzing Turkish exports using dplyr V

```
## # A tibble: 4 x 3
## # Groups:   year [4]
##   province  year    export
##      <dbl> <dbl>     <dbl>
## 1       16  2015  8634502.
## 2       16  2016  9765910.
## 3       16  2017 10535563.
## 4       35  2018   839148.
```

```
## export growth rates by province
dat_x2 %>%
  group_by(province) %>%
  mutate(
    lag_x =  dplyr::lag(export, n = 1, order_by = year),
    gr_x =  100*(export - lag_x)/lag_x
    ) %>%
  arrange(province, desc(year))  %>%
  print(n=4)
```

Analyzing Turkish exports using dplyr VI

```
## # A tibble: 1,371 x 5
## # Groups:   province [81]
##   province  year  export    lag_x   gr_x
##      <dbl> <dbl>   <dbl>    <dbl>  <dbl>
## 1        1  2018  150322. 1822782. -91.8
## 2        1  2017 1822782. 1607018.  13.4
## 3        1  2016 1607018. 1683497.  -4.54
## 4        1  2015 1683497. 1908505. -11.8
## # i 1,367 more rows
```

```
## cumulative change in export shares
dat_x2 %>%
  filter (!is.na(export) & year %in% c(2002,2009,2018)) %>%
  group_by(year) %>%
  mutate(sh_x = 100*export/sum(export)) %>%
  group_by(province) %>%
  mutate(diff_sh = sh_x - first(sh_x, order_by = year)) %>%
  arrange(province) %>%
  print(n=7)
```

Analyzing Turkish exports using dplyr VII

```
## # A tibble: 242 x 5
## # Groups:   province [81]
##   province  year   export   sh_x diff_sh
##      <dbl> <dbl>    <dbl>  <dbl>   <dbl>
## 1        1  2002  461040. 1.28    0
## 2        1  2009 1135887. 1.11   -0.167
## 3        1  2018  150322. 1.21   -0.0722
## 4        2  2002    8097. 0.0225  0
## 5        2  2009   58091. 0.0569  0.0344
## 6        2  2018   12722. 0.102   0.0797
## 7        3  2002   55184. 0.153   0
## # i 235 more rows
```

```
x_perc = dat_x2 %>%
  filter (!is.na(export)) %>%
  group_by(year) %>%
  summarise(p10=quantile(export, probs=0.1),
            p50=quantile(export, probs=0.5),
            p90=quantile(export, probs=0.90)
            )
x_perc %>%
```

Analyzing Turkish exports using dplyr VIII

```
  filter(year > 2015)

## # A tibble: 3 x 4
##    year   p10    p50     p90
##   <dbl> <dbl>  <dbl>   <dbl>
## 1  2016  9219. 167067. 1874348.
## 2  2017 12737. 162682. 2333958.
## 3  2018   966.  15323.  188769.

#### take log of p10 and p50
x_perc %>%
  filter(year > 2015) %>%
  mutate(
    across(c(p10,p90), .fns = log),
    ineq_9010 = p90-p10
    )
```

Analyzing Turkish exports using dplyr IX

```
## # A tibble: 3 x 5
##    year   p10     p50   p90 ineq_9010
##   <dbl> <dbl>   <dbl> <dbl>     <dbl>
## 1  2016  9.13 167067.  14.4      5.31
## 2  2017  9.45 162682.  14.7      5.21
## 3  2018  6.87  15323.  12.1      5.28
```