Introduction to R Programming

Ozan Bakış[1]
[1]Bahcesehir University, Department of Economics and BETAM

Outline

What is R? I

- 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 (+1300 on CRAN, +1600 on Bioconductor, ??? on 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

- http://cran.r-project.org/web/views/
- Official (and difficult): http://cran.r-project.org/manuals.html
- Contributed documentation (customized): http://cran.r-project.org/other-docs.html
- Books: http://www.r-project.org/doc/bib/R-books.html
- Help: Stackoverflow and (lots of) mailing lists https://www.r-project.org/mail.html

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)
```

Working with R II

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

Getting help: Google it! But also:

```
help("solve") # help(solve) works as well
?"solve"      # ?solve works as well
?"for"  # ?for does not work
help.search("binomial distribution") # help(solve) works as well
```

Working with R III

Finding, installing and removing packages:

- Packages : http://cran.r-project.org/web/packages/
- Task views: http://cran.r-project.org/web/views/
- To install/remove a package:

```
install.packages("dplyr", dep = TRUE)
install.packages(c("dplyr","tidyr"), dep = TRUE)
remove.packages("fclust")
remove.packages(c("fclust","pmatch"))
```

Working with R IV

Getting and setting working directory:

```
getwd()
setwd("c:\\my work\\R")
setwd("c:/my work/R")
```

In RStudio we can change:

- Working directory: Session → Set Working Directory → Choose directory
- Default working directory: Tools → Global Options → General → Default working directory

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.

Saving and loading objects II

- To see the current objects the command is

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

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

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

Outline

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)
2 * log(x) + 3
```

```
## [1] 4.18 5.29 5.77
```

```
z = c(10,8,0)
```

- Generating vectors with a given pattern:

```
seq(from = 0, to = 1, by = 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
seq(0, 1, 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
```

Vectors II

```
# 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"
```

```
# (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
```

Vectors III

```
v2 = runif(5,min=1,max=4)#default min=0,max=1
v2
```

```
## [1] 3.52 1.96 3.32 1.60 2.35
```

```
## random numbers from a given set
sample(1:10,size=10,replace=FALSE)
```

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

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

```
##  [1]  4  4  5  5  4  9 10  5  3  3
```

- Basic vector operations:

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

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

```
(v2 = 4:1)
```

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

Vectors IV

```
(v3 <- 1:2)
```

```
## [1] 1 2
```

```
v1+v2
```

```
## [1] 5 5 5 5
```

```
v1+v3 ##Attention !!!
```

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

```
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
```

Vectors V

Missing values:

```
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
```

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

$$5x + 7y = 1$$
$$4x + 3y + 2z = 2$$
$$6x - 2y - z = 3$$

```
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
```

Matrices III

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

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

```
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
```

Indexing - subsetting II

```
## [1] 66  3 21 96 84
```

```
x2 > 40
```

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

```
x2[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
```

```
## [1] 66 96 84
```

```
x2[x2 > 40] # same as above
```

```
## [1] 66 96 84
```

`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.

Indexing - subsetting III

- `A[i,]` extracts *i*th row.
- `A[,j]` extracts *j*th column.
- 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, DF, and entering some values manually for variables:

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

Data frame II

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

```
d[order(d$v2, d$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:

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

Data frame III

```
##   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 Excel and CSV files I

```r
#For CSV no need for extra package
write.csv(d, "ver1.csv") # sep = "," and  dec = "."
write.csv2(d, "ver2.csv") # sep = ";" and dec = ","
v1 = read.csv("ver1.csv") # sep = "," and dec = "."
v2 = read.csv2("ver2.csv") # sep = ";" and dec = ","
#For Excel we can use openxlsx package
library(openxlsx)
write.xlsx(d,"ver3.xlsx") #write excel
# first get get an 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")
# then, load it using read.xlsx package
dat = read.xlsx("il_ihracat.xlsx",
                cols = 1:16, rows=5:1458, colNames = TRUE)
head(dat)
```

Reading-writing non-excel 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

Data manipulation with dplyr I

```
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")
```

```
dat_x %>%
  filter (province == 34) %>% print(n=4)
```

```
## # A tibble: 204 x 4
##    year province month      export
##   <dbl>    <dbl> <fct>       <dbl>
## 1  2002       34 January  1530915.
## 2  2002       34 February 1362440.
## 3  2002       34 March    1644882.
## 4  2002       34 April    1619166.
## # ... with 200 more rows
```

Data manipulation with dplyr II

```
#group_by(): How to group data (to facilitate "split-apply-combine")
#top_n(): to select the top or bottom entries in each group, ordered by wt
#top 3 provinces by export share by year and month
dat_x %>%
  filter (!is.na(export) & year == 2017) %>%
  group_by(year,month) %>%
  mutate(sh_x = 100*export/sum(export)) %>%
  top_n(n=3, wt = sh_x) %>%
  arrange(month, -sh_x) %>% print(4)

## # A tibble: 36 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       35 January   690017.  6.13
```

Data manipulation with dplyr III

```
##  4  2017       34 February 6182411. 51.1
##  5  2017       16 February  881377.  7.29
##  6  2017       41 February  712545.  5.89
##  7  2017       34 March    7453151. 51.5
##  8  2017       16 March     952632.  6.58
##  9  2017       41 March     880979.  6.09
## 10  2017       34 April    6828343. 53.1
## # ... with 26 more rows
```

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

Data manipulation with 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 highes exports after 2014, default is increasing
dat_x2 %>%
  filter (province !=34) %>%
  group_by(year) %>%
  filter(min_rank(desc(export))==1 & year > 2014)
```

Data manipulation with 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.
```

```
## provinces with lowest exports before 208
dat_x2 %>%
  group_by(year) %>%
  filter(min_rank(export)==1 & year < 2008)
```

Data manipulation with dplyr VI

```
## # A tibble: 6 x 3
## # Groups:   year [6]
##   province  year export
##      <dbl> <dbl>  <dbl>
## 1       29  2002  23.5
## 2       29  2004 119.
## 3       29  2006   9.27
## 4       49  2005  41.1
## 5       62  2007  67.4
## 6       69  2003 140.
```

```
dat_x2 %>%
  group_by(year) %>%
  mutate(million_x=ifelse(export>=1000000, 1, 0)) %>%
  count(million_x)
```

Data manipulation with dplyr VII

```
## # A tibble: 34 x 3
## # Groups:   year [17]
##     year million_x     n
##    <dbl>     <dbl> <int>
##  1  2002         0    75
##  2  2002         1     5
##  3  2003         0    74
##  4  2003         1     5
##  5  2004         0    71
##  6  2004         1     8
##  7  2005         0    73
##  8  2005         1     8
##  9  2006         0    73
## 10  2006         1     8
## # ... with 24 more rows
```

Data manipulation with dplyr VIII

```
dat_x2 %>%
  group_by(year) %>%
  mutate(million_x=ifelse(export>=1000000, 1, 0)) %>%
  count(year, wt=million_x)
```

```
## # A tibble: 17 x 2
## # Groups:   year [17]
##     year     n
##    <dbl> <dbl>
##  1  2002     5
##  2  2003     5
##  3  2004     8
##  4  2005     8
##  5  2006     8
##  6  2007    10
##  7  2008    13
##  8  2009    11
```

Data manipulation with dplyr IX

```
##  9  2010    14
## 10  2011    15
## 11  2012    16
## 12  2013    18
## 13  2014    16
## 14  2015    15
## 15  2016    15
## 16  2017    17
## 17  2018     1
```

```
dat_x2 %>%
  group_by(year) %>%
  mutate(million_x=ifelse(export>=1000000, 1, 0)) %>%
  count(year, wt=million_x, sort = TRUE)
```

Data manipulation with dplyr X

```
## # A tibble: 17 x 2
## # Groups:   year [17]
##     year     n
##    <dbl> <dbl>
##  1  2013    18
##  2  2017    17
##  3  2012    16
##  4  2014    16
##  5  2011    15
##  6  2015    15
##  7  2016    15
##  8  2010    14
##  9  2008    13
## 10  2009    11
## 11  2007    10
## 12  2004     8
## 13  2005     8
```

Data manipulation with dplyr XI

```
## 14  2006     8
## 15  2002     5
## 16  2003     5
## 17  2018     1
```

```
dat_x2 %>%
  group_by(year) %>%
  mutate(million_x=ifelse(export>=1000000, 1, 0)) %>%
  tally(million_x)
```

```
## # A tibble: 17 x 2
##     year     n
##    <dbl> <dbl>
##  1  2002     5
##  2  2003     5
##  3  2004     8
##  4  2005     8
```

Data manipulation with dplyr XII

```
##  5  2006     8
##  6  2007    10
##  7  2008    13
##  8  2009    11
##  9  2010    14
## 10  2011    15
## 11  2012    16
## 12  2013    18
## 13  2014    16
## 14  2015    15
## 15  2016    15
## 16  2017    17
## 17  2018     1
```

Data manipulation with dplyr XIII

```
dat_x2 %>%
  group_by(province) %>%
    summarise(avg_x = mean(export),
              min_x = min(export),
              max_x = max(export),
              nobs = n()) %>%
  print(n=8)
```

```
## # A tibble: 81 x 5
##   province    avg_x   min_x    max_x  nobs
##      <dbl>    <dbl>   <dbl>    <dbl> <int>
## 1        1 1258992. 150322. 1916196.    17
## 2        2  113379.   8097.  542820.    17
## 3        3  211896.  24786.  362111.    17
## 4        4   43001.   2776.   87645.    17
## 5        5   43974.   1312.   98218.    17
## 6        6 4846643. 529935. 8102722.    17
```

Data manipulation with dplyr XIV

```
## 7         7  706496. 115651. 1240524.    17
## 8         8   47918.   3480.   91620.    17
## # ... with 73 more rows

## export growth rates by province
dat_x2 %>%
  group_by(province) %>%
  arrange(province, year) %>%
  mutate(lag_x =  dplyr::lag(export, n = 1)) %>%
  mutate(gr_x =  100*(export - lag_x)/lag_x)  %>%
  filter(year == 2017) %>%
  arrange(-year, province)  %>% print(4)
```

Data manipulation with dplyr XV

```
## # A tibble: 81 x 5
## # Groups:   province [81]
##    province  year   export    lag_x   gr_x
##       <dbl> <dbl>    <dbl>    <dbl>  <dbl>
##  1        1  2017 1822782. 1607018.  13.4
##  2        2  2017  131623.  338329. -61.1
##  3        3  2017  318463.  296187.   7.52
##  4        4  2017   43092.   52134. -17.3
##  5        5  2017   92477.   76165.  21.4
##  6        6  2017 6740237. 6463475.   4.28
##  7        7  2017 1240524.  997112.  24.4
##  8        8  2017   52065.   55859.  -6.79
##  9        9  2017  702007.  617679.  13.7
## 10       10  2017  537182.  519260.   3.45
## # ... with 71 more rows
```

Data manipulation with dplyr XVI

```
## best performing province: cumulative change of x_sh
dat_x2 %>%
  filter (!is.na(export)) %>%
  group_by(year) %>%
  mutate(sh_x = 100*export/sum(export)) %>%
  arrange(province, year) %>%
  group_by(province) %>%
  mutate(c_diff_x = sh_x - first(sh_x)) %>%
  arrange(-c_diff_x) %>%
  filter(year==2017)   %>% print(4)

## # A tibble: 81 x 5
## # Groups:   province [81]
##    province  year   export  sh_x c_diff_x
##       <dbl> <dbl>    <dbl> <dbl>    <dbl>
##  1       27  2017 6607631. 4.21     2.49
##  2       54  2017 5249859. 3.34     2.16
```

Data manipulation with dplyr XVII

```
##  3       41  2017 8095543. 5.16     1.64
##  4       42  2017 1548194. 0.986    0.625
##  5       31  2017 2333958. 1.49     0.517
##  6       47  2017  910275. 0.580    0.515
##  7       45  2017 1989714. 1.27     0.402
##  8        7  2017 1240524. 0.790    0.330
##  9       46  2017  955209. 0.608    0.302
## 10       73  2017  471547. 0.300    0.242
## # ... with 71 more rows
```

```
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
```

Data manipulation with dplyr XVIII

```
x_perc %>% filter(year > 2013)
```

```
## # A tibble: 5 x 4
##    year    p10     p50      p90
##   <dbl>  <dbl>   <dbl>    <dbl>
## 1  2014  6062. 214064. 2115434.
## 2  2015  7361. 186820. 1839282.
## 3  2016  9219. 167067. 1874348.
## 4  2017 12737. 162682. 2333958.
## 5  2018   966.  15323.  188769.
```

```
x_perc %>%
  mutate_each(funs("log"=log), p10,p50) -> x_perc2
```

```
## `mutate_each()` is deprecated.
## Use `mutate_all()`, `mutate_at()` or `mutate_if()` instead.
## To map `funs` over a selection of variables, use `mutate_at()`
```

Data manipulation with dplyr XIX

```
x_perc2
```

```
## # A tibble: 17 x 6
##     year    p10     p50      p90 p10_log p50_log
##    <dbl>  <dbl>   <dbl>    <dbl>   <dbl>   <dbl>
##  1  2002   576.  21338.  431330.    6.36    9.97
##  2  2003  1190.  26689.  620829.    7.08   10.2
##  3  2004  2015.  34725.  892258.    7.61   10.5
##  4  2005  2095.  46125.  965882.    7.65   10.7
##  5  2006  1759.  59381.  958987.    7.47   11.0
##  6  2007  3893.  87802. 1280695.    8.27   11.4
##  7  2008  5337. 103431. 1762181.    8.58   11.5
##  8  2009  6060.  93267. 1417802.    8.71   11.4
##  9  2010  6414. 120037. 1698405.    8.77   11.7
## 10  2011  5910. 143675. 2050555.    8.68   11.9
## 11  2012  6226. 150370. 2039566.    8.74   11.9
## 12  2013  9829. 190564. 2097846.    9.19   12.2
```

Data manipulation with dplyr XX

```
## 13  2014  6062. 214064. 2115434.    8.71   12.3
## 14  2015  7361. 186820. 1839282.    8.90   12.1
## 15  2016  9219. 167067. 1874348.    9.13   12.0
## 16  2017 12737. 162682. 2333958.    9.45   12.0
## 17  2018   966.  15323.  188769.    6.87    9.64
```

Describing data I

```
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")


myvars = c("hwage","educ","female","exper","emp_sect")
head(hls[,myvars])

##   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
## 4 58.33   15      0    21     priv
## 5  3.89    8      0    16     priv
## 6  1.46    8      0    49     priv

str(hls[,myvars])
```

Describing data II

```
## 'data.frame': 762 obs. of  5 variables:
##  $ hwage   : num  8.75 2.92 2.53 58.33 3.89 ...
##  $ educ    : int  2 2 5 15 8 8 5 15 5 15 ...
##  $ female  : int  0 1 1 0 0 0 0 1 0 1 ...
##  $ exper   : int  33 2 22 21 16 49 22 6 17 2 ...
##  $ emp_sect: Factor w/ 3 levels "other","priv",..: 3 2 2 2 2 2 2 2 2 3 ...
```

```
summary(hls[,myvars])
```

```
##      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
```

Describing data III

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

```
summary(hls[,"exper"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0    10.0    18.0    18.9    27.0    72.0
```

```
seq(0,8,by=3)
```

```
## [1] 0 3 6
```

```
cut(0:8, breaks=seq(0,8,by=3))
```

Describing data IV

```
## [1] <NA>  (0,3] (0,3] (0,3] (3,6] (3,6] (3,6] <NA>  <NA>
## Levels: (0,3] (3,6]
```

```
cut(0:8, breaks=seq(0,8,by=3), include.lowest = TRUE)
```

```
## [1] [0,3] [0,3] [0,3] [0,3] (3,6] (3,6] (3,6] <NA>  <NA>
## Levels: [0,3] (3,6]
```

```
cut(0:8, breaks=seq(0,9,by=3), include.lowest = TRUE)
```

```
## [1] [0,3] [0,3] [0,3] [0,3] (3,6] (3,6] (3,6] (6,9] (6,9]
## Levels: [0,3] (3,6] (6,9]
```

```
cut(0:8, breaks=seq(0,9,by=3), include.lowest = TRUE, right=FALSE)
```

```
## [1] [0,3) [0,3) [0,3) [3,6) [3,6) [3,6) [6,9] [6,9] [6,9]
## Levels: [0,3) [3,6) [6,9]
```

Describing data V

```
hls$exper_gr = cut(hls$exper, breaks = seq(0,60,by=10), right=FALSE,
                   include.lowest = TRUE)

tab = table(hls$exper_gr)
tab

##
##  [0,10) [10,20) [20,30) [30,40) [40,50) [50,60]
##     184     234     200     108      33       1

round( 100*prop.table(tab), 1)

##
##  [0,10) [10,20) [20,30) [30,40) [40,50) [50,60]
##    24.2    30.8    26.3    14.2     4.3     0.1
```

Describing data VI

```
## 2 categorical variables
#tab2 = with(hls, table(gender, e_sect)) # or better
tab2 = xtabs( ~ female+emp_sect, data=hls)
tab2
```

```
##       emp_sect
## female other priv pub
##      0     3  456 135
##      1     6  101  61
```

```
prop.table(tab2, margin=1) # row sum = 100
```

```
##       emp_sect
## female   other    priv     pub
##      0 0.00505 0.76768 0.22727
##      1 0.03571 0.60119 0.36310
```

```
prop.table(tab2, margin=2) # col sum = 100
```

Describing data VII

```
##        emp_sect
## female other  priv   pub
##      0 0.333 0.819 0.689
##      1 0.667 0.181 0.311
```

```
tab3 = xtabs(wts ~ female+emp_sect, data=hls)
prop.table(tab3, margin=2)
```

```
##        emp_sect
## female other  priv   pub
##      0 0.333 0.807 0.683
##      1 0.667 0.193 0.317
```

```
## one categorical and one numerical variable, mean
aggregate(hwage ~ exper_gr, FUN = mean, data=hls)
```

Describing data VIII

```
##   exper_gr hwage
## 1   [0,10)  5.34
## 2  [10,20)  6.81
## 3  [20,30)  6.02
## 4  [30,40)  6.27
## 5  [40,50)  7.09
## 6  [50,60]  2.16
```

```
## one categorical and one numerical variable, weighted mean
sapply(
  split(hls, hls$exper_gr),
  FUN=function(dat) weighted.mean(dat$hwage, dat$wts)
  )
```

```
##  [0,10) [10,20) [20,30) [30,40) [40,50) [50,60]
##    5.26    6.89    6.04    6.50    7.08    2.16
```

Describing data IX

```
## or
library(dplyr)
hls %>%
  group_by(exper_gr)  %>%
  summarise(vmean =  weighted.mean(hwage, wts))

## # A tibble: 7 x 2
##   exper_gr vmean
##   <fct>    <dbl>
## 1 [0,10)    5.26
## 2 [10,20)   6.89
## 3 [20,30)   6.04
## 4 [30,40)   6.50
## 5 [40,50)   7.08
## 6 [50,60]   2.16
## 7 <NA>      3.84
```

Describing data X

```
plot(log(hwage) ~ factor(female), hls)
plot(log(hwage) ~ educ, hls)
```