UiO : PROMENTA
Det samfunnsvitenskapelige fakultet

Working with Data (part 1)

E. F. Haghish

25/8/2022

Working with Data

- Loading data into R
- Writing data in different formats
- Summarizing data

Prerequisite

- Working directory
- File path on your computed
  - Relative path
  - Absolute path
- File path is different on Windows and Mac/Linux
- R follows Linux path syntax
- When you load a file into R or save a dataset, you need to specify the file path

Loading data into R

- From internet
- CSV files
  - Comma seperated
  - Semicolon seperated
  - ...
- R data files
  - Rsd
  - RData
- Other statistical software
  - SPSS
  - Stata
  - xlsx

R datasets

- When you load R, it automatically makes a number of datasets available. To list the datasets available in R use the `data()` function:
- For example, `cars` dataset which includes 2 variables, `speed` indicating the speed of car and `dist` indicating the stopping distance. Type `?cars` to read the datasets' help file.

```
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Typing data in R

- R does not have a GUI interface for entering data (Excel, SPSS, Stata have, however)
- You can define data using the `c` function

```r
height = c(170, 172,165, 166, 172, 180, 168, 176)
```

`RDS` vs `RData`

- `RDS` is used for saving and loading a single object
  - `saveRDS()`
  - `readRDS()`
- `RData` is more general and can include many objects in a single file

Reading RDS and RData

```r
df <- readRDS("path/to/file.rds")
df <- load("path/to/file.RData")
```

Cleaning workspace

- You can clear the R worksapce within the RStudio GUI or by typing

```
rm(list = ls())
```

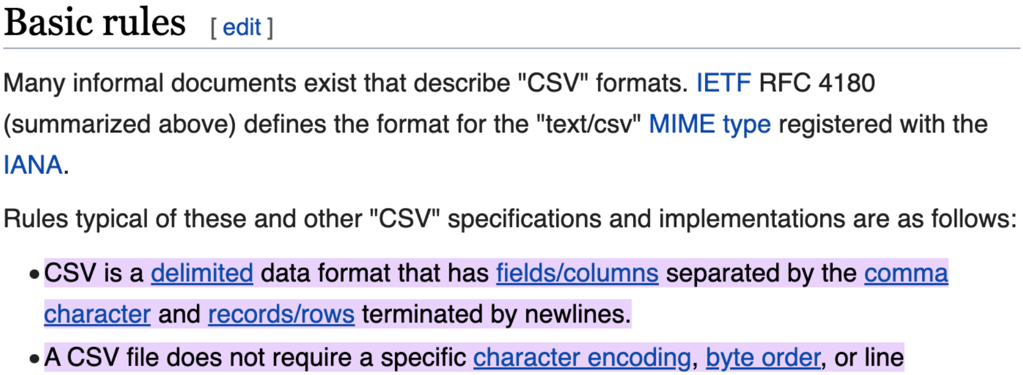
- the `rm` function removes any object that is specified
- it can take an argument of `list =` to get a list of objects you want to remove
- the `ls` function is identical to `objects` and it returns acharacter vector of the object names in the workspace
- `ls` does not remove attached objects. Use the `search` function to see the objects that are attached to your workspace
- use the `detach` function to remove the objects or unload packages from the memory

Saving workspace

- When you exit R, it automatically prompt a question whether you want to save the workspace or not
- You can save the objects in the R workspace using the `save` and `save.image` functions which can save `.RData` or `.rda` which is just the abbreviation.
- these functions can save any sort of object that is currently in R's workspace, scalar, vector, list, dataset, etc.
- you can also `load` the saved worksapace back into R
- You can also save **a single R objects** in `rds` format using `saveRDS` function and read it using the `readRDS`
- This differs from `save` and `load`, which save and restore one or more named objects into an environment
- `rds` can be used to save a dataset within R and read it

CSV data format

- Often is comma separated
- Could be semicolon separated
  - That would require a different syntax!
- Other chatacters might be used for separating the observations
- Each row must start at a new row in the text file



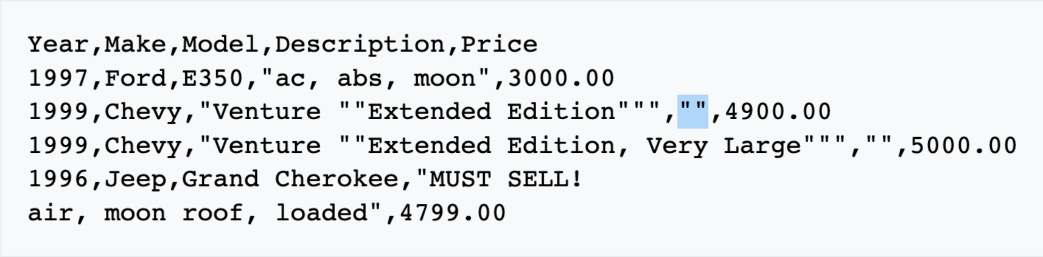
Basic rules [edit]

Many informal documents exist that describe "CSV" formats. IETF RFC 4180 (summarized above) defines the format for the "text/csv" MIME type registered with the IANA.

Rules typical of these and other "CSV" specifications and implementations are as follows:

- CSV is a delimited data format that has fields/columns separated by the comma character and records/rows terminated by newlines.
- A CSV file does not require a specific character encoding, byte order, or line

The above table of data may be represented in CSV format as follows:



```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""","",5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

Comma-, semicolon-, and Tab-Delimited Input Files

- some data sets are comma separated
- R provides three convenience functions, `read.csv`, `read.csv2`, and `read.delim` for working with comma separated data
- use `read.csv` for comma separated `csv` files
- use `read.csv2` for semicolon separated `csv` files
- use `read.delim` for tab-delimited `sep="\t"` data
- It is better to focus on the `read.table` function and learn how to import your data correctly and use all of its arguments. The other functions of the family are just wrappers.

Fixed-Width Input Files

- sometimes input data is stored with no delimiters between the values, but with each variable occupying the same columns on each line of input
- use the `read.fwf` function to read the data
- the `read.fwf` requires you to specify the "format of the data" in terms of `width=` (similar to the idea of `nchar`) to tell R how wide each column is
- The function takes care of trailing white spaces as long as the columns are defined correctly

Using data from the internet

- R can access the data frim the internet
- R can read the data without storing it on the disk
- You can also ask it to store the data on the disk
- The data might be archived (zipped) and sometimes have to unzip it first
- pay attention to the **URL** address. sometimes if the address begins with `HTTPS` - a protocol for secure communication - instead of `HTTP`, a special care is needed (which also depends on your operating system)

Reading CSV data

The following datasets are example data sets to work with within R:
http://www.exploredata.net/Downloads/WHO-Data-Set

```
url = "http://www.exploredata.net/ftp/WHO.csv"
data = read.csv(url)
```

semicolon separated dataset

```
url = "https://raw.githubusercontent.com/haghish/ST516/master/data/seligson.csv"
data = read.csv2(url)
```

Reading files from internet

```
url = "https://github.com/haghish/ST516/blob/master/data/seligson.csv?raw=true"
data = read.csv2(url)
head(data)
```

Downloading files from internet

- use the `download.file` function
- use the `method="curl"` for
- if the dataset is zipped, use the `unzip` function

```
url = "https://github.com/haghish/ST516/blob/master/data/seligson.csv.zip?
download.file(url, "./seligson.zip")
unzip("./seligson.zip")
data = read.csv2("./seligson.csv")
```

- you can also use the `unz` function to access a file in a zipfile without unzipping it
- you should know how to access that file in the zipfile, i.e. the path to it in the archived file (a zipfile can have subdirectories)

```
data = read.csv2(unz('./seligson.zip','seligson.csv'))
```

Test

- use the `airquality` data and drop all of the missing values
- use the `subset` function to select 2 variables in the data set only
- use the `subset` function to drop observations that have `Temp` value below 75
- select a random sample of 10 from the `airquality` dataset

Reading Stata data

```r
library(readstata13) # recommended
library(foreign)
library(haven)

df <- readstata13::read.dta13("path/to/file.dta")
df <- foreign::read.dta("path/to/file.dta")
df <- haven::read_dta("path/to/file.dta")
```

Note

Different packages can return data in different classes, especially for **character** and **factor** variables

Reading SPSS data

```r
library(foreign)
library(haven)

df <- foreign::read.spss("path/to/file.sav")
df <- haven::read_spss("path/to/file.sav")
```

Note

Different packages can return data in different classes, especially for **character** and **factor** variables

Reading Excel sheet

- The readxl package makes it easy to get data out of Excel and into R
- has no external dependencies
- supports `.xls` and `.xlsx`

```r
library(readxl)
library(tidyverse)

# which sheet would you like to read?
df <- readxl::read_excel("path/to/file.xlsx",  sheet = 1)
```

Examples of working with `readxl` package

```
library(readxl)
readxl_example()
```

```
##  [1] "clippy.xls"    "clippy.xlsx"   "datasets.xls"  "datasets.xlsx"
##  [5] "deaths.xls"    "deaths.xlsx"   "geometry.xls"  "geometry.xlsx"
##  [9] "type-me.xls"   "type-me.xlsx"
```

```
readxl_example("clippy.xls")
```

```
## [1] "/Library/Frameworks/R.framework/Versions/4.2/Resources/library/readxl/extdata/clippy.xls"
```

```
xlsx_example <- readxl_example("datasets.xlsx")
read_excel(xlsx_example)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
## # i Use `print(n = ...)` to see more rows
```

```
xls_example <- readxl_example("datasets.xls")
read_excel(xls_example)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
## # i Use `print(n = ...)` to see more rows
```

Specify a worksheet by name or number.

```
read_excel(xlsx_example, sheet = "chickwts")
```

```
## # A tibble: 71 x 2
##    weight feed
##     <dbl> <chr>
##  1    179 horsebean
##  2    160 horsebean
##  3    136 horsebean
##  4    227 horsebean
##  5    217 horsebean
##  6    168 horsebean
##  7    108 horsebean
##  8    124 horsebean
##  9    143 horsebean
## 10    140 horsebean
## # ... with 61 more rows
## # i Use `print(n = ...)` to see more rows
```

```
read_excel(xls_example, sheet = 4)
```

```
## # A tibble: 1,000 x 5
##      lat  long depth   mag stations
##    <dbl> <dbl> <dbl> <dbl>    <dbl>
##  1 -20.4  182.   562   4.8       41
##  2 -20.6  181.   650   4.2       15
##  3 -26    184.    42   5.4       43
##  4 -18.0  182.   626   4.1       19
##  5 -20.4  182.   649   4         11
##  6 -19.7  184.   195   4         12
##  7 -11.7  166.    82   4.8       43
##  8 -28.1  182.   194   4.4       15
##  9 -28.7  182.   211   4.7       35
## 10 -17.5  180.   622   4.3       19
## # ... with 990 more rows
## # i Use `print(n = ...)` to see more rows
```

Specify rows and columns!

```
read_excel(xlsx_example, range = "mtcars!B1:D2")
```

```
## # A tibble: 1 x 3
##      cyl  disp    hp
##    <dbl> <dbl> <dbl>
## 1      6   160   110
```

Summarizing observations

- the `head` function displays the first 5 observations of the data
- the `tail` function displays the last 5 observations of the data
- you can also display a particular number of observations. For example, only 1 row just to get a sense of the data:

```
head(cars, n = 1)
```

```
##   speed dist
## 1     4    2
```

summary function

- use the `summary()` to get summary statistics about your data. The function can get a subset or the whole dataset

```
summary(cars$speed)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    15.0    15.4    19.0    25.0
```

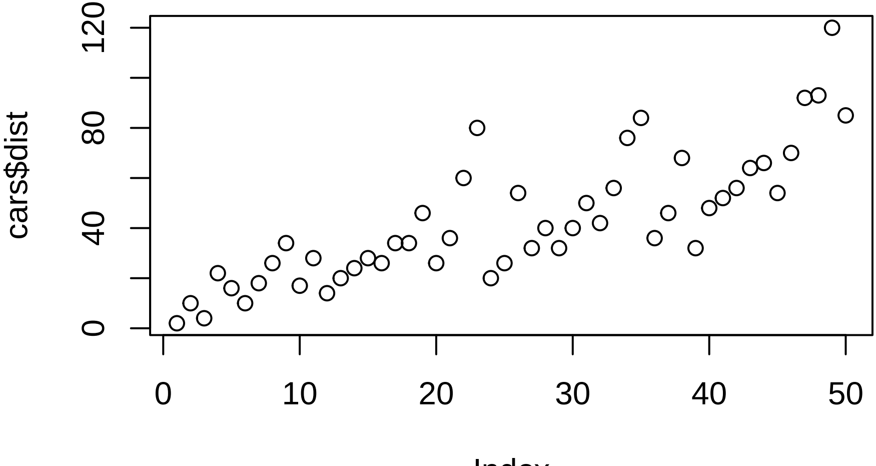
```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

scatter plot

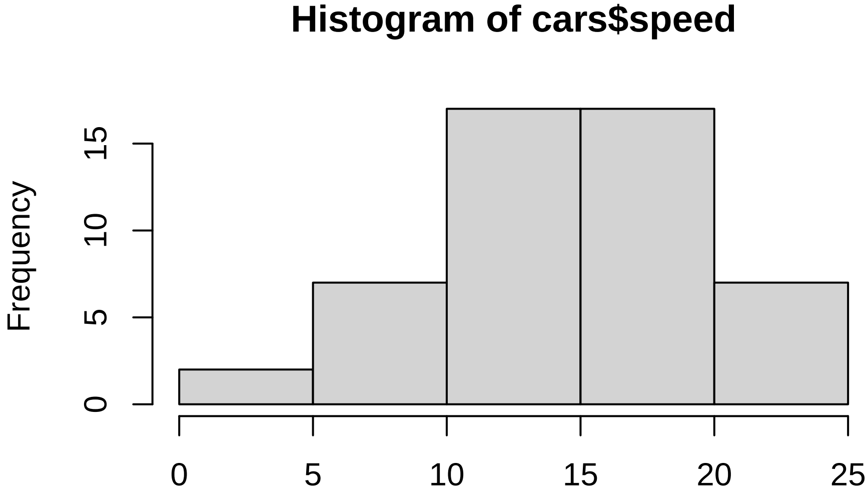
- for creating a scatter plot, use the `plot` function to view your data:

```
plot(cars$dist)
```



histogram

```
hist(cars$speed)
```


Histogram of cars$speed
Frequency