R data types: Data Frames

Alberto Garfagnini

Università di Padova

R lecture 4

R Data frames

- two important S3 vectors built on top of lists are `data frames` and `tibbles`
- a `data frame` is like a matrix, with a 2-dim rows-and-columns structure
- it's a named list of vectors, with attributes for columns and rows names, (`names`, `row.names`), belonging to the `data.frame` class

- technically, a `data frame` is a `list` with all equal length vectors

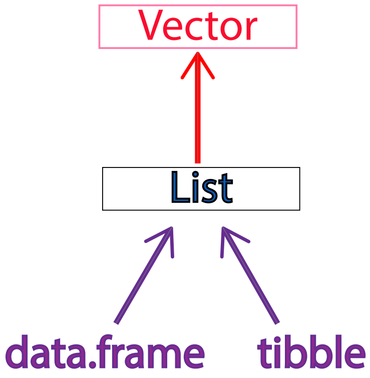
```
df1 <- data.frame(x = 1:3, y = letters[1:3])
typeof(df1)
%> [1] "list"

attributes(df1)
%> $names
%> [1] "x" "y"

%> $class
%> [1] "data.frame"

%> $row.names
%> [1] 1 2 3

str(df1)
%> 'data.frame':        3 obs. of  2 variables:
%>  $ x: int  1 2 3
%>  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```



R Data Frames : examples

- we have a table with the results of two exams for the stdent of an hipotetical course, and we want to import them in a data.frame

Exam_1	Exam_2	Gender
27	25	M
28	30	F
. . .		
27	27	M
25	28	F

```
exam1 <- c(27,28,24,24,30,26,23,23,24,28,27,25)
exam2 <- c(25,30,26,24,30,30,25,25,30,28,27,28)
gender <- c("M","F","M","M","M","M","M","M","F","F","M","F")

dc <- data.frame(exam1, exam2, gender)
head(dc, n=2) # extract the first two lines of the data frame
%>    exam1 exam2 gender
%> 1     27    25      M
%> 2     28    30      F


dc1 <- data.frame(exam1, exam2, gender,
                  stringsAsFactors = FALSE)
str(dc1)
'data.frame':   12 obs. of  3 variables:
 $ exam1 : num  27 28 24 24 30 26 23 23 24 28 ...
 $ exam2 : num  25 30 26 24 30 30 25 25 30 28 ...
 $ gender: chr  "M" "F" "M" "M" ...
```

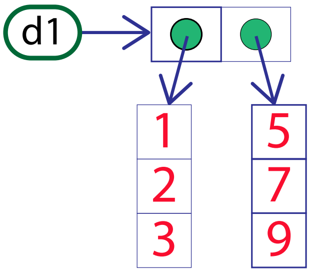
From R 4.0
stringsAsFactors = FALSE
by default

R Data Frames objects creation

- Data frames are list of vectors, therefore copy-on-modify has important consequences

```
d1 <- data.frame(x = c(1, 2, 3),
                 y = c(5, 7, 9))
d1
%>   x y
%> 1 1 5
%> 2 2 7
%> 3 3 9
```

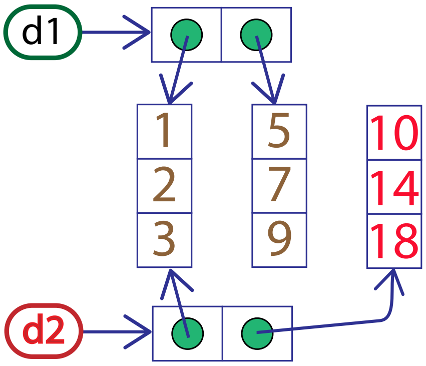
```
> lobstr::ref(d1)
█ [1:0x55905d24e9e8] <df[,2]>
├─x = [2:0x55905e564eb8] <dbl>
└─y = [3:0x55905e564e68] <dbl>
```



- if we modify a column ➜ only the reference to the new column will be updated

```
d2 <- d1
d2[, 2] <- d2[, 2] * 2
d2
%>   x  y
%> 1 1 10
%> 2 2 14
%> 3 3 18
```

```
> lobstr::ref(d1)
█ [1:0x55905d24e9e8] <df[,2]>
├─x = [2:0x55905e564eb8] <dbl>
└─y = [3:0x55905e564e68] <dbl>
```



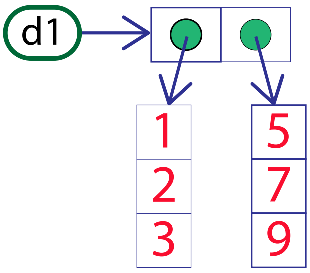
```
> lobstr::ref(d2)
█ [1:0x55905c9f1628] <df[,2]>
├─x = [2:0x55905e564eb8] <dbl>
└─y = [3:0x55905e92cd78] <dbl>
```

R Data Frames objects creation

- Data frames are list of vectors, therefore `copy-on-modify` has important consequences

```
d1 <- data.frame(x = c(1, 2, 3),
                 y = c(5, 7, 9))
d1
%>   x y
%> 1 1 5
%> 2 2 7
%> 3 3 9
```

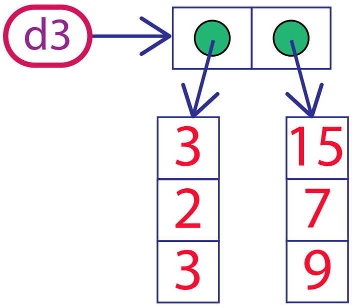
```
> lobstr::ref(d1)
█ [1:0x55905d24e9e8] <df[,2]>
├─x = [2:0x55905e564eb8] <dbl>
└─y = [3:0x55905e564e68] <dbl>
```



- but if any row is modified ➜ every column is modified because every column must be copied

```
d3 <- d1
d3[1, ] <- d3[1, ] * 3
d3
%>   x  y
%> 1 3 15
%> 2 2  7
%> 3 3  9
```

```
> lobstr::ref(d3)
█ [1:0x55905e6f1058] <df[,2]>
├─x = [2:0x55905ea0c238] <dbl>
└─y = [3:0x55905ea0c1e8] <dbl>
```

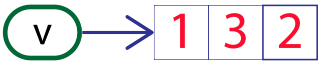


Modify-in-place

- Modifying an R object usually creats a copy
- but there are 2 exceptions:
- objects with single binding get a special performance optimization
- environments, a special type of object, are always modified in place

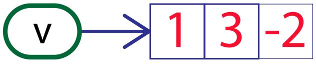
```
v <- c(1, 3, 2)
lobstr::obj_addr(v)
%> [1] "0x55905ea0f4e8"
```

```
lobstr::obj_addr(v)
%> [1] "0x55905ea0f4e8"
```



```
v[[3]] <- -2
lobstr::obj_addr(v)
%> [1] "0x55905ea0f4e8"
```

```
lobstr::obj_addr(v)
%> [1] "0x55905ea0f4e8"
```



- but it is very difficult to predicts when R applies this optimization
- concerning object binding, R only counts 0, 1 or `MANY`
- it means that if an object has 2 bindings (i.e. `many`), and one gets deleted, the reference does not go back to 1 (`many - 1 = many`)
- when a funtion is called, it makes a reference to the object ➜ it is very difficult to predict weather or not a copy will occur
- cfr: `https://developer.r-project.org/Refcnt.html`

Accessing data frames elements

- a data frame is a list, therefore we can access them via component index value [[j]] or via component names

```
str(dc)
%> 'data.frame':   12 obs. of  3 variables:
%>  $ exam1 : num  27 28 24 24 30 26 23 23 24 28 ...
%>  $ exam2 : num  25 30 26 24 30 30 25 25 30 28 ...
%>  $ gender: Factor w/ 2 levels "F","M": 2 1 2 2 2 2 2 2 1 1 ...

dc[[1]] # access by component index
%>  [1] 27 28 24 24 30 26 23 23 24 28 27 25

dc$exam1 # access by component name
%>  [1] 27 28 24 24 30 26 23 23 24 28 27 25
%> Levels: F M
```

- but a data frame can be treated in a matrix-like fashion, as well

```
dc[,1] # select column 1
%>  [1] 27 28 24 24 30 26 23 23 24 28 27 25

dc[1,1] # and access the single element, as well
%> [1] 27
```

Data frames row names

- data frames allow to label each row with a name, a character vector containing only unique names

```
df1 <- data.frame( age = c(35, 25, 18),
                   hair = c("blond", "brown", NA),
                   row.names = c("Bob", "Tom", "Sam"))
df1
%>     age  hair
%> Bob  35 blond
%> Tom  25 brown
%> Sam  18  <NA>

names(df1)
%> [1] "age"  "hair"

row.names(df1)
%> [1] "Bob" "Tom" "Sam"
```

- but row names are a bad practice:

(1) metadata is metadata : storing it in a different way to the rest of data is a bad idea

(2) row names are a poor abstraction for labeling rows : they only work when a row can be identified by a single string

(3) row names must be unique : any duplication of rows will create new row names ➜ complicated ”string surgery” may be needed

Advanced data frames : data selection

```
dc[2:4,]  # Select only rows 2:4
%>   exam1 exam2 gender
%> 2    28    30      F
%> 3    24    26      M
%> 4    24    24      M

dc[-(2:10),] # drop rows 2:10
%>    exam1 exam2 gender
%> 1     27    25      M
%> 11    27    27      M
%> 12    25    28      F
```

- with the sample function , data can be selected at random

```
dc[sample(1:12,3),]  # select 3 rows at random
%>   exam1 exam2 gender
%> 8    23    25      M
%> 9    24    30      F
%> 6    26    30      M

dc[sample(1:12,3),]  # select 3 rows at random
%>    exam1 exam2 gender
%> 1     27    25      M
%> 10    28    28      F
%> 2     28    30      F
```

Advanced data frames : data selection

- suppose we want to extract all columns that contain numbers, rather than characters or logicals, from a data frame

```
dc[,sapply(dc,is.numeric)]
%>    exam1 exam2
%> 1     27    25
%> 2     28    30
%> 3     24    26
%> 4     24    24
%> 5     30    30
%> 6     26    30
%> 7     23    25
%> 8     23    25
%> 9     24    30
%> 10    28    28
%> 11    27    27
%> 12    25    28
```

```
dc <- data.frame(exam1, exam2, gender)
str(dc)
'data.frame':  12 obs. of  3 variables:
$ exam1 : num  27 28 ...
$ exam2 : num  25 30 ...
$ gender: Fact w/ 2 levels "F","M": 2 1
```

- and now we want to get only factors (and remove numerics)

```
dc[,sapply(dc,is.factor)]
%>  [1] M F M M M M M M F F M F
%> Levels: F M
```

Advanced data frames and NA elements

- sometimes our data frame can have missing values (NA) and we may need to omit those values
- we can create a shorter data frame using the na.omit() function

```
data
%>    slope  pH  area
%> 1     11 4.1   3.6
%> 2     NA 5.2   5.1
%> 3      3 4.9   2.8
%> 4      5  NA   3.7
```

```
na.omit(data)
%>    slope  pH  area
%> 1     11 4.1   3.6
%> 3      3 4.9   2.8
```

```
clean_data <- na.exclude(data)
clean_data
%>    slope  pH  area
%> 1     11 4.1   3.6
%> 3      3 4.9   2.8
```

```
lapply(clean_data, mean)
%> $slope
%> [1] 7
%> $pH
%> [1] 4.5
%> $area
%> [1] 3.2
```

```
# Let's count the missing values
apply(apply(data,2,is.na),2,sum)
%> slope    pH   area
%>      1     1      0
```

Advanced data frames : sorting elements

```
dc[order(exam1),]
%>     exam1 exam2 gender
%> 7      23    25      M
%> 8      23    25      M
...
```

```
dc[order(exam1, decreasing=TRUE),]
%>     exam1 exam2 gender
%> 5      30    30      M
%> 2      28    30      F
```

- dc[order(gender, exam1, exam2, decreasing=TRUE),]

```
%>     exam1 exam2 gender
%> 5      30    30      M
%> 11     27    27      M
%> 1      27    25      M
%> 6      26    30      M
%> 3      24    26      M
%> 4      24    24      M
%> 7      23    25      M
%> 8      23    25      M
%> 2      28    30      F
%> 10     28    28      F
%> 12     25    28      F
%> 9      24    30      F
```

Summary of data selection in data frames

- given a data frame called data, we assume n is a row number, and m is one of the column.
- the syntax [n,] selects all the columns given row n, while [,m] selects all the rows with column m

command	meaning
`data[n,]`	select all of the columns from row n of the data frame
`data[-n,]`	drop the whole of row n from the data frame
`data[1:n,]`	select all of the columns from rows 1 to n of the data frame
`data[-(1:n),]`	drop all of the columns from rows 1 to n of the data frame
`data[c(i,j,k),]`	select all of the columns from rows i, j, and k of the data frame
`data[x > y,]`	use a logical test ($x > y$) to select all columns from certain rows
`data[,m]`	select all of the rows from column m of the data frame
`data[,-m]`	drop the whole of column m from the data frame
`data[,1:m]`	select all of the rows from columns 1 to m of the data frame
`data[,-(1:m)]`	drop all of the rows from columns 1 to m of the data frame
`data[,c(i,j,k)]`	select all of the rows from columns i, j, and k of the data frame
`data[,x > y]`	use a logical test ($x > y$) to select all rows from certain columns
`data[,c(1:m,i,j,k)]`	add duplicate copies of columns i, j, and k to the data frame
`data[x > y,a != b]`	extract certain rows ($x > y$) and certain columns ($a! = b$)
`data[c(1:n,i,j,k),]`	add duplicate copies of rows i, j, and k to the data frame

The tibble data structure

- it is a modern reimagining of the data frame
- it is provided by the tibble package which is part of the tidyverse core library

```
library(tidyverse)
%>   Attaching packages --------   tidyverse 1.3.0
%>   ggplot2 3.3.0       purrr    0.3.3
%>   tibble  2.1.3       dplyr    0.8.5
%>   tidyr   1.0.2       stringr 1.4.0
%>   readr   1.3.1       forcats 0.5.0
%>   Conflicts ------------ tidyverse_conflicts()
%>   dplyr::filter() masks stats::filter()
%>   dplyr::lag()    masks stats::lag()
```

- a data frame can be converted to a tibble

```
dct <- tibble(dc)
dct
%> # A tibble: 12 x 1
%>    dc$exam1 $exam2 $gender
%>       <dbl>  <dbl> <fct>
%>   1      27     25 M
  ...
%> 12       25     28 F
```

https://tibble.tidyverse.org/

- or created from vectors (as for the data frame)

```
dct <- data.frame(exam1 = c(27,28,24,24,30,26,23,23,24,28,27,25),
                  exam2 = exam2, gender)
```

Tibbles vs. data.frame : printing

- two main differences in the usage of a tibble versus a data.frame:
- printing and subsetting
- Tibbles have a refined print method that shows only the first 10 rows:

```
atb <- tibble( a = lubridate::now() + runif(1e3) * 86400,
               b = 1:1e3,
               c = runif(1e3),
               d = sample(letters, 1e3, replace = TRUE) )
atb
%> # A tibble: 1,000 x 4
%>    a                       b      c d
%>    <dttm>              <int>  <dbl> <chr>
%>  1 2020-03-24 08:56:26     1 1.00   q
%>  2 2020-03-24 10:57:29     2 0.996  z
%>  3 2020-03-24 00:32:33     3 0.620  r
%>  4 2020-03-24 01:16:23     4 0.804  d
%>  5 2020-03-24 03:32:17     5 0.311  e
%>  6 2020-03-24 00:22:27     6 0.206  u
%>  7 2020-03-23 12:58:48     7 0.0390 e
%>  8 2020-03-23 20:36:03     8 0.449  d
%>  9 2020-03-24 01:29:00     9 0.271  r
%> 10 2020-03-24 10:52:01    10 0.460  v
%> # with 990 more rows
```

Tibbles vs. data.frame : subsetting

- tibbles can extract by name or position

```
tb1 <- tibble( x = runif(5),
               y = rnorm(5) )

# Extract by name
tb1$x
#> [1] 0.7330 0.2344 0.6604 0.0329 0.4605
tb1[["x"]]
#> [1] 0.7330 0.2344 0.6604 0.0329 0.4605

# Extract by position
tb1[[1]]
#> [1] 0.7330 0.2344 0.6604 0.0329 0.4605
```

- tibbles are more strict than `data.frame`: they never do partial matching, and they will generate a warning if the column you are trying to access does not exist

```
tdc <- as_tibble(dc)
```

```
dc$gender
#>   [1] M F M ... F M F
#> Levels: F M

dc$gen
#>   [1] M F M ... F M F
#> Levels: F M

.
```

```
tdc$gender
#>   [1] M F M ... F M F
#> Levels: F M

tdc$gen
#> NULL
#> Warning message:
#> Unknown or uninitialised column:
   'gen'.
```

Data Input

Data Input

- numbers can be inputed through the keyboard, from the Clipboard, from an external file on disk, or from an external file on the Web
- use the concatenate function for up to 10 numbers
- and scan() for typing or pasting data into a vector

```
y <- c (6,7,3,4,8,5,6,2)

tu <- scan()
%> 1: 6
%> 2: 3
%> 3: 4
%> 4: 2
%> 5:
%> Read 4 items
tu
%> [1] 6 3 4 2
```

- but the easiest way is to read data from a file (or from the Web), already shaped in a data frame format

Data Input using read.table()

- the read.table() function reads data from a local file and creates a data frame

```
data <- read.table("yield.txt",header=T)

data
%>    year wheat barley oats rye corn
%> 1  1980   5.9    4.4  4.1 3.8  4.4
%> 2  1981   5.8    4.4  4.3 3.7  4.1
%> 3  1982   6.2    4.9  4.4 4.1  4.0
...
%> 27 2006   8.0    5.9  6.0 6.1  4.5
%> 28 2007   7.2    5.7  5.5 5.7  3.9
%> 29 2008   8.3    6.0  5.8 6.1  4.4
```

- the parameter header = T tells R to use the first row as column names

```
names(data)
%> [1] "year"   "wheat"  "barley" "oats"   "rye"    "corn"

str(data)
%> 'data.frame':        29 obs. of  6 variables:
%>  $ year  : int  1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 ...
%>  $ wheat : num  5.9 5.8 6.2 6.4 7.7 6.3 7 6 6.2 6.7 ...
%>  $ barley: num  4.4 4.4 4.9 4.7 5.6 5 5.2 5 4.7 4.9 ...
%>  $ oats  : num  4.1 4.3 4.4 4.3 4.9 4.6 5.2 4.6 4.6 4.5 ...
%>  $ rye   : num  3.8 3.7 4.1 3.7 4.7 4.6 4.7 4.8 4.6 4.8 ...
%>  $ corn  : num  4.4 4.1 4 4.1 4.7 4.3 4.3 4.5 4.2 3.8 ...
```

Data Input using read.table()

- if the separator between variable names and data fields are not blanks or tabs, (\t), a different separator can be specified with the sep="," option

```
datav <- read.table("bowens.csv",sep=",", header=T)

str(datav)
%> 'data.frame':        733 obs. of  3 variables:
%>  $ place: Factor w/ 727 levels "Abingdon","Admoor␣Copse",..: 1 2 3 ..
%>  $ east : int  50 60 48 70 59 60 60 59 61 60 ...
%>  $ north: int  97 70 87 73 65 65 63 66 63 67 ...
```

`read.table()` : separators and decimal points

- the default field separator character in `read.table()` is `sep=" "`: which identifies with one or more spaces, one or more tabs (`\t`), and one or more newlines (`\n`)
- for comma-separated fields use `read.csv()`
- for semicolon-separated fields use `read.csv2()`
- for tab-delimited fields with decimal points as a commas, use `read.delim2()`

```
File: bowens.csv
--------------------
|place,east,north  |
|Abingdon,50,97    |
|Admoor Copse,60,70|
|...               |
|Youlbury,48,3     |
--------------------

str(bw)
%> 'data.frame':   733 obs. of  3 variables:
%>  $ place: Factor w/ 727 levels "AERE␣Harwell",..: 2 3 1 4 5 ...
%>  $ east : int  50 60 48 70 59 60 60 59 61 60 ...
%>  $ north: int  97 70 87 73 65 65 63 66 63 67 ...
```

`read.csv()` and `read.delim()`

- additional functions to read a file in table format exist

```
> ?read.table
...
read.delim(file, header = TRUE, sep = "\t", quote = "\"",
␣␣␣␣␣␣␣␣␣␣␣dec␣=␣".",␣fill␣=␣TRUE,␣comment.char␣=␣"",␣...)
...
read.csv(file,␣header␣=␣TRUE,␣sep␣=␣",",␣quote␣=␣"\"",
         dec = ".", fill = TRUE, comment.char = "", ...)
...
read.csv2(file, header = TRUE, sep = ";", quote = "\"",
␣␣␣␣␣␣␣␣␣␣dec␣=␣",",␣fill␣=␣TRUE,␣comment.char␣=␣"",␣...)
...
read.delim2(file,␣header␣=␣TRUE,␣sep␣=␣"\t",␣quote␣=␣"\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

- further detailed instructions in the 'R Data Import/Export' manual:

  https://cran.r-project.org/doc/manuals/r-release/R-data.html

Data Input from the Web and from DB

- R can read data form the network using HTTP by specifying the file URL

```
wc <- read.table("https://tinyurl.com/murders-txt", header=T)

str(wc)
%> 'data.frame':   50 obs. of  4 variables:
%>  $ state      : Factor w/ 50 levels "Alabama","Alaska",..: 1 2 ...
%>  $ population: int  3615 365 2212 2110 21198 2541 3100 ...
%>  $ murder     : num  15.1 11.3 7.8 10.1 10.3 6.8 3.1 6.2 ...
%>  $ region     : Factor w/ 4 levels "North.Central",..: 3 4 4 ...
```

- several packages available on CRAN to help R communicate with DBMSs:

  combining a unified 'front-end' package with a 'back-end' module, several common relational databases can be accessed (RMySQL, ROracle, RPostgreSQL and RSQLite)

- finally, R can read binary data files: NASA's HDF5 (Hierarchical Data Format, https://www.hdfgroup.org/HDF5/) and UCAR's netCDF data files (network Common Data Form, http://www.unidata.ucar.edu/software/netcdf/)

- and image files

Example: data Input from the Web

- let's retrieve the latest data on the COVID-19 Virus infection from the European Centers for Disease Control https://www.ecdc.europa.eu/en

- R can read data form the network using HTTP by specifying the file URL

European Centre for Disease Prevention and Control

An agency of the European Union

ecdc
All topics: A to Z
News & events
Publications & data
Tools
About us

Coronavirus disease

The disease is rapidly spreading worldwide and the number of cases in Europe is rising with increasing pace in several affected areas.

Latest information on COVID-19
Coronavirus disease (COVID-19)
COVID-19: Social distancing measures
Antimicrobial resistance in zoonoses
2019/2020 influenza season

COVID-19

Several countries are now experiencing sustained local transmission of coronavirus disease 2019 (COVID-19), including Europe.

The COVID-19 pandemic is rapidly evolving, and outbreak investigations are ongoing. ECDC is closely monitoring this outbreak, providing risk assessments, public health guidance, and advice on response activities to EU Member States and the EU Commission.

- Latest situation update, epidemiological curve and global distribution

Example: data Input from the Web

- we download an EXCEL file
- we use the following packages: lubridate, curl and readxl

```
url <-  "https://www.ecdc.europa.eu/sites/default/files/documents/"
fname <- "COVID-19-geographic-disbtribution-worldwide-"
date <- lubridate::today() - 1
ext = ".xlsx"
target <- paste(url, fname, date, ext, sep="")
message("target:␣", target)

tmp_file <- tempfile("data", "/tmp", fileext=ext)
tmp <- curl::curl_download(target, destfile=tmp_file)
```

- data are imported in a tibble data structure

```
(data <- readxl::read_xlsx(tmp_file))
%> A tibble: 6,012 x 8
%>   DateRep               Day Month  Year Cases Deaths Countries.  GeoId
%>   <dttm>              <dbl> <dbl> <dbl> <dbl>  <dbl> <chr>       <chr>
%> 1 2020-03-21 00:00:00   21     3  2020     2      0 Afghanistan   AF
%> 2 2020-03-20 00:00:00   20     3  2020     0      0 Afghanistan   AF
%> 3 2020-03-19 00:00:00   19     3  2020     0      0 Afghanistan   AF
%> 4 2020-03-18 00:00:00   18     3  2020     1      0 Afghanistan   AF
%> 5 2020-03-17 00:00:00   17     3  2020     5      0 Afghanistan   AF
%> 6 2020-03-16 00:00:00   16     3  2020     6      0 Afghanistan   AF
%> 7 2020-03-15 00:00:00   15     3  2020     3      0 Afghanistan   AF
%> 8 2020-03-11 00:00:00   11     3  2020     3      0 Afghanistan   AF
%> 9 2020-03-08 00:00:00    8     3  2020     3      0 Afghanistan   AF
%>10 2020-03-02 00:00:00    2     3  2020     0      0 Afghanistan   AF
% ... with 6,002 more rows
```

Homework

- learn how to extract and manipulate data from the imported tibble
- in the next lecture we will see how to represent data with histograms and plots