



# Data rectangling:

*a journey from **JSON** to **CSV***

---

Muhammad Aswan Syahputra

# About me

- Sensory scientist @ [Sensolution.ID](#)
- Instructor @ [R Academy Telkom University](#)
- Initiator of [Komunitas R Indonesia](#) (est. 13 August 2016)
- 📁 : [sensehubr](#), [nusandata](#), [bandungjuara](#), [prakiraan](#), etc
- 💻 : [sensehub](#), [thermostats](#), [aquastats](#), [bcrp](#), [bandungjuara](#), etc
- 📩 : [aswansyahputra](#)
- 🐦 : [aswansyahputra\\_](#)
- 🐧 : [aswansyahputra](#)
- 💼 : [aswansyahputra](#)



# Know your neighbours!

- *Who are you?*
- *What do you do with data?*
- *How do you describe your experience with R?*

# Know your neighbours!

- *Who are you?*
- *What do you do with data?*
- *How do you describe your experience with R?*

03 : 00

**Let's play with some basics!**



# Let's play with some basics!



```
(x1 <- "useR! Yogyakarta")
#> [1] "useR! Yogyakarta"
(x2 <- TRUE)
#> [1] TRUE
(x3 <- 1.43)
#> [1] 1.43
(x4 <- 1L:5L)
#> [1] 1 2 3 4 5
```

Can you guess the **type** of **x1**, **x2**, **x3**, and **x4**? How about their **length**?

# Let's play with some basics!



```
(x1 <- "useR! Yogyakarta")
#> [1] "useR! Yogyakarta"
(x2 <- TRUE)
#> [1] TRUE
(x3 <- 1.43)
#> [1] 1.43
(x4 <- 1L:5L)
#> [1] 1 2 3 4 5
```

Can you guess the **type** of **x1**, **x2**, **x3**, and **x4**? How about their **length**?

```
typeof(x1)
#> [1] "character"
length(x1)
#> [1] 1
typeof(x2)
#> [1] "logical"
length(x2)
#> [1] 1
typeof(x3)
#> [1] "double"
length(x3)
#> [1] 1
```

What about **x4**? 😬

**How to combine  $x_1$ ,  $x_2$ ,  $x_3$ ,  
and  $x_4$  without losing their  
properties?**

# The use of c()

```
(xs_c ← c(x1, x2, x3, x4))  
#> [1] "useR! Yogyakarta" "TRUE"  
#> [4] "1"                 "2"  
#> [7] "4"                 "5"
```

It seems off, doesn't it? 😱 Can you explain? 😕

# The use of c()

```
(xs_c ← c(x1, x2, x3, x4))  
#> [1] "useR! Yogyakarta" "TRUE"  
#> [4] "1"                 "2"  
#> [7] "4"                 "5"
```

It seems off, doesn't it? 😱 Can you explain? 😕

Let's check it! 🔎

```
length(xs_c)  
#> [1] 8  
typeof(xs_c)  
#> [1] "character"
```

length ✗, type ? What's happening? 🤔

# The use of `list()`

```
(xs_list ← list(x1, x2, x3, x4))
#> [[1]]
#> [1] "useR! Yogyakarta"
#>
#> [[2]]
#> [1] TRUE
#>
#> [[3]]
#> [1] 1.43
#>
#> [[4]]
#> [1] 1 2 3 4 5
```

Hmm, not so familiar but it seems to be what we wanted, right? 😊

# The use of `list()`

```
(xs_list ← list(x1, x2, x3, x4))
#> [[1]]
#> [1] "useR! Yogyakarta"
#>
#> [[2]]
#> [1] TRUE
#>
#> [[3]]
#> [1] 1.43
#>
#> [[4]]
#> [1] 1 2 3 4 5
```

Let's also check it! 🔎

```
length(xs_list)
#> [1] 4
typeof(xs_list)
#> [1] "list"
```

`length` ✓, `type` ? What is `list`? 🤔

Hmm, not so familiar but it seems to be what we wanted, right? 😊

**Hold on!**



# Hold on!



How to check if the **type** and **length** of each element are preserved?

# The good old **for** loop ❤️

```
types_xs_c ← vector("character", length = length(xs_c))
for (i in seq_along(xs_c)) {
  types_xs_c[[i]] ← typeof(xs_c[[i]])
}
types_xs_c
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
```

# The good old **for** loop ❤️

```
types_xs_c ← vector("character", length = length(xs_c))
for (i in seq_along(xs_c)) {
  types_xs_c[[i]] ← typeof(xs_c[[i]])
}
types_xs_c
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
```

```
lengths_xs_c ← vector("integer", length = length(xs_c))
for (i in seq_along(xs_c)) {
  lengths_xs_c[[i]] ← length(xs_c[[i]])
}
lengths_xs_c
#> [1] 1 1 1 1 1 1 1 1 1
```

# The good old **for** loop ❤️

```
types_xs_c ← vector("character", length = length(xs_c))
for (i in seq_along(xs_c)) {
  types_xs_c[[i]] ← typeof(xs_c[[i]])
}
types_xs_c
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
```

```
lengths_xs_c ← vector("integer", length = length(xs_c))
for (i in seq_along(xs_c)) {
  lengths_xs_c[[i]] ← length(xs_c[[i]])
}
lengths_xs_c
#> [1] 1 1 1 1 1 1 1 1 1
```

How would you perform the same procedure for `xs_list`? 🤔 Save your results as `types_xs_list` and `lengths_xs_list`!

# Let me introduce you to functional 😎

```
vapply(xs_c, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
vapply(xs_c, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 1 1 1 1 1 1
```

# Let me introduce you to functional 😎

```
vapply(xs_c, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
vapply(xs_c, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 1 1 1 1 1 1
```

```
vapply(xs_list, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "logical"   "double"    "integer"
vapply(xs_list, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 5
```

# Let me introduce you to functional 😎

```
vapply(xs_c, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
vapply(xs_c, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 1 1 1 1 1 1
```

```
vapply(xs_list, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "logical"   "double"    "integer"
vapply(xs_list, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 5
```

Ok, it surely looks simpler but still... 😐

# Let me introduce you to functional 😎

```
vapply(xs_c, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
vapply(xs_c, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 1 1 1 1 1 1
```

```
vapply(xs_list, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "logical"    "double"     "integer"
vapply(xs_list, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 5
```

Ok, it surely looks simpler but still... 😐

```
library(purrr)
map_chr(xs_list, typeof)
#> [1] "character" "logical"    "double"     "integer"
map_int(xs_list, length)
#> [1] 1 1 1 5
```

# Let me introduce you to functional 😎

```
vapply(xs_c, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "character" "character" "character" "character" "character"
#> [7] "character" "character"
vapply(xs_c, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 1 1 1 1 1 1
```

```
vapply(xs_list, typeof, character(1), USE.NAMES = FALSE)
#> [1] "character" "logical"   "double"    "integer"
vapply(xs_list, length, integer(1), USE.NAMES = FALSE)
#> [1] 1 1 1 5
```

Ok, it surely looks simpler but still... 🤨

```
library(purrr)
map_chr(xs_list, typeof)
#> [1] "character" "logical"   "double"    "integer"
map_int(xs_list, length)
#> [1] 1 1 1 5
```

So much simpler and better, isn't it? 🎉

list resembles JSON very  
much!

list resembles JSON very  
much!

Have a look at following comparison using  
subset of billionaires data



## Raw JSON file

```
cd data-raw
cat billionaires_small.json
#> [
#>   {
#>     "wealth": {
#>       "worth in billions": [3.6],
#>       "how": {
#>         "category": ["Traded Sectors"],
#>         "from emerging": [true],
#>         "industry": ["Consumer"],
#>         "was political": [false],
#>         "inherited": [true],
#>         "was founder": [true]
#>       },
#>       "type": ["founder non-finance"]
#>     },
#>     "company": {
#>       "sector": ["agricultural products"]
#>       "founded": [1929],
#>       "type": ["new"],
#>       "name": ["J.R. Simplot Company"],
#>       "relationship": ["founder"]
#>     },
#>     "rank": 1151
#>   }
#> ]
```

## Raw JSON file

```
cd data-raw
cat billionaires_small.json
#> [
#>   {
#>     "wealth": {
#>       "worth in billions": [3.6],
#>       "how": {
#>         "category": ["Traded Sectors"],
#>         "from emerging": [true],
#>         "industry": ["Consumer"],
#>         "was political": [false],
#>         "inherited": [true],
#>         "was founder": [true]
#>       },
#>       "type": ["founder non-finance"]
#>     },
#>     "company": {
#>       "sector": ["agricultural products"]
#>       "founded": [1929],
#>       "type": ["new"],
#>       "name": ["J.R. Simplot Company"],
#>       "relationship": ["founder"]
#>     },
#>     "rank": 1151
#>   }
#> ]
```

## When imported to R

```
str(billionaires_small, max.level = 3)
#> List of 3
#> $ :List of 7
#>   ..$ wealth      :List of 3
#>   .. ..$ worth in billions: num 3.6
#>   .. ..$ how        :List of 6
#>   .. ..$ type       : chr "founder"
#>   ..$ company     :List of 5
#>   .. ..$ sector     : chr "agricultural"
#>   .. ..$ founded    : int 1929
#>   .. ..$ type       : chr "new"
#>   .. ..$ name        : chr "J.R. Simplot"
#>   .. ..$ relationship: chr "founder"
#>   ..$ rank         : int 115
#>   ..$ location     :List of 4
#>   .. ..$ gdp        : num 1.06e+13
#>   .. ..$ region      : chr "North America"
#>   .. ..$ citizenship : chr "United States"
#>   .. ..$ country code: chr "USA"
#>   ..$ year         : int 2001
#>   ..$ demographics:List of 2
#>   .. ..$ gender      : chr "male"
#>   .. ..$ age          : int 92
#>   ..$ name         : chr "John Simplot"
```

How to extract the  
element(s) of a **list**?

# From a billionaire, extract info



```
library(purrr)
pluck(billionaires_small, 1) # you can also
#> $wealth
#> $wealth$`worth in billions`
#> [1] 3.6
#>
#> $wealth$how
#> $wealth$how$category
#> [1] "Traded Sectors"
#>
#> $wealth$how$`from emerging`
#> [1] TRUE
#>
#> $wealth$how$industry
#> [1] "Consumer"
#>
#> $wealth$how$`was political`
#> [1] FALSE
#>
#> $wealth$how$inherited
#> [1] TRUE
#>
```

# From a billionaire, extract info



```
library(purrr)
pluck(billionaires_small, 1) # you can also
#> $wealth
#> $wealth`worth in billions`
#> [1] 3.6
#>
#> $wealth$how
#> $wealth$how$category
#> [1] "Traded Sectors"
#>
#> $wealth$how$`from emerging`
#> [1] TRUE
#>
#> $wealth$how$industry
#> [1] "Consumer"
#>
#> $wealth$how$`was political`
#> [1] FALSE
#>
#> $wealth$how$inherited
#> [1] TRUE
#>
```

```
pluck(billionaires_small, 1, "name") # you c
#> [1] "John Simplot"

pluck(billionaires_small, 1, "rank")
#> [1] 115

pluck(billionaires_small, 1, "wealth", "wort
#> [1] 3.6
```

# From some billionaires, extract info

```
map(billionaires_small, pluck, "name")
#> [[1]]
#> [1] "John Simplot"
#>
#> [[2]]
#> [1] "Banyong Lamsam"
#>
#> [[3]]
#> [1] "Richard Farmer"

map(billionaires_small, pluck, "wealth", "wo
#> [[1]]
#> [1] 3.6
#>
#> [[2]]
#> [1] 2.5
#>
#> [[3]]
#> [1] 1.8
```

# From some billionaires, extract info 💰

```
map(billionaires_small, pluck, "name")
#> [[1]]
#> [1] "John Simplot"
#>
#> [[2]]
#> [1] "Banyong Lamsam"
#>
#> [[3]]
#> [1] "Richard Farmer"

map(billionaires_small, pluck, "wealth", "wo
#> [[1]]
#> [1] 3.6
#>
#> [[2]]
#> [1] 2.5
#>
#> [[3]]
#> [1] 1.8
```

```
(billionaire_names ← map_chr(billionaires_s
#> [1] "John Simplot"   "Banyong Lamsam" "Ri
(billionaire_ranks ← map_int(billionaires_s
#> [1] 115 143 272
(billionaire_worth ← map_dbl(billionaires_s
#> [1] 3.6 2.5 1.8
```

**Awesome, `map()`  
provides a shortcut!  
Bye `pluck()`~ 🙌**

Yeay, we can extract some  
infos 💪

Yeay, we can extract some  
infos 💪

But, now they are scattered 😞

Of course you can combine them later using `data.frame()` or `tibble()`, but... 😢

Of course you can combine them later using `data.frame()` or `tibble()`, but... 😢

```
data.frame(  
  name = billionaire_names,  
  rank = billionaire_ranks,  
  worth_in_billions = billionaire_worth,  
  stringsAsFactors = FALSE  
)  
#>           name rank worth_in_billions  
#> 1  John Simplot  115          3.6  
#> 2 Banyong Lamsam  143          2.5  
#> 3 Richard Farmer  272          1.8
```

Of course you can combine them later using `data.frame()` or `tibble()`, but... 😢

```
data.frame(  
  name = billionaire_names,  
  rank = billionaire_ranks,  
  worth_in_billions = billionaire_worth,  
  stringsAsFactors = FALSE  
)  
#>       name rank worth_in_billions  
#> 1 John Simplot 115      3.6  
#> 2 Banyong Lamsam 143      2.5  
#> 3 Richard Farmer 272      1.8
```

```
library(tibble)  
tibble(  
  name = billionaire_names,  
  rank = billionaire_ranks,  
  worth_in_billions = billionaire_worth  
)  
#> # A tibble: 3 x 3  
#>   name           rank worth_in_billions  
#>   <chr>          <int>            <dbl>  
#> 1 John Simplot    115            3.6  
#> 2 Banyong Lamsam   143            2.5  
#> 3 Richard Farmer   272            1.8
```

**Why don't we contain the `list` in `dataframe/tibble` in the first place?**

# Why don't we contain the **list** in **dataframe/tibble** in the first place?

Let's embrace **list-column** 🚫

```
library(tibble)
billionaires_small_df <-
  billionaires_small %>%
  enframe()
billionaires_small_df
#> # A tibble: 3 x 2
#>   name    value
#>   <int> <list>
#> 1     1 <named list [7]>
#> 2     2 <named list [7]>
#> 3     3 <named list [7]>
```

# Why don't we contain the list in dataframe/tibble in the first place?

Let's embrace **list-column** 🚫

```
library(tibble)
billionaires_small_df <-
  billionaires_small %>%
  enframe()
billionaires_small_df
#> # A tibble: 3 x 2
#>   name    value
#>   <int> <list>
#> 1     1 <named list [7]>
#> 2     2 <named list [7]>
#> 3     3 <named list [7]>
```

Now we can make use of **dplyr**, ain't it cool? 🙌

```
library(dplyr)
billionaires_small_df %>%
  mutate(
    name = map_chr(value, "name"),
    rank = map_int(value, "rank"),
    worth_in_billions = map_dbl(
      value,
      list("wealth", "worth in billions")))
) %>%
  select(-value)
#> # A tibble: 3 x 3
#>   name           rank worth_in_billions
#>   <chr>          <int>            <dbl>
#> 1 John Simplot    115             3.6
#> 2 Banyong Lamsam  143             2.5
#> 3 Richard Farmer  272             1.8
```

# Let's practice!



- Open your RStudio, then install *usethis* package
- Once succeed, run *usethis :: use\_course("aswansyahputra/kpdr\_jogja")*
- Follow the instructions and new RStudio session will be automatically opened
- Please open *hands-on.Rmd* and read the instructions thoroughly



# Thank you!

👉 [t.me/GNURIndonesia](https://t.me/GNURIndonesia)

🌐 [r-indonesia.id](http://r-indonesia.id)

✉️ [info@r-indonesia.id](mailto:info@r-indonesia.id)