Session 12
Data Wrangling 2

R for Social Data Science

Jeffrey Ziegler, PhD

Assistant Professor in Political Science & Data Science
Trinity College Dublin

Fall 2022

Road map for today

Last time:

- Data frames in base R
- Data input and output

This time:

- Alternatives to data frames
- ‘tidyverse‘ packages
- Working with tabular data
- Summary statistics

Alternatives to data frame

- Two major alternatives to data frames are:
  - ‘tibble‘ from ‘tibble‘ package (part of ‘tidyverse‘ package ecosystem)
  - ‘data.table‘ from ‘data.table‘
- ‘tibble‘ provides features enhancing user experience (readability, ease of manipulation)
- ‘data.table‘ provides speed

Data table - fast data frame

- As opposed to data frames, data tables are updated by reference
- Frees up a lot of RAM for big data!
- Provides low-level parallelism
- SQL-like operations for data manipulation
- Has no external dependencies (other than base R itself)

```
dt <- data.table::data.table(
    x = 1:4,
            y = c("a", "b", "c", "d"),
            z = c(TRUE, FALSE, FALSE, TRUE)
)
```

```
x y     z
1: 1 a  TRUE
2: 2 b FALSE
3: 3 c FALSE
4: 4 d  TRUE
```

‘tidyverse‘ packages

- ‘tidyverse‘ package ecosystem - rich collection of data science packages
- Designed with consistent interfaces and generally higher usability than base R function
- Notable packages:
  - ‘readr‘ - data input/output (also ‘readxl‘ for spreadsheets, ‘haven‘ for SPSS/Stata)
  - ‘dplyr‘ - data manipulation (also ‘tidyr‘ for pivoting)
  - ‘ggplot2‘ - data visualisation
  - ‘lubridate‘ - working with dates and time
  - ‘tibble‘ - enhanced data frame

```
install.packages("tidyverse")
```

Tibble: user-friendly data frame

- Tibbles are designed to be backward compatible with base R data frames
- Console printing of tibbles is cleaner (prettified, only first 10 rows by default)
- Tibbles can have columns that themselves contain lists as elements
- Tibbles can be created with ‘tibble::tibble()‘ function
- Or objects can be coerced into a tibble using ‘tibble::as_tibble()‘ function

Tibble: user-friendly data frame

```r
tb <- tibble::tibble(
    x = 1:4,
        y = c("a", "b", "c", "d"),
        z = c(TRUE, FALSE, FALSE, TRUE)
)
tb
```

```
# A tibble: 4 × 3
x    6y     z
<int> <chr> <lgl>
1     1 a     TRUE
2     2 b     FALSE
3     3 c     FALSE
4     4 d     TRUE
```

Tibbles work (mostly) like data frames

```
str(tb)
```

```
tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
$ x: int [1:4] 1 2 3 4
$ y: chr [1:4] "a" "b" "c" "d"
$ z: logi [1:4] TRUE FALSE FALSE TRUE
```

```
dim(tb)
```

```
[1] 4 3
```

```
tb[c("x", "z")]
```

```
# A tibble: 4 × 2
x z
<int> <lgl>
1     1 TRUE
2     2 FALSE
3     3 FALSE
4     4 TRUE
```

```
tb[tb$y=="b", ]
```

```
# A tibble: 1 × 3
x y     z
<int> <chr> <lgl>
1     2 b     FALSE
```

Data manipulation with ‘dplyr‘

- ‘dplyr‘ - core package for data manipulation in ‘tidyverse‘
- Principal functions are:
  - ‘filter()‘ - subset rows from data
  - ‘mutate()‘ - add new/modify existing variables
  - ‘rename()‘ - rename existing variable
  - ‘select()‘ - subset columns from data
  - ‘arrange()‘ - order data by some variable
- For data summary:
  - ‘group_by()‘ - aggregate data by some variable
  - ‘summarise()‘ - create a summary of aggregated variables

```
library("dplyr")
```

Subsetting with 'dplyr'

```
dplyr::filter(tb, y == 'b', z == FALSE)
```

```
# A tibble: 1 × 3
x y     z
<int> <chr> <lgl>
1     2 b     FALSE
```

```
# Note that dplyr functions do not require enquoted variable names
dplyr::select(tb, x, z)
```

```
#  A tibble: 4 × 2
x z
<int> <lgl>
1     1 TRUE
2     2 FALSE
3     3 FALSE
4     4 TRUE
```

```
# We can also use helpful tidyselect functions for more complex rules
dplyr::select(tb, tidyselect::starts_with('x'))
```

```
# A tibble: 4 × 1
x
<int>
1     1
2     2
3     3
4     4
```

Renaming/modifying columns with 'dplyr'

```r
# Data is not modified in-place, you need to re-assign the results
tb <- dplyr::rename(tb, random = x)
dplyr::mutate(tb, random_8plus = ifelse(random >= 3, TRUE, FALSE))
```

```
# A tibble: 4 × 4
random y     z     random_8plus
<int> <chr> <lgl> <lgl>
1      1 a     TRUE  FALSE
2      2 b     FALSE FALSE
3      3 c     FALSE TRUE
4      4 d     TRUE  TRUE
```

‘%>%‘ OPERATOR

- Users of ‘tidyverse‘ packages are encouraged to use pipe operator ‘%>%‘
- Allows to chain data transformations without creating intermidate variables
- Passes result of previous operation as a first first argument to next
- Base R now also includes its own pipe operator ‘|>‘ but it’s still relatively uncommon

```
<result> <- <input> %>%
<function_name>(., arg_1, arg_2, ..., arg_n)

<result> <- <input> %>%
<function_name>(arg_1, arg_2, ..., arg_n)
```

'%>%' OPERATOR EXAMPLE

```
tb <- tb %>%
  dplyr::mutate(random_2 = rnorm(4)) %>%
  dplyr::filter(z == FALSE)
tb
```

```
# A tibble: 2 × 4
random y     z     random_2
<int> <chr> <lgl>    <dbl>
1      2 b     FALSE    0.195
2      3 c     FALSE    1.68
```

‘%>%‘ operator vs built-in ‘|>‘ operator

- Since R version 4.1.0 (mid-2021), there is a built-in ‘|>‘ pipe operator

```
# Pipe %>% can also be used with non-dplyr functions
tb$y %>% .[2]
```

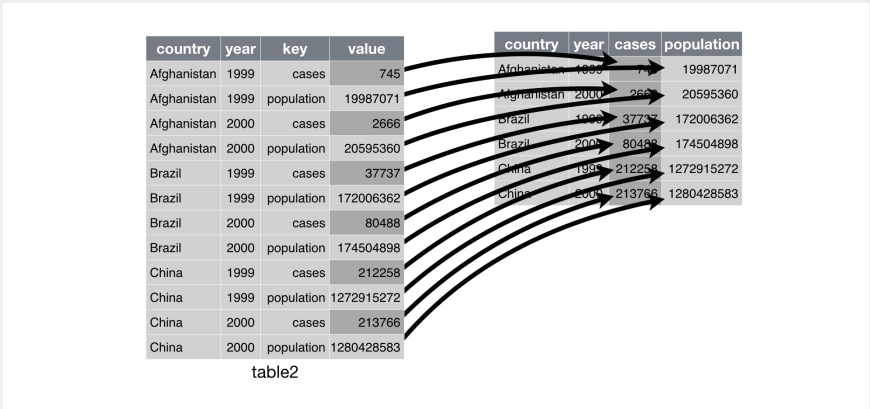
```
[1] "c"
```

```
# Base R pipe operator |> is more restrictive (e.g. tb$x |>
    ‘[‘(2) doesn’t work)
tb |> nrow()
```

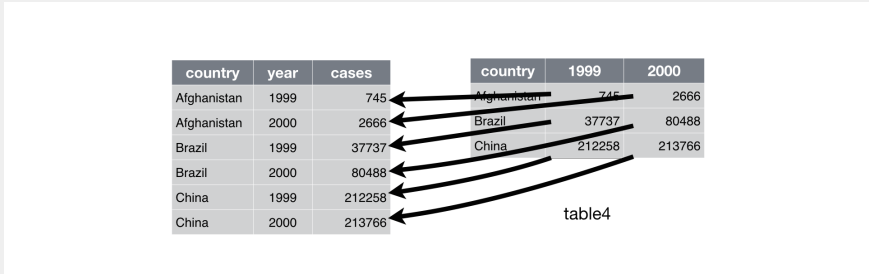
```
[1] 2
```

Pivoting data

- Sometimes you want to pivot you data by:
  - Spreading some variable across columns (‘tidyr::pivot_wider()‘)
  - Gathering some columns in a variable pair (‘tidyr::pivot_longer()‘)



(a) pivot_wider()



(b) pivot_longer()

Source: R for Data Science

Pivoting data examples

```r
tb2 <- tibble::tibble(
  country = c("Afghanistan", "Brazil"),
  `1999` = c(745, 2666),
  `2000` = c(37737, 80488)
)
tb2
```

```
# A tibble: 2 × 3
country     `1999` `2000`
<chr>        <dbl>  <dbl>
1 Afghanistan    745  37737
2 Brazil        2666  80488
```

```r
# Note that pivoting functions come 'tidyr' package
tb2 <- tb2 %>%
  tidyr::pivot_longer(cols = c("1999", "2000"), names_to = "year", values_to = "cases")
tb2
```

```
# A tibble: 4 × 3
country     year  cases
<chr>       <chr> <dbl>
1 Afghanistan 1999    745
2 Afghanistan 2000  37737
3 Brazil      1999   2666
4 Brazil      2000  80488
```

Pivoting data examples

```
1 tb2 <-tb2 %>%
2   tidyr::pivot_wider(names_from="year", values_from="cases")
3 tb2
```

```
# A tibble: 4 × 3
country     year  cases
<chr>       <chr> <dbl>
1 Afghanistan 1999    745
2 Afghanistan 2000  37737
3 Brazil      1999   2666
4 Brazil      2000  80488
```

Summarizing numeric variables

```r
# load local data in as dataframe
kaggle2021 <- read.csv("../datasets/kaggle_survey_2021_responses.
    csv", stringsAsFactors = F)[-1,]
summary(as.numeric(kaggle2021[,1]))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    120     443     656   11055    1038 2488653
```

Summarizing categorical variables

```
# table() function is flexible, can tabulate a single variable and do crosstabs
table(kaggle2021[3])
```

```
Man   Nonbinary       Prefer not to say Prefer to self-describe    Woman
20598      88                     355     42          4890
```

```
# Wrapping it inside prop.table() gives proportions of each category
prop.table(table(kaggle2021[3]))
```

```
Man   Nonbinary   Prefer not to say Prefer to self-describe   Woman
0.793054326    0.003388134      0.013668040     0.001617064     0.188272437
```

```
# Wrapping it inside sort() gives value sorting, as opposed to alphabetic (or facto
    levels)
sort(table(kaggle2021[3]), decreasing = TRUE)[1]
```

```
Man
20598
```

Tutorial - Tabulation and Summary Stats

- Load 'kaggle_survey_2021_responses.csv' dataframe from GitHub repository to global environment
- Do so from the url, and from the local file
- Consider country of residence reported by respondents (question Q3)
- Make sure you can select the column both using both it name and index
- Calculate the percentages of top 3 countries of residence in the sample

Tutorial - Pivoting Tables

- To simplify working with the dataset, let's create a unique id for each respondent (you can use 'seq_along()' function in combination with any other variable to do so)
- Finally, use 'pivot_wider' function from 'tidyr' package to create a separate column for each age group
- If original pivoting produced columns that are populated by values of the categorical variable and 'NA''s, use 'mutate' function to replace them with 0's and 1's
- Finally, use 'pivot_longer' function to convert this representation of the dataset back into its original form
- You might also need to use 'dplyr::filter()' function to remove redundant rows

Overview

This week:

- Data frames in base R
- Data input and output
- Alternatives to data frames
- ‘tidyverse‘ packages
- Working with tabular data
- Summary statistics

Next week:

- Data visualization