Data Wrangling

Fundamentals of Computing and Data Display

Christoph Kern Ruben Bach[1]

(c.kern, r.bach)@uni-mannheim.de

[1]Slides draw on Grolemund, G (2018). Master the Tidyverse: An Introduction to R for Data Science. https://github.com/rstudio/master-the-tidyverse

Outline

Introduction

Base R

- "R is a free software environment for statistical computing and graphics."
- Focus on flexibility and the ability to program
- Leans towards developing rather than basic data analysis

```
> # Collapsing data the R way
>   cc_agg <- aggregate(cc_sub[, 2:5],
>   by = list(cc_sub$district),
>   FUN = sum)
```

The tidyverse

"A collection of modern R packages that share common philosophies, embed best practices, and are designed to work together"

`https://www.tidyverse.org/`

The tidyverse

install.packages("tidyverse")	library("tidyverse")
install.packages("ggplot2")	library("ggplot2")
install.packages("dplyr")	library("dplyr")
install.packages("tidyr")	library("tidyr")
install.packages("readr")	library("readr")
install.packages("purrr")	library("purrr")
install.packages("tibble")	library("tibble")
install.packages("stringr")	library("stringr")
install.packages("forcats")	library("forcats")
install.packages("hms")	
install.packages("lubridate")	
install.packages("DBI")	
install.packages("haven")	
install.packages("httr")	
install.packages("jsonlite")	
install.packages("readxl")	
install.packages("rvest")	
install.packages("xml2")	
install.packages("modelr")	
install.packages("broom")	

The tidyverse

Core tidyverse

- `ggplot2`, for data visualisation
- `dplyr`, for data manipulation
- `tidyr`, for data tidying
- `readr`, for data import
- `purrr`, for functional programming
- `tibble`, for tibbles
- `stringr`, for strings
- `forcats`, for factors

Tidy data

Many ways to structure the same underlying data

- R prefers **tidy data**
  1. Each variable is in its own column
  2. Each case is in its own row
  3. Each value is in its own cell
- Advantages of tidy data
  1. Consistent way of storing data eases data transformations and reshaping
  2. Utilizes vectorized nature of many (built-in) R functions
  3. Tidyverse designed to work with tidy data

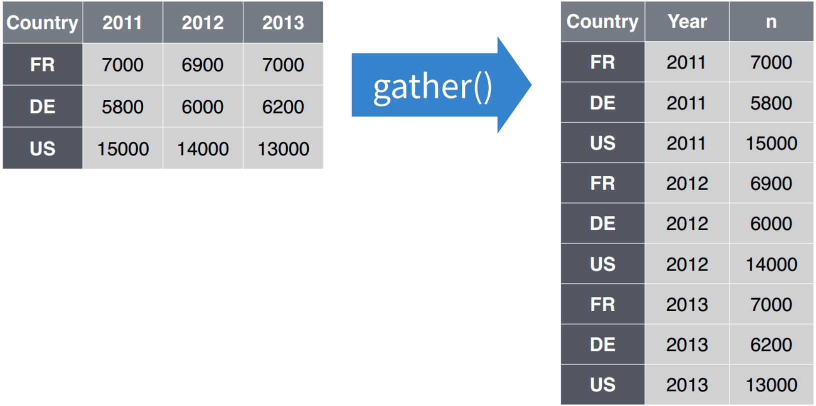
tibble

The natural habitat of tidy data

- A Tibble: Tidyverse version of a `data.frame`
  - Print method shows only the first 10 rows
  - Subsetting with [returns a tibble, [[, $ return vectors
- Converting into tibbles: `as_tibble()`
  - Prevents type conversion, renaming, row.names, error prone recycling
  - Tidyverse functions typically return tibbles
- Enhanced `str()`: `glimpse()`
- `vignette("tibble")`

tidyr

Figure: Reshape from wide to long format



Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

```
> # Reshape with gather()
> gather(cases, key = "year", value = "n", 2:4)
```

tidyr

Figure: Reshape from wide to long format

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

tidyr

Figure: Reshape from wide to long format

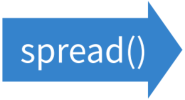
Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

Country	Year	n
FR	2011	7000
DE	2011	5800
US	2011	15000
FR	2012	6900
DE	2012	6000
US	2012	14000
FR	2013	7000
DE	2013	6200
US	2013	13000

tidyr

Figure: Reshape from long to wide format

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



city	large	small
New York	23	14
London	22	16
Beijing	121	56

```
> # Reshape with spread()
> spread(pollution, key = size, value = amount)
```

tidyr

Figure: Reshape from long to wide format

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

tidyr

Figure: Reshape from long to wide format

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	large	small
New York	23	14
London	22	16
Beijing	121	56

tidyr

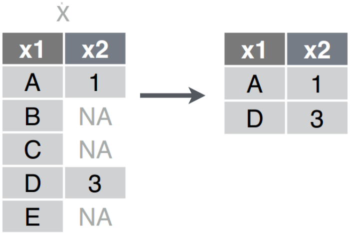
Figure: Split a column by dividing values at a specific character

country <chr>	year <int>	codes <chr>	new <chr>	type <chr>	sexage <chr>	n <int>
Afghanistan	1980	new_sp_m014	new	sp	m014	NA
Afghanistan	1981	new_sp_m014	new	sp	m014	NA
Afghanistan	1982	new_sp_m014	new	sp	m014	NA
Afghanistan	1983	new_sp_m014	new	sp	m014	NA
Afghanistan	1984	new_sp_m014	new	sp	m014	NA
Afghanistan	1985	new_sp_m014	new	sp	m014	NA
Afghanistan	1986	new_sp_m014	new	sp	m014	NA
Afghanistan	1987	new_sp_m014	new	sp	m014	NA
Afghanistan	1988	new_sp_m014	new	sp	m014	NA
Afghanistan	1989	new_sp_m014	new	sp	m014	NA

```
> # Split column with separate()
> separate(who$codes, into = c("new", "type", "sexage"), sep = "_"))
> # separate(who$codes, c("new", "type", "sexage"), sep = c(4, 7))
```

tidyr

Figure: Drop rows that contain NA's (in the specified columns)

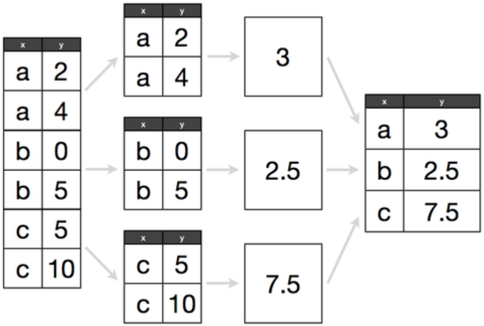



```
> # Listewise deletion with drop_na()
> drop_na(x, x2)
> # drop_na(x)
```

Split-Apply-Combine

1. "split" problem into smaller pieces
2. "apply" function to each piece separatly
3. "combine" results back together

- In R, loops are not ideal for this problem
  - Slow, hard to read, much to code
- Vectorized functions, `plyr`, `dplyr`
  - Faster, easier to read, draw e.g. on C++



apply

The apply family (Base R)

- `apply(object, n, function)`
    - Apply a function to the rows ($n = 1$) or columns ($n = 2$) of an object
    - Input: array, matrix
    - Output: array, list
- "List apply": `lapply(object, function)`
    - Apply a function to the columns of an object
    - Input: list, data.frame
    - Output: list
- "Simplified apply": `sapply(object, function)`
    - Apply a function to the columns of an object
    - Input: list, data.frame
    - Output: vector, matrix
- `vapply()`, `mapply()`, `rapply()`

apply

Apply by groups (Base R)

- `tapply(object, factor, function)`
  - Apply a function to a vector grouped by factor levels
  - Input: vector
  - Output: array
- `by(object, factor, function)`
  - Apply a function to the columns of an object by factor levels
  - Input: data.frame, matrix
  - Output: list, array

```
> # Apply example from Advanced R book
> fix_missing <- function(x) {
>   x[x == -99] <- NA
>   x
> }
> df[] <- lapply(df, fix_missing)
```

plyr

Split-apply-combine with **`plyr`**

- Consistent naming schema, parallel processing
- `d*ply(.data, .variables, .fun, ...)`
  - First argument is data to be processed
  - Second argument describes how to split up input
  - Third argument is processing function

Table: The plyr family

	Output			
Input	Array	Data frame	List	Discarded
Array	`aaply`	`adply`	`alply`	`a_ply`
Data frame	`daply`	`ddply`	`dlply`	`d_ply`
List	`laply`	`ldply`	`llply`	`l_ply`

dplyr

A fast `plyr` for data frames: **dplyr**

- Optimized for speed
- Consistent function names and interfaces
- Results are returned as tibbles
- Powerful in connection with piping

`https://dplyr.tidyverse.org/`

dplyr

Base R data transformation workflows

1. Save each intermediate step as a new object
2. Overwrite the original object many times
3. Compose functions

dplyr

- Provides the pipe operator %>% to pass results to next function
  - Run a sequence of transformations in one code block
  - Readable structure of data operations
  - Easy to modify, add and remove steps
- Left-hand side is piped in as first argument of right-hand side

```
> who %>%
>   gather("codes", "n", 5:60) %>%
>   separate(codes, into = c("new", "type", "sexage"), sep = "_")
```

dplyr

Base R data transformation workflows

1. Save each intermediate step as a new object
2. Overwrite the original object many times
3. Compose functions

dplyr

- Provides the pipe operator **%>%** to pass results to next function
  - Run a sequence of transformations in one code block
  - Readable structure of data operations
  - Easy to modify, add and remove steps
- Left-hand side is piped in as first argument of right-hand side

```
> who %>%
>   gather("codes", "n", 5:60) %>%
>   separate(codes, into = c("new", "type", "sexage"), sep = "_")
```

dplyr

Other pipe operators from **magrittr**

- **%T>%**
  - Returns the left-hand side value (not right-hand side result)
- **%$%**
  - Allows to refer to individual vectors (w/o data.frame context)
- **%<>%**
  - Assign result of pipeline to object

```
# equivalent to who %<>% gather...
> who <- who %>%
>   gather("codes", "n", 5:60) %>%
>   separate(codes, into = c("new", "type", "sexage"), sep = "_")
```

dplyr

Figure: Select columns

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

name	prop
John	0.0815
William	0.0805
James	0.0501
Charles	0.0451
Garrett	0.0001
John	0.081

```
> # Select columns with select()
> select(babynames, name, prop)
> # select range of cols with :, exclude cols with -
```

dplyr

Figure: Filter rows

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001

```
> # Filter rows with filter()
> filter(babynames, name == "Garrett" & year == 1880)
```

dplyr

Table: Operators

?comparison	?Logic
x < y	a & b
x > y	a | b
x == y	xor(a,b)
x <= y	!a
x >= y	
x != y	
x %in% y	
is.na(x)	
!is.na(x)	

dplyr

Figure: Order rows

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1881	M	John	8769	0.081
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001

```
> # Order rows with arrange()
> arrange(babynames, desc(n))
```

dplyr

Figure: Summarize data

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

total	max
127538	99680

```
> # Summarize data with summarise()
> babynames %>%
>   summarise(total = sum(n), max = max(n))
```

dplyr

Figure: Split-apply-combine with dplyr

city	particle size	amount (μg/m^3)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount (μg/m^3)
New York	large	23
New York	small	14

London	large	22
London	small	16

Beijing	large	121
Beijing	small	56

city	mean	sum	n
New York	18.5	37	2
London	19.0	38	2
Beijing	88.5	177	2

```
> # Split data by groups with group_by()
> pollution %>%
>   group_by(city) %>%
>   summarise(mean = mean(amount), sum = sum(amount), n = n())
```

dplyr

Figure: Create new variables

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop	percent
1880	M	John	9655	0.0815	8.15
1880	M	William	9532	0.0805	8.05
1880	M	James	5927	0.0501	5.01
1880	M	Charles	5348	0.0451	4.51
1880	M	Garrett	13	0.0001	0.01
1881	M	John	8769	0.081	8.1

```
> # Create new variables with mutate()
> babynames %>%
>   mutate(percent = round(prop*100, 2))
```

dplyr

Data analysis workflows

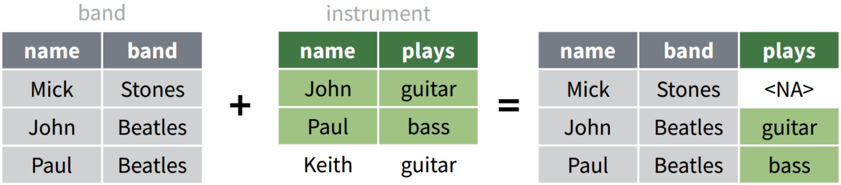
```
> # Temporal objects
> boys_2015 <- filter(babynames, year == 2015, sex == "M")
> boys_2015 <- select(boys_2015, name, n)
> boys_2015 <- arrange(boys_2015, desc(n))
> boys_2015
```

```
> # Nesting functions
> arrange(select(filter(babynames, year == 2015, sex == "M"), name, n), desc(n))
```

```
> # Piping
> babynames %>%
>   filter(year == 2015, sex == "M") %>%
>   select(name, n) %>%
>   arrange(desc(n))
```

joins

Figure: Merge data I




```
> # Merge data with left_join()
> band %>% left_join(instrument, by = "name")
```

joins

Figure: Merge data II

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

```
> # Merge data with right_join()
> band %>% right_join(instrument, by = "name")
```

joins

Figure: Merge data III

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

```
> # Merge data with full_join()
> band %>% full_join(instrument, by = "name")
```

joins

Figure: Merge data IV

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

```
> # Merge data with inner_join()
> band %>% inner_join(instrument, by = "name")
> # Adjust by if names do not match
> # band %>% inner_join(instrument2, by = c("name" = "artist"))
```

joins

Figure: Filter data I

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

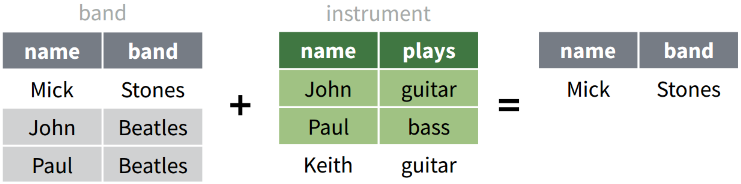
=

name	band
John	Beatles
Paul	Beatles

```
> # Filter data with semi_join()
> band %>% semi_join(instrument, by = "name")
```

joins

Figure: Filter data II




```
> # Filter data with anti_join()
> band %>% anti_join(instrument, by = "name")
```

APIs

Google Trends

- Jun, S.-P., Yoo, H. S., Choi, S. (2018). Ten years of research change using Google Trends: From the perspective of big data utilizations and applications. *Technological Forecasting and Social Change 130*, 69–87.
- Lazer, D., Kennedy, R., King, G., Vespignani, A. (2014). The Parable of Google Flu: Traps in Big Data Analysis. *Science 343*(6176), 1203–1205.
- Mellon, J. (2013). Where and When Can We Use Google Trends to Measure Issue Salience? *PS: Political Science & Politics, 46*(2), 280–290.
- Stephens-Davidowitz, S. (2014). The cost of racial animus on a black candidate: Evidence using Google search data. *Journal of Public Economics 118*, 26–40.
- Stephens-Davidowitz, S. and Varian, H (2015). A hands-on guide to Google data. `http://people.ischool.berkeley.edu/~hal/Papers/2015/primer.pdf`
- Vosen, S. and Schmidt, T. (2011). Forecasting Private Consumption: Survey-Based Indicators vs. Google Trends. *Journal of Forecasting 30*, 565–578.

APIs

Census API

- Overview: `https://www.census.gov/data/developers/data-sets.html`
- ACS 5-Year data: `https://www.census.gov/data/developers/data-sets/acs-5year.html`
- R package: `https://cran.r-project.org/web/packages/censusapi/vignettes/getting-started.html`

Resources

- Textbook
  - Boehmke, B. C. (2016). *Data Wrangling in R*. New York, NY: Springer.
- Paper
  - Wickham, H. (2014). Tidy Data. *Journal of Statistical Software 59*(10), 1–23.
  - Wickham, H. (2011). The Split-Apply-Combine Strategy for Data Analysis. *Journal of Statistical Software 40*(1), 1–29.
- Online Learning
  - `https://www.datacamp.com/community/tutorials/r-tutorial-apply-family`
  - `https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html`
- Cheatsheets
  - `https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf`
  - `https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf`